

Tool-Supported Development of Secure Wireless Sensor Networks

Von der Fakultät für Mathematik, Naturwissenschaften und Informatik
der Brandenburgischen Technischen Universität Cottbus

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
(Dr.-Ing.)

genehmigte Dissertation

vorgelegt von

Dipl.-Informatiker

Steffen Peter

geboren am 05.11.1977 in Hoyerswerda

Gutachter: Prof. Dr. rer. nat. Peter Langendörfer

Gutachter: Prof. Dr. rer. nat. Claus Lewerentz

Gutachter: Prof. Dr. rer. nat. Dirk Westhoff

Tag der mündlichen Prüfung: 13. Dez. 2011

Contents

Contents	1
Abstract	5
Kurzfassung	5
1 Introduction	7
1.1 Research Context	7
1.2 Research Objectives	9
1.3 Research Contributions	10
1.4 Structure of this Thesis	11
2 Wireless Sensor Networks	15
2.1 Application Space of WSNs	15
2.1.1 Examples	15
2.1.2 A WSN Taxonomy	18
2.2 Sensor Node Hardware	20
2.2.1 Commercial Sensor Node Platforms	20
2.2.2 Application-specific Sensor Nodes	22
2.2.3 General System Architecture	24
2.3 Sensor Node Software	26
2.3.1 Operating System	26
2.3.2 Protocol Stack	27
2.4 Conclusions and Implication for the Thesis	29
3 State of the Art in WSN System Engineering	31
3.1 WSN Programming Methodologies	31
3.1.1 General Methodologies	32
3.1.2 Examples	34
3.1.3 Conclusions	40
3.2 General Development Flows	40
3.3 Requirement Engineering	45
3.3.1 Requirements	45
3.3.2 Requirement Elicitation Techniques	47
3.3.3 Feature Models	48
3.4 Component-Based Development	49
3.4.1 Components	49

3.4.2	Model-Driven Software Development	52
3.4.3	Roles in the CBD	53
3.4.4	Examples for CBD Frameworks	54
3.5	Conclusions and Implication for the Thesis	56
4	Security Engineering for WSN	57
4.1	Security Nomenclature	58
4.1.1	Security Attributes	58
4.1.2	Technical Security Terminology	61
4.2	Attacker Models	64
4.2.1	Attacker Classification	64
4.2.2	Attacker Goals	66
4.3	Attack Space on WSN scenarios	68
4.4	Protection Means - Countermeasures	70
4.4.1	Cryptography	70
4.4.2	Applied Countermeasures	72
4.5	Secure In-Network Aggregation	74
4.5.1	In-Network Aggregation	74
4.5.2	Hop-by-Hop Encryption	75
4.5.3	Concealed Data Aggregation	75
4.5.4	State of the Art	76
4.5.5	Security Evaluation	79
4.5.6	Comparison of CDA approaches	80
4.6	Holistic Approaches to configure Security in WSNs	81
4.6.1	Security Service Toolboxes	82
4.6.2	Transparent Security Layers	83
4.6.3	Composable Security Approaches	83
4.7	Security Engineering	85
4.7.1	Formal Security Models	85
4.7.2	Top-Down Security Engineering	86
4.7.3	Bottom-up Security Aggregation	91
4.7.4	Security Metrics	92
4.8	Conclusions and Implication for the Thesis	95
5	A Component-Based Configuration Toolkit	97
5.1	Design Flow	98
5.1.1	Four-step Development Flow	98
5.1.2	The configKIT Development Process	101
5.1.3	The configKIT Data Structures	102
5.2	Property - Relations	104
5.2.1	Definition	104
5.2.2	Formal Specification of the PRG	108
5.2.3	Dynamic Aspects of the PRG	112
5.2.4	Conclusions	115
5.3	Requirement Definition	115
5.3.1	Data Representation	117
5.3.2	Domain-specific requirement elicitation	118

5.3.3	Requirement Translation	119
5.4	Component Model and Structures	120
5.4.1	Component Model	121
5.4.2	Static Component Structure - The Component Repository	123
5.4.3	Dynamic Data Structures - The Component Composition Graph	125
5.4.4	A-priori Reasoning about Composition Properties	128
5.4.5	Property Models	130
5.5	Selection Algorithm	131
5.5.1	Initial Problem	132
5.5.2	Algorithms	134
5.5.3	Complexity and Optimizations	138
5.5.4	Conclusions	140
5.6	Practical Tool Chain and Implementation Details	141
5.6.1	Selection Algorithm - Implementation Details	141
5.6.2	Tool support	145
5.7	Conclusions	147
6	Security Models	149
6.1	Integration of Existing Approaches	150
6.1.1	KMS Guidelines	150
6.1.2	Cionca Security Configuration Approach	152
6.1.3	Conclusions	155
6.2	View-based Model Development	156
6.2.1	Countermeasure-centric View	157
6.2.2	Vulnerability-centric View	158
6.2.3	Attack-centric View	159
6.2.4	Conclusions	160
6.3	Practical Security Models	160
6.3.1	Feature-based Countermeasure-centric Security Model (FCSM)	160
6.3.2	Quality-based Countermeasure-centric Security Model (QCSM)	162
6.3.3	Generic Attack-centric Security Model (GASM)	168
6.3.4	Attack-Tree-based Attack-centric Security Model (ATASM)	171
6.3.5	A generic Vulnerability-centric Security Model (GVSM)	175
6.4	Conclusions	176
7	Secure INA as Case Study	179
7.1	Experiment Setup	179
7.2	Practical Implementation of CDA	181
7.3	Simulation Results	183
7.4	Setting up configKIT for INA	184
7.5	Modeling INA with different Security Models	186
7.5.1	FCSM	187
7.5.2	QCSM	188
7.5.3	GASM	189
7.5.4	ATASM	189
7.5.5	GVSM	191
7.6	Configuring INA with configKIT	192

7.7 Conclusions	195
8 Conclusions and Future Work	197
Bibliography	201
Used Abbreviations	217
List of Figures	219
List of Tables	222

Abstract

The development of secure systems is already a very challenging task. In the domain of wireless sensor networks this challenge is even aggravated by severe constraints of the sensor node devices and the exposed character of the networks.

To cope with this issue, this thesis proposes a tool-supported development flow named configKIT, that helps users to integrate secured applications in the domain of Wireless Sensor Networks. It is a component-based framework that selects and composes configurations of hardware and software components for WSN applications from high-level user requirements, automatically. Therefore, the composition process utilizes a flexible meta-model to describe properties of the components, the requirements, and the system semantics, which allows the assessment of the behavior of the composed system. Based on this modeling technology five practical security models are investigated, which base on different technical views on a general security ontology for WSNs. Each model is discussed theoretically and practically, based on a practical integration in the configKIT framework.

The configuration toolkit and the security models are finally evaluated by applying the techniques developed to the non-trivial example of secure in-network aggregation. The evaluation shows that all five practical security models developed in this thesis work correctly and with reasonable model overhead. These results promote the notion of a practically applicable toolkit to configure secure applications in WSNs.

Kurzfassung

Die Entwicklung sicherer Systeme ist eine sehr anspruchsvolle Aufgabe. Im Bereich der drahtlosen Sensornetze wird diese durch die eingeschränkten Ressourcen noch erschwert. Hier setzt diese Arbeit an, die einen werkzeugunterstützten und modellbasierten Entwurfsprozess vorstellt.

Dieser ermöglicht es, ausgehend von fertigen Komponenten automatisch Anwendungen zu generieren, die den Anforderungen der Anwendung bzw. des Anwendungsentwicklers genügen. Insbesondere unterstützt dieser Ansatz eine ganzheitliche Betrachtung und Bewertung von Sicherheitsaspekten.

Für die Einschätzung der Erfüllung der Anforderungen können dabei flexibel austauschbare Modelle zur Bewertung verschiedenster funktionaler und nicht-funktionaler Aspekte instanziiert werden. Für den Aspekt Sicherheit werden in der Arbeit mehrere konkrete Bewertungsmodelle basierend auf unterschiedlichen Sichten auf ein generelles Sicherheitsmodell für drahtlose Sensornetze hergeleitet sowie deren Modellierungs- und Anwendungseigenschaften diskutiert.

Am praktischen Beispiel der sicheren In-Netzwerk-Aggregation wird abschließend die Qualität und Aussagekraft sowie der notwendige Modellierungsaufwand für jeden dieser Ansätze bewertet. Diese Evaluierung unter Verwendung der in dieser Arbeit implementierten Werkzeuge demonstriert nicht nur die generelle Zweckmäßigkeit des werkzeugunterstützten Ansatzes sondern auch, dass bereits relativ einfach zu implementierende qualitative Sicherheitsmodelle eine gute Repräsentation der Sicherheitsproblematik bieten und somit zukünftig helfen können die Sicherheit in drahtlosen Sensornetzen zu erhöhen.

Chapter 1

Introduction

This chapter introduces the scope and context of the research conducted in this thesis. It further explains the addressed research objectives and achieved contributions of this work. Finally, the structure of the rest of this thesis is presented in the last section of this chapter.

1.1 Research Context

Wireless Sensor Networks (WSN) comprise networks of tiny computing devices, called sensor nodes. Equipped with wireless communication the nodes are the basis of large autonomous networks typically used to monitor physical phenomena or to control actuators in the environment. WSNs have become interesting for a broad range of applications reaching from home, environmental and structural health monitoring to medical, military and homeland security applications. These applications possess a diversity of challenging requirements of functional, qualitative but also security-related aspects such as dependability, resilience and security.

The high requirements are opposed by the scarce resources a sensor node can offer. Resources such as energy, processing power and memory are kept as small as possible with the aim of reducing the costs for the network and to increase the lifetime of the typically battery-powered devices.

Coping with these severe constraints is one significant design challenge in the development of such networks. Therefore, developers have to trade off functionalities, qualities, and needed resources, which in most scenarios are three contradicting goals. This demands a broad range of experience covering knowledge in distributed algorithms, radio propagation, network protocols, electrical engineering, and embedded programming, among others. Also security engineering is an important aspect for many WSN applications, which however is not covered by the experience of most network developers. Despite or due to this heterogeneity of specialties needed for WSN integration, no structured and tool-supported holistic development flows exist. The pre-dominant way of development for WSNs involves handcrafted code and a trial-and-error methodology based on simulations or experiments. This approach is expensive and complicated to apply even for experts, while it is not suitable for end users or experts in the application domain, which are faced with the task of configuring WSNs in practice.

This methodology has another weakness with regard to system security. It is not feasible to integrate security measures based on experiments, since it describes non-functional attributes that cannot be tested. Current answers to the problem include pre-configured security layers promising to enable one sort of security. With respect to the heterogeneity of WSN applications with their individual demands for concealment, integrity and reliability of the network and the individual security-resource trade-offs, this rarely leads to optimal solutions.

While much research effort has been spent on improving the security in specific areas of WSNs, satisfactory approaches connecting these valuable solutions to flexible security toolboxes are still missing.

In this thesis we aim to develop such a toolbox integrated in a WSN development flow which eventually enables even end users and domain experts to configure secure WSNs.

As illustrated as Figure 1.1, this objective puts the scope of the thesis in the center of three intersecting domains of the computer sciences: system engineering, security engineering and WSN engineering. The domain of system engineering concerns process flows and support in integrating the systems, which is needed to research means for the flow. Security engineering clearly is required for the understanding, selection and evaluation security-related properties of the system. And WSN engineering originates the individual requirements and specific solutions which shall be integrated.

All three domains are highly specialized with an abundance of valuable tools and approaches to solve specific problems. However, most approaches are not suitable for application outside their core domain. One problem is that the specific demands of WSNs with their extremely efficient and small, highly specialized, often distributed solutions are not covered by standard software. Also security properties are mostly not respected sufficiently by these tools. On the other hand, practical integration is often not a core interest of security designers.

Since the isolation problem became visible, increasing research has been spent on the interfaces between the domains: project like SHIELDS [SHI10] and Trustsoft [HR06] have researched ways to develop good software and thus treat the interface between system engineering and security. European projects like UbiSec&Sens [Ubi07] and WSA4CIP [WSA10] among others have stimulated the intersected domain of security for WSNs. This emphasizes the importance of the topic covered by this thesis in the center of these three domains.

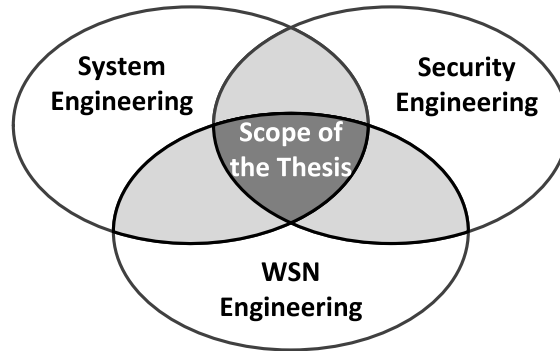


Figure 1.1: Scope of the thesis in the center of the intersecting domains system engineering, security engineering and WSN engineering.

1.2 Research Objectives

The primary objective of this thesis is the investigation of means to support developers and users to integrate security in WSNs. The vision is that a user enters the requirements for the WSN application and its security demands, and based on this input a tool proposes a system that satisfies the requirements.

To reach this primary objective in this thesis a range of scientific and technical objectives will be investigated. These objectives are:

Study the characteristics of Wireless Sensor Networks: In order to develop tools and to evaluate security in WSNs, foremost it is required to understand the properties and techniques of the networks. This includes technology of the devices, their hardware and software, the network and the applications executed with WSNs. It is the goal to identify commonalities and differentiators which can be utilized for an efficient development framework for WSNs.

Understand security in WSNs: The primary security targets are similar to those in traditional computer networks in striving for concealment, integrity and reliability. However, due to the exposed nature of WSNs and the resource constraints such as limited power and memory, threats are different and traditional protection measures are often not applicable.

This requires to study in detail the security requirements, the threats and attacks as well as the range of countermeasures for WSNs. Only with such a thorough understanding is it possible to develop security approaches suitable for a broad spectrum of application scenarios.

Investigate models to assess security in WSNs: In order to apply the results of the security analysis in broad practice, heuristics and models are needed. These models should condense the multi-dimensional security problems to a technically usable comprehensible form.

This abstraction naturally comes at the cost of details. Thus, the identification of security models that on one hand are expressive and consistent, and on the other hand can be embedded in a practical design flows is an important objective.

Investigate frameworks to find suitable WSN configurations: It is the goal that the security models can be embedded in a framework that configures software and hardware of WSNs automatically. Therefore, suitable frameworks need to be studied and further developed. Important aspects of such tools are the support for automatic composition and semantic analysis. While the former needs efficient search algorithms, the latter demands a powerful meta-model which allows to express behavioral models to generate a-priori knowledge about the system under development. The challenge is to find a suitable description model that provides the expressiveness, while not increasing the complexity of the framework significantly.

Provide means that allow users to express application requirements: There exists a semantic gap between requirements as expressed by users and the technical form they are needed to drive an integration process. This is true for technical terms in general and for security in particular. For a tool that eventually should

be applied by end users, a methodology has to be found to bridge that gap. This requires an investigation of requirement elicitation as well as requirement transformation techniques.

1.3 Research Contributions

As answer to the objectives, this thesis presents a set of security models to express issues of security in WSNs. The models originate from a novel methodology that allows to infer technical models based on different views on a holistic security ontology.

The models can be integrated in a novel development framework, named configKIT, that helps users to integrate Wireless Sensor Networks. It is a component-based tool that composes configurations for WSN applications from user requirements, automatically. It instantiates different models to assess semantic properties of the system, such as the security models introduced above.

In the following the research contributions of this thesis are briefly introduced.

Design process for WSNs: A design process, named configKIT, is presented that focuses on the actual integration of WSNs performed by actual users. The process applies the component-based development paradigm, which presumes the components are readily available in the component repository. The selection algorithm chooses a set of components from the repository with the aim the resulting system satisfies the requirements. Compiled at development time the resulting system promises the best possible performance. On this granularity no similar approach has been presented in the domain of WSNs.

Requirement engineering: For the requirement engineering an approach is presented to bridge the semantic gap between requirements given by users and technical requirements expected in the WSN domain. It consists of two steps that are connected by a graph structure of interrelated requirements. In the first step the user sets the requirements on a domain specific level by selecting and parameterizing attributes provided by a catalog. The second step is a translation from the high-level requirements to the detailed technical requirements. The resulting graph is the input for the actual application selection step.

Holistic security model: A security ontology is developed which explains the relations between the system and its security goals to the attacker and its goals. While this ontology does not directly allow to evaluate the quality of a specific system, it helps to explain and understand security in WSNs.

This is the starting point for a novel methodology to develop usable security models from the holistic security ontology. Exploiting the additional abstraction provided by specific views on the ontology it became possible to develop actual technical models which can clearly decide whether the system under development satisfies the requirements within this model.

Practically applicable security models The security model development methodology results in five different security models, covering different aspects of the security system.

The models evaluate security either based on countermeasures, attacks, or vulnerabilities of the system to assess its security strength.

All five models are implemented in the configKIT framework and evaluated based on the use case of the secure in-network aggregation. The models return the correct systems, while the needed efforts for modeling vary. The vulnerability-based model is likely the most advanced model to assess security of WSNs configurations. Also the light-weight quality-based countermeasure model delivers the correct results and with reduced description overhead.

Each presented model embedded in the configKIT design flow promises to improve the configuration of security-related WSN systems significantly.

Property-Relation-Graph to model the semantics of the system: The PRG is a flexible graph structure that can maintain all sorts of the system states, values, descriptions and their interrelations. This allows to generate an expressive graph representing the knowledge of the system. It observe the state of requirements, constraints and potential conflicts in the system and therefore influences the selection algorithm. A configuration is only accepted if it is free of conflicts. Since this well-defined data structure is open and expandable it is the perfect means to express models for specific system aspects.

The PRG structure as semantic knowledge base of the configKIT framework is the main discriminator between other component-based development tools and configKIT.

Selection Algorithm: The selection algorithm facilitates the configuration of WSNs in three ways: it picks specific modules from a component repository, ensures the different modules are compatible – both structurally and semantically – and that they fulfill the given requirements.

While an exhaustive search in theory has a non-polynomial worst case complexity, in practice the algorithm finishes even for systems with more than 50 components in a few milliseconds. This can be achieved by exploiting the inherent properties of typical component dependencies.

Evaluation of the configKIT approach to configure secure applications: The capabilities of the presented design flow and the security models are evaluated in various ways. It is shown that with configKIT it is possible reproduce the applied techniques and the results of the most advanced security configuration approaches for WSNs presented in literature.

Applying the novel security models it is shown that the tool can progress beyond this state-of-the-art. This is demonstrated by configuring WSN applications using the non-trivial example of secure in-network aggregation.

This promotes the notion of a practically applicable toolkit to configure secure applications in WSNs.

1.4 Structure of this Thesis

This thesis is organized in eight chapters, grouped in four parts. The Introduction of the thesis and Chapter 2 concerning wireless sensor networks deliver an understanding of the work and its context. Chapter 3 and 4 provide the state of the art in system engineering for WSNs and security of WSNs, respectively. The core contributions of this thesis are

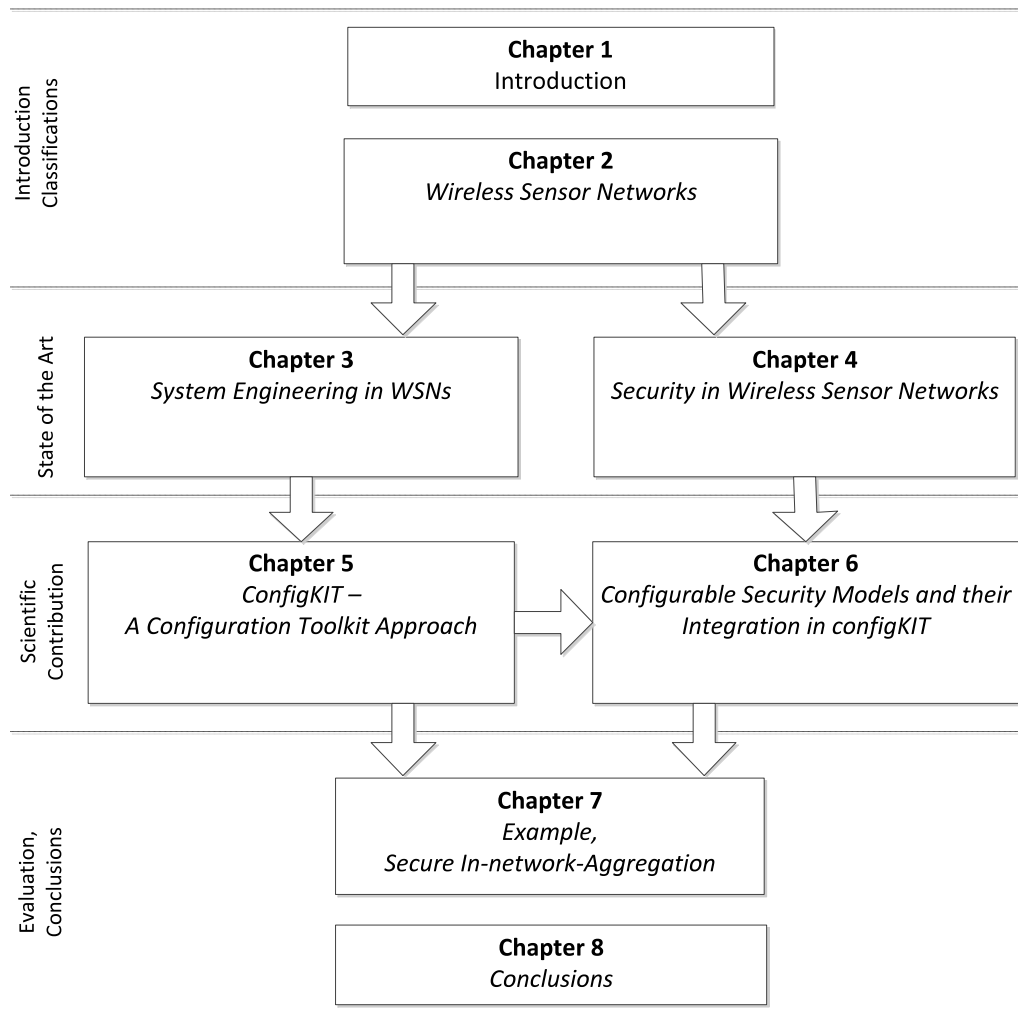


Figure 1.2: Structure of the thesis.

located in Chapter 5, which describes the configuration toolkit, extending the state of the art of system engineering, and Chapter 6, which investigates security models and their integration into the toolkit. In this way, Chapter 6 combines the security state of the art and the toolkit approach. Finally, Chapter 7 evaluates the models and the toolkit using the example of in-network aggregation. Figure 1.2 illustrates the interrelations between the chapters.

Chapter 2 - Wireless Sensor Networks presents an overview of today's characteristics and applications for the wireless sensor network technology. It introduces parameters of typical WSN hardware platforms and the architecture of the software applied on these systems.

Chapter 3 - State of the Art in WSN System Engineering provides a survey of approaches and methodologies that could help the engineering process of WSN systems. This includes programming approaches, tools, and abstraction from the

domain of WSNs which are investigated in the first part of the chapter. Further, processes and development approaches from the domain of software engineering are evaluated. The result is a joint top-down and bottom-up process combining requirements engineering and component-based development in one development flow for which the fundamental aspects of both techniques are studied in detail.

Chapter 4 - Security Engineering for Wireless Sensor Networks

studies security for WSNs and identifies methods to assess and engineer security during system development. Therefore, it defines the term security and presents an ontology describing the participants and objects as well as their relation to the security for WSNs. Further attacker motivations, actual attacks, and general countermeasures are studied in the domain of WSNs. On the protection side, tools to make security manageable are highlighted.

Chapter 5 - A Component-Based Configuration Toolkit

describes a general component-based design flow, named configKIT, for system engineering of WSNs. The knowledge structure of this process is the Property-Relation-Graph (PRG) which is introduced in this chapter. The graph allows to depict and to maintain complex interrelations between the objects. For this data structure a meta-model is presented and a tool set is introduced which is able to solve problems that can be expressed by the meta-model.

This graph structure is applied in the requirement definition phase, suitable also for end users, and the component-based composition process. The composition process further includes a fully automatic selection algorithm that respects the requirements and is able to compose WSN systems. The corresponding structures and the algorithm are described in this chapter. Finally, a tool-set is introduced which supports all steps of the development flow.

Chapter 6 - Security Models concerns the development and integration of security assessment models. For this purpose first the most promising approaches presented in related work are integrated as models in configKIT. The integration demonstrate the flexibility of configKIT, but also reveals some weaknesses of the existing models. To cope with the issue a view-based model development is introduced which infers technical security models starting from the general security ontology. Finally, five concrete models are introduced, each focusing on specific security aspects. The models include a novel qualitative security assessment model and an attack-tree based vulnerability model which are discussed in detail.

Chapter 7 - Secure In-Network-Aggregation as Case Study for configKIT

investigates the feasibility of configuring secure in-network aggregation (INA) by means of configKIT and the proposed security models. Therefore, first, reference data for the six design options of INA are gathered by implementing the approaches in a modular way and simulating the resulting applications. The modular implementations with the well-defined interfaces are basis for an automatic integration process. As first step it requires a general integration of the WSN application and the INA components in the configKIT framework. Then the security models presented in the previous chapter are applied to model the security attributes for the INA components. Configuration test runs provide test results that allow to evaluate the practical applicability of the configKIT approach.

Chapter 2

Wireless Sensor Networks

Wireless Sensor Networks (WSN) are networks of small-sized computer systems which measure environmental phenomena and communicate over wireless radio channels. This chapter introduces the tasks, nature, and challenges of WSNs.

Therefore, this chapter starts with a description of typical application scenarios for WSNs. The aim of this small survey is twofold: first, it provides a general understanding for WSNs and their application space. Second, it allows to classify the application space in few general application classes. This generalization becomes valuable later when we compose WSN applications based on high-level application patterns.

The second part of this chapter describes typical hardware and software configurations of WSN nodes. The hardware description includes typical sensor node platforms, but also delivers a brief introduction of important subcomponents. In particular the severely constrained hardware parameters of WSN nodes pose a challenging environment for sensor network developers.

2.1 Application Space of WSNs

This section describes and evaluates the applications space of WSNs. A set of selected example applications should give an impression and a general understanding of WSN applications. It is not the intention to provide a thorough survey on sensor network applications but to show possible instances that support the development of the taxonomy in the second part of this section. A more detailed survey is for example given in [MP10].

2.1.1 Examples

Environmental monitoring: Certainly the most typical WSN application is monitoring of environments. Monitoring in this context means sensing environmental phenomena and transferring the data to a sort of base station. The base station is able to visualize the measurements directly or distributes the reading over other networks like the Internet to clients that process the data. Figure 2.1 shows the general structure of such a wireless sensor network.

One example for this type of architecture is the monitoring of agriculture environments. In [JRK⁺09] a vineyard was equipped with sensor nodes that measured moisture,

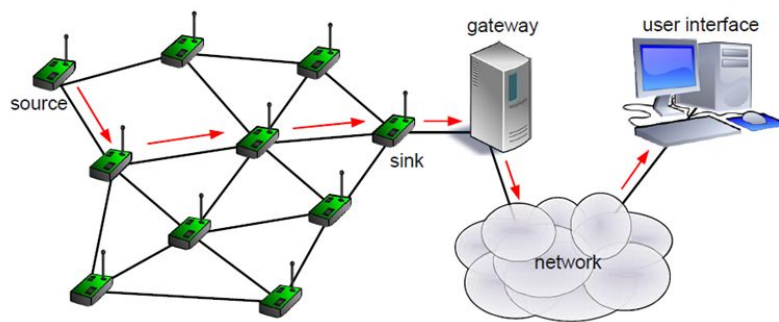


Figure 2.1: Structure of a wireless sensor network [OP10]: Sensor nodes establish a wireless network. They are connected to a gateway which is the access point to the Internet.

temperature and the light level. The data was transferred periodically to the sink while the nodes closer to the base station worked as hops for the nodes farther away.

The prototype application of environmental monitoring with WSNs was deployed 2002 on Great Duck Island [MCP⁺02] to monitor the nesting grounds of the Leach's Storm Petrel, a seabird. There, a WSN with 32 nodes was deployed to continuously measure environmental variables, including temperature, humidity, barometric pressure, and light level, around the nesting places without disturbing the birds.

Habitat Monitoring: While on Great Duck Island actually the environment was monitored, later projects even attach nodes directly to animals to monitor their behavior. A famous example is the ZebraNet project [ZSLM04] that equipped Zebras in Kenya with sensor nodes to perform studies of animal migrations and animal interaction patterns. They recorded the location of the Zebras with global positioning system (GPS) and other collected data "to note whether the animals are eating, moving or resting". Whenever two zebras equipped with the nodes met, the collected data was exchanged. Thus, it was sufficient to find one zebra every few weeks to gather the data of the herd.

Another project tracks the movement of Whooping Cranes during their yearlong migration [ABV⁺10]. The birds are equipped with sensor nodes and the methodology is similar to ZebraNet. After the winter migration that lasts several months it is possible to gather the data the birds collected.

Home Control/Smart Home: WSNs also find applications in the home area, either to collect data in and around the house but also to control electronics in the house. [GWZ⁺05] presented a building automation system based on standard WSN technology implementing wireless thermostats, light switches, and door monitors. The system can be controlled from the Internet via a gateway to the WSN. Notably, the gateway tunnels the WSN data packets in the Internet, allowing end-to-end security to the nodes.

Health Monitoring WSNs are considered as enabling technology for smart home health care. It is the goal that the status of elderly or diseased or injured people can be monitored at home in the same quality it can be in a hospital [VWS⁺06]. Most approaches propose to equip the patients with wearable sensor nodes that collect information such as body

temperature, oxygen level, heart rate, or position in the room. The base station at home collects the data and sends it periodically to a doctor or nurse control station. Currently no working application in this field is known. However, the implications on data security in such applications are interesting for further research.

Homeland Security and Protection: A demonstration closely related to the health care monitoring has been presented by the project Feuerwhere [PSL10], where first responders are equipped with sensor nodes that measure e.g. temperature and oxygen levels. The values are distributed and collected within a body area network while a strong node uses a powerful radio to transmit the gathered data to the operation control.

Flood Detection [BRR08] and detection of fires [YWM05] are other application scenarios in the field of homeland protection, while they are very similar to environmental monitoring.

A different application of WSNs in the domain of homeland security is perimeter surveillance and intrusion detection. A special property of the demonstration presented in [GCN09] is the possibility for special forces to deploy the network easily and dynamically, for example to protect high-risk events. The setup allows to deploy acoustic, visual, vibrational or thermometer sensors and to show suspicious events in a GUI on a computer in the control center.

Industrial Applications Also in industrial environments WSNs are promising means to gather data that are potentially widespread in the industrial facilities. They are in particular interesting since WSNs in industrial applications do not only monitor but can perform actual control.

For instance the RealFlex project [PSL09] addressed that issue by pursuing the goal of flexibilization in automation systems architectures by the integration of reliable wireless sensor nodes that allow real time processing. The project eventually could demonstrate the feasibility by deploying wireless sensor nodes in a system of wells in a waterworks facility. There, Bluetooth equipped sensor nodes could successfully transmit data of pressure and flow rate to a control station for several months [KMP10]. The readings could control actuators (valves) which were also connected to the WSN.

A similar scenario is addressed in a demonstration of the WSN4CIP project [WSA10]. There, a drinking water pipeline on a distance of nearly 20 km shall be monitored by wireless sensor nodes. The sensed values include pressure and flow rates and position of valves but also information to physically protect the pipelines.

Industrial plant monitoring [KAB⁺05] is another application scenario within the industrial domain. Analysis of vibrations, infrared data and ultrasonic, collected by WSNs, can indicate potential problems with the machines in the plant early before causing severe damage. In one scenario on a shipboard 26 sensor nodes and 150 sensors were deployed to monitor the engines and machines on board.

A future prospect of WSNs sees its application in the domain of factory automation. In this field hundreds of sensors may control fast moving robot arms which manufacture goods. A first demonstration has been given also in the Realflex project. Still the extend of the WSNs in this scenario is rather small due to reliability and safety issues caused by the radio channel.

2.1.2 A WSN Taxonomy

Based on the examples presented in the previous section we can identify distinctions between the networks. The result is a small taxonomy similar to other WSN taxonomies presented in literature [MP10]. Naturally, there is some overlap, while there are also differences. Our taxonomy consists of five dimensions:

Communication Pattern:

The communication pattern of a WSN is related to the entities that may communicate to each other. The two common types are:

node-sink many sensor nodes in the WSN send the data to one or many sinks,

node-node sensor nodes are required to send to other sensor nodes.

The typical communication pattern in today's WSNs is node to sink, while for some application node-to-node communication is implemented additionally. None of the applications described in the previous section exists without a sink that either manages the nodes or is the medium to visualize the information of the network. In theory autonomous networks without sinks are considerable for sensor-actor networks. Sense-only networks without sink appear a bit pointless unless the gathered data should be collected manually, reading memory content. Anyway, even more-or-less autonomous sensor-actor networks, like a light-switch to light-relay network need some managing instance.

Activity

The Activity of WSNs determines whether the nodes are used for sensing only or also for control of actuators. Possible characteristics are:

sense-only the network is only a sensing network without major active elements (beside the node's LEDs and the radio),

act-only: the network is only an actuator network without sensors (beside the radio),

sense-and-act: beside sensors also actuators (e.g. switches, regulators, valves) are part of the network.

The dominating network type in the examples is the sense-only network. In home and industrial application the sense-and-react networks emerge, while act-only networks are rather theoretic.

For convenience in this thesis we use the term WSN (wireless sensor networks) as general class for active and passive networks. Active networks are often termed WSN (wireless sensor and actuator networks).

Data Exchange Patterns:

The data exchange pattern determines the kind and trigger of the transmitted information. Typical types in WSNs are:

periodic: WSNs collecting and distributing data in a continuous periodical process,

event triggered: WSNs work event-triggered if they only distribute information when a specific sensor-based condition is met,

queried: Queried WSNs do not pro-actively distribute information but wait until they are queried - typically from a sink or base station. It is often used when the sensor network is considered as a sort of database [GHH⁺02].

WSNs can combine two or three exchange patterns. For example the water pipeline monitoring application sends periodically status updates, but tries to send an event as soon as possible if an alarm situation occurs. Further, the base station can query additional information which are not distributed pro-actively, such as the voltage level of the nodes' battery.

Mobility:

Mobility describes possible movement of nodes and sink. It can be:

static: In static WSNs both nodes and sink are considered not to move after deployment,

mobile nodes: If nodes are attached to animals or objects they may move,

mobile sink: A mobile sink in most cases is a mobile reader which can be moved within the network.

Most oft today's WSN applications are static. If sensors are attached to animals or people the nodes become mobile. Mobile readers typically occur in connection with mobile nodes, such as in the animal monitoring applications. A mobile sink in a smart house, vineyard, or industrial facility is considerable, while not applied in the concrete examples described above.

Structural Network Dynamic:

Structural Dynamic determines the expected changes in the network structure over its lifetime. Typically the structure of WSNs may be:

static: The network is preconfigured and the structure does not change over lifetime.

managed: The network structure and topology can change. New nodes may appear or disappear.

ad-hoc: The network structure changes dynamically without central management. New nodes may appear or disappear and the network has to adapt automatically.

The dominant scheme in today's WSNs is the managed dynamic, while the ad-hoc scheme is a goal for the future with the intention to reduce the management efforts for the networks. Static networks are often used as underlying scheme for WSN deployment. However, most WSNs are inherently dynamic due to possible node or communication failure, which is a reason for many practical implementation problems [BISV08].

As result of this taxonomy we can state that WSNs are:

- small computation nodes with the ability to communicate over a wireless channel,
- equipped with sensors to gather information from the environment,
- optionally connected to actuators which control the environment,
- connected to a sink,
- optionally mobile,
- follow pro-active or passive communication pattern,
- and are organized in a flexible network topology.

As result, the design space of WSNs appears rather small. In fact, if only functional aspects are considered (as in this taxonomy), sensor networks are pretty similar. The differences occur with the non-functional aspects.

For example comparing

- a sensor and actuator network for monitoring and control of a water pipeline and
- a network for monitoring and control of a robot in the fabric automation.

Both scenarios have the same functional requirements and thus find the same spots in the taxonomy. The differences are merely in parameters and qualities. The pipeline monitoring has a period of many seconds to minutes - the robot of a few milliseconds. The pipeline has to cover a distance of kilometers - the robot of few meters. Two differences that change everything, while basically the actual application is still the same. The factors that ultimately differentiate the WSNs are the quality aspects. We are going to exploit this characteristic property of WSNs later in this thesis at the definition of the configuration process by defining only few top applications which can cover the whole space of WSN applications. The difference is the set of services and components that eventually determine the qualities and properties of the application.

2.2 Sensor Node Hardware

This section provides an overview of typical sensor node hardware. In order to outline the hardware design space and the constraints of today's sensor nodes, first, commercial off-the-shelf node systems are presented, followed by application specific sensor nodes and novel platform based WSN systems.

In the second part of this section we zoom in to the actual hardware components used on sensor boards. This description refines the general hardware design space and is the basis for a possible high-level component-based hardware development of sensor node platforms.

2.2.1 Commercial Sensor Node Platforms

Several commercial Commercial-Off-The-Shelf (COTS) sensor node systems have been developed and marketed in recent years. In the following major platforms are discussed briefly. A more extended review of wireless sensor network node technologies is provided in [JHv⁺09].

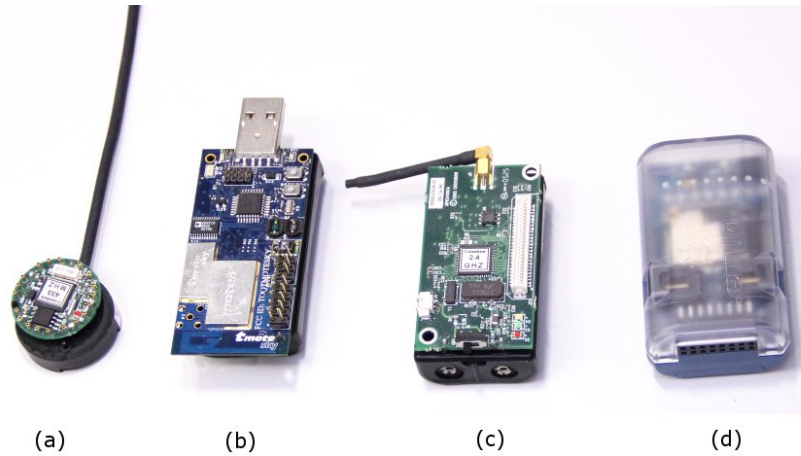


Figure 2.2: Commercial off-the-shelf sensor nodes: (a) mica2Dot, (b) TmoteSky, (c) mica2, (d) Sentilla spots. The size of the three larger nodes is determined by the two AA batteries they need.

Mica Motes

The Mica sensor nodes, developed in the U.C. Berkeley and later commercialized by Crossbow and MEMSIC [MEM11], are among the most popular COTS WSN nodes. They are equipped with a 8-bit Atmega128 microcontroller from Atmel. Different versions are available supporting different radio protocols, ranging from 433 MHz (e.g. the micaDot in Figure 2.2 a)), to 868 Mhz radio or 2.4 GHz (e.g. the micaZ in Figure 2.2 c)). Also different form factors are available, usually dominated by the power supply. The mica2dot has a smaller button cell battery, while the standard configuration of the nodes has two larger AA batteries.

The mica nodes are equipped with a vertical connector that allows to attach dedicated sensor boards to the static base board consisting of computation, communication and power supply.

MSP 430 based systems

Texas Instrument presented the low power 16-bit microcontroller platform which is backbone for most of today's state-of-the-art sensor node platforms. The 16-bit processor and bus is more powerful than the 8-bit Atmel128 and thus allows to apply WSNs in appli-

Table 2.1: Data of Off-the-shelf sensor nodes.

Node	micaz/mica2	Tmote Sky	sun spot
uProc	Atmega128L	MSP430F1611	AT91RM9200
Bus	8 bit	16 bit	32 bit
Clock	8 MHz	8 MHz	180 MHz
RAM	4 kB	10 kB	512 kB
Flash	128 kB	48 kB	4 MB
Flash (ext)	512 kB	1 MB	-

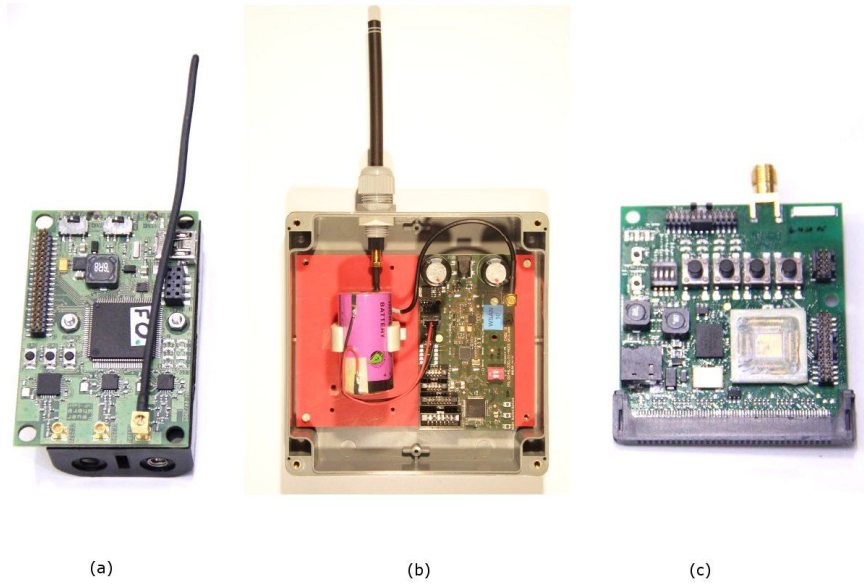


Figure 2.3: Application specific sensor nodes: a) a node applied in a firefighter monitoring project, b) A node as it is applied in the water monitoring application. (Note: the image of the node is scaled down. With its strong radio it is larger than the other two.), c) is a prototype node with a 32-bit application-specific microcontroller.

cations with higher computational requirements. Popular COTS platforms applying the MSP430 are the TelosB [PSC05] from Crossbow and its successor Tmote Sky (see Figure 2.2 b)), or the ScatterWeb nodes developed by FU Berlin [SLR05]. The configurations of memory are similar to the mica family. The supported radios usually support 2.4 GHz (802.15.4) but also 868 MHz radio.

Other Platforms

With the Sun SPOT (Small Programmable Object Technology) Sun presented a JAVA platform for wireless sensor networking. Sentilla [Sen11] has followed a similar trend with their SPOT motes (see Figure 2.2 d)). The Sun SPOTS set on a relatively high-powered 32-bit architecture allowing more computation-intensive operations. Intel presented with the iMote [NKA⁺05] and the imote2 [NHS⁺08] a even more powerful 32-bit platform, equipped with a 2.4 GHz Bluetooth radio.

However, in our perception of WSNs such 32 bit processors are out of scope. They are too complex and expensive.

2.2.2 Application-specific Sensor Nodes

Standard sensor node platforms are beneficial to build test networks. In many applications they are sufficient to work in practice. However, many application scenarios have requirements that cannot be covered by standard nodes. Reasons may be dedicated radio requirements, form factors, specific memory configurations, the need for dedicated circuits to reduce energy or accelerate operations, but also high security requirements that demand tamper resistance or cryptographic hardware accelerators.

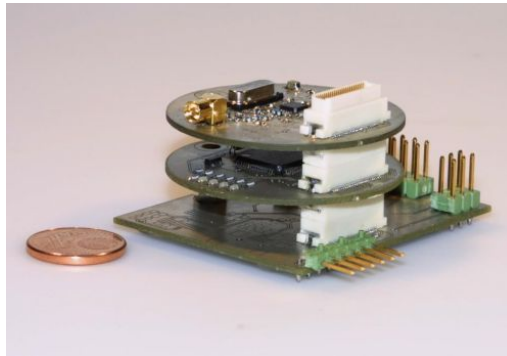


Figure 2.4: Photo of a stacked sensor node platform: small boards for sensors, communication, and data processing are plugged to a sensor node.

Examples for such dedicated nodes developed in IHP [IHP11] are shown in Figure 2.3. Figure 2.3 b) shows a node which is applied in the pipeline monitoring application. In this scenario a small form factor is not as important as robustness, a long lifetime without changing batteries, and the possibility to send over long distances. Therefore the node has a powerful 868MHz radio able to send with up to 500mW sending power and a strong battery. The Lithium Size D battery can be recognized in the opened weather-resistant housing of the node depicted as Figure 2.3 b). The microcontroller is an MSP430 with 16 kB internal RAM and 256 kB internal flash memory. The node has no external non-volatile memory, reducing the risk of extraction of sensitive data from the nodes, which are deployed in the unprotected field.

Figure 2.3 a) is a node applied in the Feuerwhere application. The nodes have a smaller form factor, but the major difference is the availability of redundant radios, allowing to send in the body area network and to the control center concurrently, exploiting the beneficial radio properties of 2.4GHz radio and 868MHz radio, respectively. The nodes also work with the 16-bit MSP430 microcontroller, while the external flash of 4 MB and many connectors for a diverse set of sensors increases the general flexibility of the nodes. The node depicted in Figure 2.3 c), based on the work presented in [PZV⁺08], contains a more powerful 32-bit MIPS processor with embedded accelerators for network protocols and cryptographic operations. While overpowered for traditional sensors this sort of nodes can be used as heads in clusters of the network or operate as sinks. The hardware acceleration of protocols reduces energy by up to three orders of magnitude for specific operations and thus increases the lifetime of the nodes, while still it is possible to process data of many nodes of the cluster or in the network. The disadvantage of such an application-specific integrated circuits (ASIC) is the increased costs due to engineering and manufacturing of the ASICs.

Component-based Sensor Node Platforms

Modularized sensor node platforms emerge as potential answer if individually developed dedicated sensor nodes are too expensive, but off-the-shelf nodes do not provide the required flexibility. A system of stackable function layers (computation, communication, power supply) has been presented in [PDDR06]. The authors equipped the layers with FPGAs (Field Programmable Gate Arrays) allowing hardware acceleration without the

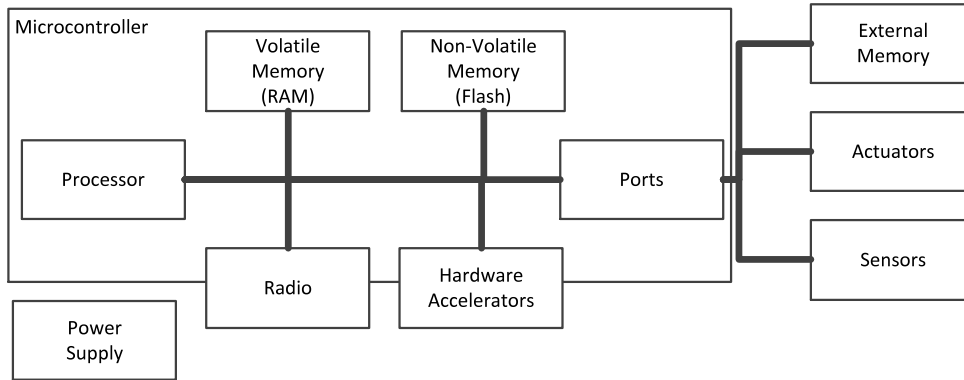


Figure 2.5: Typical hardware architecture of a sensor node: In the microcontroller are integrated processor, internal memory and ports. Over the ports sensors, actuators and external memory are connected. Radio and hardware accelerators may be integrated on the controller or on the board.

need for manufacturing silicon.

Microsoft Research followed a similar approach in their mPlatform [LPZ07]. It includes reconfigurable hardware to adapt interface protocols between different layers of the platform. It extends the possibilities of connecting different service layers. An example for a configurable sensor node platform developed in IHP is shown in Figure 2.4. There communication, computation, sensing and power supply are on individual layers that can be connected to a physical stack.

While in theory the extended design space provided by such modular systems is promising it is also very challenging for system integrators to understand the space of components with their interdependencies, specific benefits and disadvantages. The fact that even FPGAs can be programmed to add hardware accelerators at deployment time additionally motivates the need for tool support that can manage this hardware/software design space. Thus, it is important that configuration tool do not limit their configuration capabilities to software but can also be extended to support hardware configurations.

2.2.3 General System Architecture

Basically, all sensor nodes follow the same general architecture which is depicted as Figure 2.5. Inside of the microcontroller is a microprocessor connected to a system bus that also connects RAM, flash and several ports internally. The ports allow to access sensors, actuators and external memory units.

Memory

The space of amount of memory in WSN nodes is shown in Table 2.1. RAM is usually used as data memory, while Flash memory is typically used for program data. Unlike in PCs the RAM in sensor nodes is not dynamic RAM but static RAM (SRAM). Since SRAM is relatively large in silicon and expensive, it is kept as small as possible. Clearly, the 4kBytes available in the mica nodes is a design challenge for software developers. The amount of non-volatile flash memory for most nodes is more relaxed. It is also common practice to extend it with external flash memory blocks connected to the ports

of the microcontroller. Of course a sensor network designer should regard that external non-volatile memory can be easily accessed from potential data thieves.

Radio

For most of today's sensor nodes the radio transceiver is not part of the microcontroller. It allows combining the microcontroller with the right radio for each application. Standard radio transceiver for current sensor nodes are the sub-GHz TI CC1100 [Tex06] and the TI CC2420 [Tex07] which uses the 2.4 GHz band.

Recently chip manufacturers have presented integrated solutions such as the TI CC430 [Tex11] which combines the MSP430 microcontroller and a sub-GHz radio in one core, or the ATmega128RFA1 [Atm11] which combine an ATmega128 microcontroller and a 2.4 GHz radio.

Hardware Accelerators

Hardware accelerators are applied to execute specific operations faster and usually significantly more energy efficient. The disadvantages of such accelerators are the lack of flexibility and the increased costs to manufacture the circuits in silicon. Typical operations that are worth to be accelerated by dedicated units concern network access and cryptography.

Several design options exist to place the accelerators. For instance [KP04] extended the instruction set of a processor to accelerate cryptographic operations. Extending the instruction set of a processor is a very complex and invasive sort of adaptation. In contrast, [PZV⁺08] presented a solution that added units to accelerate cryptography and network operations in the microcontroller - but left the actual processor untouched. Accelerators connected outside the microcontroller are realized for example in specific radio transceivers such as the TI CC2420 which already contains an AES encryption unit. Indeed, transmitting sensitive data over unprotected buses on the board seems to be ill-designed for many WSN scenarios - but it is common practice. Anyway such design decisions are correct if the application scenario does not permit attacks that can exploit such a weakness.

Power Supply

WSNs are usually battery powered. The most commonly off-the-shelf sensor nodes, such as Mica2, MicaZ and TelosB, are equipped with 2 Alkaline AA cells. The rated capacity of an AA alkaline battery is about 2500 mAh [PLP06]. Smaller nodes like the Mica2Dot are powered by CR2354 lithium coin cell batteries. The rated capacity of such a cell is 560 mAh [Pan10], i.e. about a tenth of the double AA setup. To ensure long lifetime of the nodes, batteries as the Tadiran SL-2880 [TAD11] provide up to 19Ah at 3.6V, which is about ten times the energy of the double AA pack. They are suited well for long lasting networks such as the water pipeline monitoring application. However, the benefits are traded off for a significantly higher price and the larger size of the batteries. It can be seen that the selection of the power supply effects the suitability of the nodes for specific applications.

2.3 Sensor Node Software

While the sensor node hardware has great importance, the actual functionality of the network is described as software running on the nodes.

2.3.1 Operating System

Several dedicated operating systems (OS) for WSNs have been developed during the last few years. TinyOS is the de-facto standard operating system in the WSN research community. Other operating systems for WSN are Contiki, or Reflex just to name a few. In some cases, more powerful wireless sensors use light-weighted Linux-based OS. An extended survey on OSs for WSNs can be found in [RKJK09].

TinyOS

TinyOS [LMP⁺05], is an open source operating system designed specifically for WSNs at the U.C. Berkeley. It is a component based operating system that strictly uses the event-driven design paradigm. The event-driven nature of the OS is proven to be efficient for a large class of WSN applications.

A major advantage of Tiny OS is the minimal code size. Since the operating system is nothing more than a set of components that are bound at compile time to a fixed application image no unused code or operations remain in the software on the nodes.

In order to enable the component-based design TinyOS implements a clear definition of software interfaces and provides a well-defined hardware abstraction architecture. This makes writing platform-independent applications and adding new hardware platforms a simpler task. TinyOS uses a dedicated component-based programming language in nesC [GLVB⁺03]. Since the language implements many concepts valuable in scope of this thesis, nesC is introduced separately in Section 3.1.2.

Reflex

Reflex [WN07] is an operating system for WSNs developed at the BTU Cottbus. It is strictly event flow based, but follows the object orient programming paradigm which is emphasized by the usage of C++ as programming language.

Contrary to TinyOS, Reflex has an explicit kernel layer, containing interrupt handler and a scheduler. On top of the kernel it has libraries which provide the event-flow mechanisms that can be used by the application.

Objects in Reflex are considered as activities that can be configured using the system scheduler. In fact the activities logically replace threads. The scheduler receives interrupts from components or timers and calls or continues the activities. As result Reflex is a single thread operating system it manages to run on a single stack, which reduces the amount of needed system memory.

The program overhead is additionally reduced by compiling kernel, services and application to one system image.

The objects are similar to classic components and allow reusability over standardized interfaces of the object. Currently the number of available components for Reflex is still rather low.

Contiki

Contiki [DGV04] is another open source operating system for WSNs. It has been well recognized by the research community but also by industry. Contiki allows programming WSN applications in plain C. It is multi-tasking ready and still promises to be memory-efficient. Thereby it contains many powerful services such as a full Internet-compatible IP stack, which improves the interoperability with standard networks.

Applications in Contiki are defined as processes which run on top of the kernel. Contrary to most other OSs for WSNs, processes and services can be loaded and changed at runtime.

Similar to the concepts in Reflex processes are called by a scheduler which is triggered by events. Processes are not preempted by the scheduler. As result the OS does not need to care about variables and states of the processes – a concept called protothreads [DSVA06]. However, the disadvantage of the system services and the fixed system kernel is that “Contiki’s event kernel is significantly larger than that of TinyOS” [DGV04].

Discussion

All three OSs presented in this section are suitable OSs for WSNs. We decided to chose TinyOS as default OS and nesC as default implementation language in this thesis, because

- The applied component paradigm fits best our notion of component-based development.
- The acceptance in the research community that provided many valuable components that can be reused. The acceptance has also lead to much research work and thus objects to compare our work.
- The presence of the flexible hardware abstraction layer enables more design decisions that also include different hardware configurations.

Most of the concepts discussed in the thesis are applicable in full extend also to other implementation languages and OSs, as long as they support component-based implementations.

2.3.2 Protocol Stack

Figure 2.6 shows a general abstract software architecture of a sensor node: Below an application component is an optional middleware layer abstracting from the services below. Services can be general services or network services which are emphasized due to the importance for WSNs. The services may use a Hardware Abstraction Layer (HAL) to communicate with the hardware components.

Application: The application is one layer or one component that manages the node. It controls the access on sensors and actuators, coordinates the communication and services. As discussed earlier in this chapter, there are relatively few general types of applications. The differences appear in the usage of the application by the user and in particular by the services the application uses.

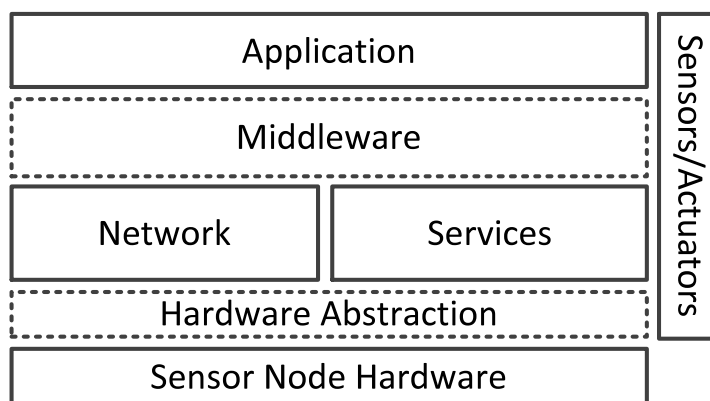


Figure 2.6: Typical software architecture of a sensor node: An application uses an optional middleware layer to use the network and the services which access the hardware over an hardware abstraction layer.

Network services: The network services basically are a stack of radio and network-related protocols that care about sending and receiving data from the nodes using a wireless radio channel. It can be considered as interface between application and the network consisting of all other nodes. Technically the radio stack is a connection between the application and the radio hardware. This service is realized by network protocols which provide abstractions from the physical radio. The protocols include Medium Access Control protocols, routing and forwarding, and transport protocols. The tasks of these protocols are similar to the tasks in traditional networks. In contrast to the Internet where IP is the dominating standard, WSNs do not have a fixed network standard. Thus the network stacks are adaptable and application-specific. For small networks even routing is not needed: then the nodes simply broadcast their messages. The network design space is huge. [AY05] listed alone 22 routing protocols for WSNs.

Services: Beside the network a sensor nodes has many other services, usually covered by the operating system. Typical services include timers, memory storage, system drivers. Also higher level services like cryptography, code management operations, clock synchronization, or random number generators may be integrated. For most services several design alternatives are available, mostly not affecting functional aspects of the application but their quality.

Middleware: Middlewares are means to provide abstraction from technical details to the application designer. The complexity of the middlewares may vary from thin adaptation layers to middleware concepts that include a significant subset of the services. Possible abstractions are node [LC02] or network abstractions [BBD⁺02], database concepts [GHH⁺02], [MFHH05], or storage abstractions [PLP09], [GWMA07]. They are considered as key to improve programmability and usability of WSNs. This is why Section 3.1 is dedicated to provide a more detailed discussion of middleware abstractions to improve programmability of WSNs.

2.4 Conclusions and Implication for the Thesis

This chapter introduced the technological background and the application space of Wireless Sensor Networks. A brief study of application scenarios revealed the great potential of the networks of tiny computing devices, called sensor nodes, which communicate wirelessly and typically monitor physical phenomena or control actuators in the environment. The application spectrum ranges from environmental monitoring, industrial application to the protection of infrastructures. One result of this survey is that on functional view WSNs in the different application spaces are pretty similar. What make the difference are the performance requirements and qualitative needs.

However, achieving the required performance and quality for the networks is the biggest challenge. Cost constraints and limited amount of energy in the usually battery-powered sensor nodes are the major reasons for the lack of computation power and the severely constraint amount of memory.

That is why in the second part of this chapter typical software and hardware architectures and components were studied. In particular for the sensor node hardware exists a vast diversity of platforms and configurations. They range from commercial off-the-shelf sensor nodes – which are readily available, have reasonable prices, but often do not provide the required performance – to application-specific nodes, which are tailored for the application to improve the performance, which also increases the costs. Despite their different characteristics, all nodes share a similar architecture. It is introduced in this chapter.

On top of the node hardware runs software consisting of operating system, networks stack, services, middleware, and the application. It is a slim architecture with the goal of utilizing the available resources as efficient as possible. The most important software decision is the choice for the operating system. After a study of state-of-the-art operating systems for WSNs we decided for TinyOS as default platform in this thesis. Reasons are the applied component methodology and the presence of flexible hardware abstraction.

Components and services on top of the operating system vary with the application and with the required performances and qualities. Implementing and integrating this software is concerned with trading off performance and overheads in resource consumption.

Realizing the demanding quality requirements from the applications on the severely constraint sensor nodes with the straight software architecture is the challenge for WSNs. The configuration tool intended in the thesis is supposed to tackle that issue.

Chapter 3

State of the Art in WSN System Engineering

It is our vision that in the future the design of WSN application is not the expensive and extremely time consuming development task it is today, but a rather straightforward process. Ultimately, it should even be possible for end-users to execute this process on their own.

This chapter provides a survey of approaches and methodologies that could help the engineering process of WSN systems. The fundamental motivation of this survey is the identification of methods supporting an automated integration process. Most approaches originate from the domain of software engineering. However, since also hardware and system architecture are part of the design process we stress the term *system engineering*.

In the first part of this chapter an overview on the existing programming methods for WSNs is given. Existing methodologies to engineer sensor network applications are evaluated in context of the related work.

Then general design flows and development processes are discussed. In this discussion we can condense the space to two methodologies: A top-down development flow that start with the requirements and which will be refined to a technical level. And second, the bottom-up approaches that start with available components and combine them to accomplish complex tasks. The driving techniques of both approaches, i.e. requirements engineering and component-based development, are studied in the second half of this chapter in order to obtain a thorough understanding of the core concepts needed for the composition toolkit introduced in Chapter 5.

3.1 WSN Programming Methodologies

In this Section the space of existing methods and tools to develop and integrate WSNs is discussed. Therefore, we present a general taxonomy that distinguishes the approaches on their abstraction level and the provided programming support, and further discuss the variety of available tool support. In the second part of this section examples from the related work are briefly introduced. They practically characterize today's concepts to develop WSNs.

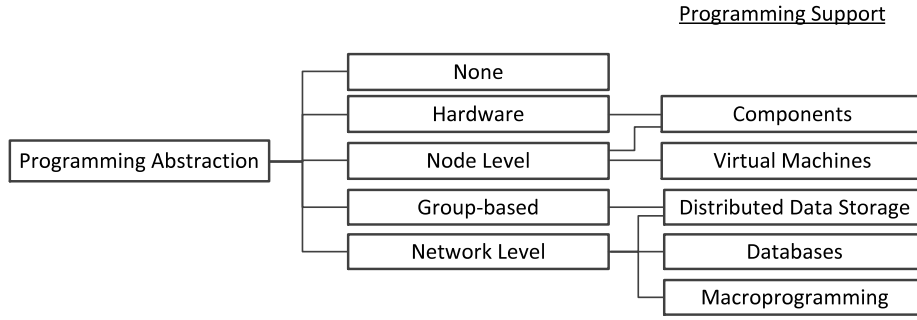


Figure 3.1: Taxonomy of WSN programming methods.

3.1.1 General Methodologies

Development and programming WSNs has been an active research for several years. As result many methodologies and approaches have been proposed to tackle the issue. Also first surveys and taxonomies such as [MP10] were published summarizing the topic. Generally, the development support can be distinguished based on the abstraction level, the programming support, and the tool support. Figure 3.1 shows a small taxonomy of the programming abstractions and the corresponding programming support. In the following section the methods are introduced.

Abstraction Levels

No abstraction: Programming sensor nodes without abstraction typically means directly accessing the hardware and memory. If executed correctly the approach promises the smallest code size and the best performance. However, integration is extremely time-consuming and error-prone and thus is not preferable.

Hardware abstraction: With hardware abstraction the nodes' hardware is not accessed directly but via a hardware abstraction layer (HAL). Such a HAL enables reusability of code for several platforms and improves the productivity of integration. Still the implementation work is difficult and time consuming. An example is nesC, the implementation language of TinyOS.

Node level abstraction: Larger middlewares or virtual machines provide an abstraction of an entire node. The middleware offers a set of powerful services which are resolved to actual hardware calls. These middlewares are programmed in an individual programming language. The concept is realized in virtual machines as MATe [LC02] and Impala [LM03].

System level abstraction: In the network abstraction the whole network with all its nodes is considered as resources of one centralized program describing the global behavior. The programmer only needs to describe the requirements on network level and a powerful middleware cares about the realization. The concept was realized in macroprogramming environments like Kairos [GKGM05] and Macrolab [HSH⁺08].

Group-based abstraction: If not the entire network should be programmed but a group, such as neighborhood or a set of specific nodes, the group-based abstraction finds application.

Programming support

To implement the abstractions several general programming concepts are available. In the domain of WSNs the following are most notable:

Component-based: Components are a well-known abstraction from implementation details. Usually component-based systems provide an abstraction from hardware so that components can be assembled regardless of the applied hardware platform.

Virtual Machines: Virtual Machines (VM) provide a powerful abstraction layer that can be programmed independent of the underlying system. The underlying system in most cases is one computer system, i.e. sensor node, while approaches exist that consider the entire network as one VM. It implies that the code has to be written only once and it will work for all platforms for which the virtual machine has been implemented. Programming is supposed to be easier due to the existing well-implemented functions in the VM. A great advantage of VMs is the dynamic reprogrammability of the nodes during run time.

The most severe disadvantage of VMs for WSNs is the processing overhead due to double instruction decoding. The VM code has to be translated in machine code which finally executes the operation on the node. Also the total performance of the programs cannot compete with optimized handcrafted code.

Databases: A popular approach to provide abstraction is the representation of the whole network as database. The program in such approaches is a query that is processed in the network. The programmer does not care about the execution of the query.

Data Storage: An alternative to the database view is the notion for the network as distributed data storage. The network is considered as sort of file system and the data can be accessed without the user knowing where the data actually is located.

Macroprogramming: Macroprogramming is a programming approach that let a programmer describe the desired behavior of the network instead of the functions of a single node. The macroprogramming framework decides how the operations are implemented in the network and on the nodes. By this, macroprogramming environment can be considered as virtual machine of the network.

Tool Support

Many tools have been developed and range from pre-compilers to simulation frameworks. In this section we focus on tools that actively support the development process of WSNs. Figure 3.2 shows a brief taxonomy of the most important tool categories to support WSN programming.

Offline Optimization: If referred to component-based optimization, the offline optimization delivers a composition of components based on the application requirements. This includes exactly the components (i.e., hardware and software modules)

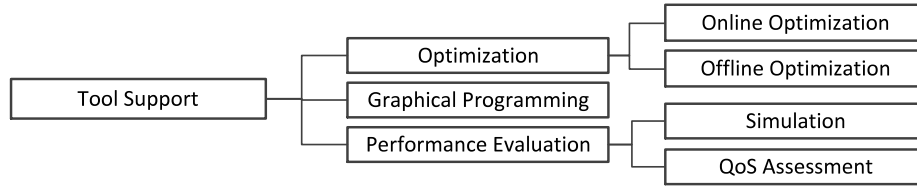


Figure 3.2: Taxonomy of WSN programming tools.

that promise to deliver the required task optimally. Hence, it promises to be highly efficient with regard to memory and computational costs.

With regard to Middlewares and Virtual Machines it aims at tailored middlewares reducing the supported operations. Hard coded specialized middlewares perform faster than flexible one-fits-all types.

Online Optimization: If it is required that the middleware reacts dynamically on run-time situation, online optimization is an option. One possible instance of online optimization is realized by code migration as realized in Agilla [FRL09]. The standard approach is to have all code and information on all nodes and just activate a specific one.

Graphical Programming Support: Graphical programming environments are an approach to open the WSN programming to end users and domain experts but also to ease the work of developers. Approaches exist to support development of distributed algorithms, such as Castalia [Bou07]. Other tools visualize component-based compositions such as Viptos [CLZ05] or GRATIS [VMD⁺05a], graphically support the process of macroprogramming (Srijan [PZP08]), or foster usability for non-experts (eBlocks [CMVH05]).

Performance Evaluation: The assessment of non-functional properties as early in the development flow as possible is important to improve development processes. Simulations are one means, while they typically rely on implemented programs. Preferable, the system properties can be estimated before implementing the system.

Simulation: Simulation tools are needed to execute extensive tests of the network before actually deploying or programming the nodes. The tools deliver information concerning communication patterns and ideally about the internal state of the nodes. In the TinyOS environment the discrete event simulator TOSSIM [LLWC03] is integrated. Avrora [TLP05] is able to emulate networks of nodes with AVR microcontrollers. A graphical simulator for distributed algorithms implemented in WSNs is Castalia [Bou07].

3.1.2 Examples

In this section specific instances of WSN programming support approaches are presented in more detail. The objective of this discussion is the identification of practical methodologies to implement WSN networks. The examples are grouped with respect to their dominating programming support.

Component-based

nesC [GLVB⁺03], the programming language of TinyOS is a component-based programming language that adds a component abstraction to the C programming language. The components rely on the hardware abstraction provided by TinyOS, which typically makes them hardware independent.

NesC uses abstract components, called modules, which use and provide interfaces. Interfaces are sets of bidirectional functions (commands and events). The provider of the interface implements the commands, which are called by the user of the interface. Events are the opposite. Modules define the interfaces they use or provide. The interface functions of a module are implemented in a specific implementation block. In these implementations standard C code can be embedded. The basic concept however is that all program code is triggered either by a command or an event and the execution is non-preemptive.

Modules are composed by configurations. These files define how the modules are connected by which interfaces. Since eventually the application is defined in the same way only components are included that are really used.

While the basic concepts in nesC and the resulting application are impressive, the required implementation techniques are tedious. Small changes may need significant amount of modifications in interface, module, implementation and configuration files. This is why many additional higher abstractions on top of nesC have been proposed.

SNACK [GKE04] – the Sensor Network Application Construction Kit– is a higher abstraction on top of nesC. It is a component composition language that provides general node-level abstraction. SNACK comes with a nesC-based component library that allows reusability. The libraries provide a high-level abstraction of concepts like network protocols, and periodic behavior. In the programming language the programmer uses components on the node and expresses their semantics as parametrized wiring.

For example

```
SenseLight(period<=10) -> [collect] RoutingTree
```

collects the readings of a light sensor every 10 seconds in a routing tree network.

The compiler analyzes the abstract code and resolves actual implementable components. In the example, RoutingTree is decomposed to a Network component and a Treebuilder and Dispatcher component among others. These components can be further decomposed to actual nesC components. Finally nesC code is generated that instantiates the resolved components. The final compilation and linking is performed with the TinyOS tool chain. During decomposition and component assignment several design options are possible. In this process SNACK is looking for reusable components that can be combined to reduce code size and improve performance. The library used by SNACK is rather small and apparently does not allow to be extended by other standard components. At least articles published up to now do not indicate how for example a new network protocol could be incorporated.

FABRIC [Pfi07] is a data-centric middleware synthesis tool for WSNs. A user defines data types and annotations. Based on this requirement the middleware compiler composes a service stack that provides a well-defined interface to the users' application.

The structure of the service layers is defined by a framework specification. While in general the framework is flexible, for the selection process it is invariant.

The requirement, i.e. the data type and annotations, is mapped to a static service stack (called domains). Each service layer contains several selectable modules with individual properties (aspects) and qualities (priorities). It is the goal to find a composition of modules that satisfies the requirements of the user. The idea is that in a first step all modules are filtered which do not satisfy the annotations of the user. As second step the interfaces of the remaining modules are checked to ensure that modules can be connected. The result is a composition of components that provide the middleware usable for the application.

As part of the aspects, FABRIC also respects security and reliability as part of the middleware. We will discuss the features in detail in Section 4.6.3.

Databases and Data Storage

tinyDB [MFHH05] provides a SQL-like query support similar to classic databases. For example the query

```
SELECT nodeid, light, temp
FROM sensors
SAMPLE PERIOD 1s FOR 10s
```

specifies that each sensor should report its own id, light, and temperature readings once per second for 10 seconds. The results of the query are sent to the root of the network where it is provided to the user.

Also aggregating and filtering queries are supported, such as

```
SELECT AVG(volume),room FROM sensors
WHERE floor = 6
GROUP BY room
HAVING AVG(volume) > threshold
SAMPLE PERIOD 30s
```

which all 30 seconds returns the rooms and the average of the measured volumes for all rooms with an average volume above a specified threshold. For data collection application with otherwise passive nodes this database approach is valuable. However, explicit node-to-node communication or user-defined functions on the nodes are not supported.

TinyDSM [PLP09] implements the distributed shared memory concept for WSNs. It allows nodes to access and share variables. This sharing is used as caching and replication and thus improves performance and reliability of the network. On top of TinyDSM the query language DSMQL realizes a relation-based database model. It supports three general operations:

get collects a number of data from a set of nodes. The syntax resembles SQL queries:

```
GET <List of Variables>
FOR [NodeID]
```

set allows to set a value on a node. This operation can be controlled by system policies.

update notifies other nodes in the network that a specific value has changed. It enables event notifications and pro-active data dissemination.

The basic operations can be extended with predicates for example the query

```
GET Temp, Hum, ID
FOR ID > 5, TS > 20
Where Temp > 5 AND Light < 100 OR HUM >= 10
```

collects temperature and humidity for nodes with specific ids, freshness and data values. In contrast to TinyDB the queries can be executed from any node in the network. Another difference is the possibility to perform active monitoring and event-detection and notification – not only to a base station but also on node-to-node basis. This is possible since the DSMQL abstraction is only an access layer on top of TinyDSM and not the underlying concept of the network. The actual TinyDSM layer is a more powerful flexible policy-controlled system with optional encryption and adaptable communication infrastructure that works independently of the executed queries. This flexibility can be exploited for component-based adaptation of TinyDSM. This means components for encryption, communication, and other services can be exchanged to influence the quality and the performance of TinyDSM.

Virtual Machines and APIs

MATE [LC02] is a byte code interpreter on top of TinyOS. It provides a set of primitives for programming, and has an own scheduler to organize executed operation flows. By this it provides the functionality of a traditional operating system. The scheduler fetches the program byte code from the memory and forwards the operations to responsible components in the virtual machine. If the instructions the application actually uses are known, the set of supported instruction of the VM can be reduced. A similar concept has been proposed by application-specific VMs (ASVM) [LGC05].

Since the application runs on the VM layer and not directly on the hardware, the VM delivers a protection of the system from crashing applications. The additional abstraction layer on top of TinyOS and the independent byte code allow software update of the application.

Similar concepts were realized in Impala [LM03] the virtual machine developed in the ZebraNet project.

MiLAN [HMCP04] is an energy-aware adaptive middleware for WSNs. Via a standard API, implemented in the C programming language, applications specify the sensing requirements and the needed qualities. A central control system monitors the network and the requirements by the single nodes and it adjusts the configuration of middleware and the network. By these means it allows a vertical optimization and management also of low-level mechanisms, for example of network parameters. The composition is realized by plug-ins and service discovery protocols. MiLAN respects reliability as function of confidence in the received values, and can trade the reliability of measured phenomena with the efforts to gather them. For example to reduce the number of false positive alarms in a fire detection sensor system, additionally to temperature a hydrogen sensor can be accessed. They also can be considered as virtual sensor with higher reliability, while it also needs additional overhead to gather and evaluate the readings.

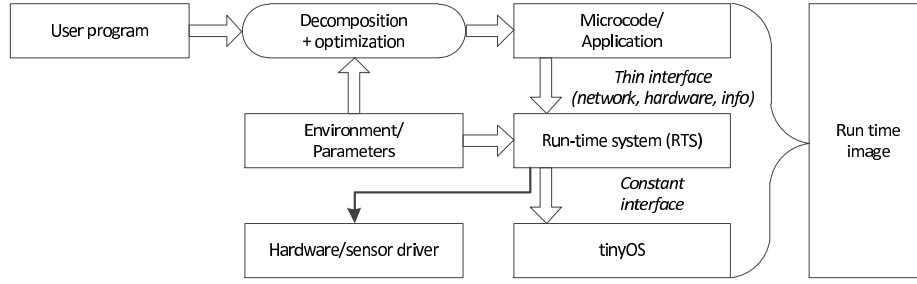


Figure 3.3: Macrolab system model: A user program is decomposed and optimized using environmental parameters for microcode which runs on a run-time system similar to a virtual machine. The microcode, the RTS and the operating system are compiled to one run time image, promising high performance.

However, MiLAN is rather a manager of available resources in the network than a support for integrating new WSN applications and protocols.

MagnetOS [BBD⁺02] is an operating system that provides a Java VM on top of a distributed sensor network. Additionally to the classic operations of a VM it contains an automatic object migration in the network. Applications are automatically and transparently partitioned into components that are dynamically placed on nodes within the network. By this, the abstraction represents the whole network as one Java VM.

Macroprogramming

Kairos [GKGM05] is a imperative procedural programming language that allows to program sensor networks as a whole. In Kairos sensor nodes are data types, organized in a list that can be accessed like standard types in the Python programming language. The programmer is supported by functions like `get_neighbors(n)` which provides a list of neighbors of the node `n`, and remote data access that allows to access variables at named nodes (`value@node`). The following code snippet shows the application of the concepts.

```

int aggregation=0; node it, temp;
for (temp=get_first(full_node_set); temp!=NULL; temp=get_next(full_node_set))
    neighboring_nodes=create_node_list(get_neighbors(temp));
for (it=get_first(neighboring_nodes); it!=NULL; it=get_next(neighboring_nodes))
    aggregation=aggregation+events@it;

```

The example aggregates the number of events in the neighborhood of each node. The variable `aggregation` then can be accessed on any node. Kairos relies on a static runtime layer deployed on all nodes managing the objects (variables) and the network communication. The example shows that despite the network abstraction, Kairos still allows (and requires) directly programmable interaction between nodes.

Macrolab [HSH⁺08] is one of the most-advanced macroprogramming implementations available for sensor networks. It introduced the concept of deployment-specific code decomposition. The user writes relatively simple, deployment-independent programs. The programs are decomposed and finally converted to microcode that can be executed

on the nodes. The development process is illustrated in Figure 3.3. The decomposition and optimization uses properties of the deployment side. In large multi-hop networks the decomposer would select distributed algorithms to reduce the network overhead, while in smaller networks centralistic solutions are preferred to reduce efforts for consistency and synchronization. Cost functions such as power, bandwidth, messages, or latency analyze potential solutions and select a preferable one. The result will be an optimized microcode application.

The microcode is executed on a run-time system (RTS) that maps the microcode on the actual operating system routines. The RTS is very thin. It just provides few functions for networking and synchronization, for status information and direct hardware access. The latter is required for user-specific sensor access. The interface between TinyOS and RTS is static. In fact the RTS is just a static module inside TinyOS. All used modules of TinyOS and their parameterization are fixed in the code of the RTS.

Microcode, RTS and TinyOS are compiled to one run-time system. The microcode is fixed for the image so that it does not need to be interpreted during run-time. That -together with the thin RTS- results in a very efficient solution.

Graphical Programming and Simulation

Viptos [CLZ05] is a graphical interface for development and simulation of TinyOS programs. It allows developers to convert nesC programs to block and arrow diagrams to visualize component relations and data flows. In the GUI Viptos is a program editing and a discrete and event-based simulation environment. Finally, the tool transforms the diagrams back to nesC configurations that can be compiled with the TinyOS tool chain.

GRATIS [VMD⁺05b], the Graphical Development Environment for TinyOS, contains a graphical tool to support component-based development in WSNs (see also Section 3.4.4). GRATIS is also able to represent nesC programs as graphical component graphs. It shows the components and the wiring between the components. The graphical representation, and the embedded development and interface checking tools of GRATIS, help developers to implement WSN applications. Similar to Viptos, GRATIS does not enable the access for end-users since its concept aims towards experienced programmers for whom the tools certainly are a great support.

Srijan [PZP08], developed for sunSPOTs, is a graphical toolkit that uses a data-driven macroprogramming backend. Based on an application task graph and some imperative code for specific functions, it automatically generates system structure and program classes that can be customized by the developer. That way the graphically-controlled macroprogramming framework organizes data flow on the node and in the network while the developer just has to fill in the function blocks.

eBlocks [CMVH05] is a graphical tool that promises to allow creation of WSN systems even for non-technicians. Similar to the world of industrial automation programs can be assembled by a dataflow-based combination of default blocks. The idea of eBlock is to describe rather high-level blocks that can be assembled in a GUI. All eBlocks should be self-explanatory which needs no training or tutorials. This is also by the extensive graphical development environment that illustrates each block with individual pictures,

such as a lamp for a LED. eBlocks comes with a code generator and a simulator that allows testing the behavior of the system. Unfortunately eBlocks is only able to express functional behavior. QoS, performance or security are not supported.

3.1.3 Conclusions

The presented fraction of existing WSN programming tool and abstraction approaches already indicates the immense diversity of such support tools. While many concepts look very promising, it has to be noted that to the best of our knowledge none of the approaches is ready for immediate usage. Usually the tools are results of research projects and tested only on small prototypes.

Beside that, notably, most presented approaches focus on support for programming rather than on integration and service composition. This is motivated by the wish of today's WSN developers to have better tools to develop and integrate new algorithms. However, it is unlikely that development of algorithms is a core task in the integration of practical WSNs. As conclusion of this methodology most abstraction approaches provide a fixed middleware and optimize the program running on top of that middleware instead of optimizing the services in the middleware. The network stack inside the virtual machines is fixed to what the middleware designer thought would be useful for a broad range of applications. Most advanced in this respect are TinyDSM, which allows to use alternative network stacks, and Macrolab, which at least allows parametrization of the radio protocols based on environmental properties.

It is further notable that quality of service –if supported at all– is limited to parametrization and composition to improve the lifetime of the network. In particular security is only respected by few programming approaches:

- FABRIC which supports security as aspects of the composable components
- Virtual machines like Mate promise improved robustness of the system against crashing application code.
- MiLAN introduces a reliability metric to combine sensor readings.
- TinyDSM can control the robustness of the network with its replication strategy. It also allows internal encryption of the distributed data storage based on a security policy.

Beside that, the support of user and developers with regard to security-related properties in WSNs is ignored. In Section 4.6 existing approaches to configure security in WSN are discussed in more detail.

In the following sections we will extend our search for suitable system engineering approaches outside the core WSNs domain.

3.2 General Development Flows

In this section existing general designs flows from the domain of software engineering are discussed. It is the goal to identify approaches that benefit the WSN development flow. The result of this evaluation is that, in order to foster reusability, a combination of top-down and bottom-up engineering is needed.

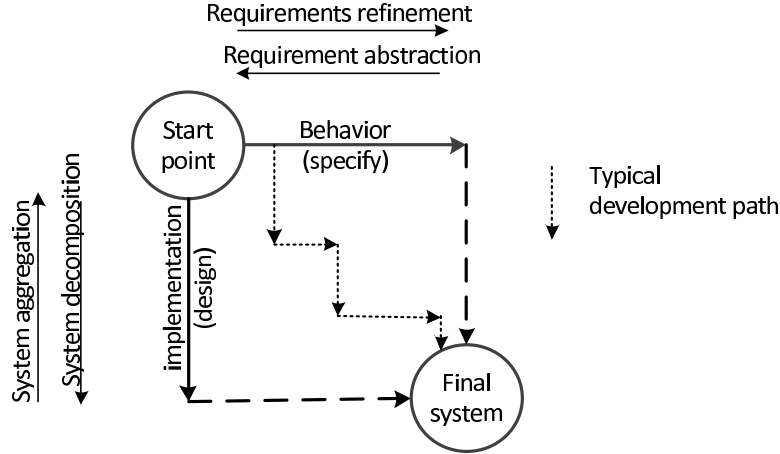


Figure 3.4: “Magic Square” of system engineering [HP85], [Wie99]

[HP85] described the software development process as sequence of transformations

$$(M^0, S^0) \rightarrow (M^1, S^1) \rightarrow \dots \rightarrow (M^f, S^f)$$

while the pair (M^i, S^i) defines a specified system at the i 'th level of detail. M^i is the i 'th level of implementational description, and S^i is the i 'th level of behavioral description. M^0 of such a description is the highly unspecified top system with its vague behavioral requirements defined in S^0 . Each level may refine the implementation detail (M) or the behavioral description (S), while the consistency of the new behavioral description S^{i+1} corresponds the requirements of S^i . Refinement of the implementation is called *design*. Refinement of the behavior is *specification*. Finally the behavior S^f of the implementable system M^f corresponds to the requirements in (M^0, S^0) .

As example in the WSN scenario, a user selects a basic type of WSN application (that is M^0) and defines the quality requirements and constraints (S^0) in a more-or-less precise fashion. It is then task of the WSN designer –or later task of the configuration tool– to perform a sequence of transformations, i.e. design, specification, and implementation steps, to reach (M^f, S^f) , that is the implementable system.

Notable is that design and specification are orthogonal. It was also illustrated in [HP85] by the Magic Square depicted as Figure 3.4. As also pointed out in [Wie99] the top left point in square represents the mission of the system at the highest level of abstraction. The upper right corner represents the system requirements at the highest level of refinement. The lower left corner represents a state in the implementation that describes structure and purpose of each low-level component. This means, in order to achieve the final system a developer can refine the specification first and then start the design process. Oppositely a system could be designed first and than validated against the specifications. As we will see later the both extremes can be recognized as the top-down and bottom up implementation strategy, respectively.

The operations along the two sides concern *requirements* and the *system*:

Requirements refinement: Initial requirements given by the user are usually not by technical nature. Often such requirements are not even clear and unambiguous. Requirement refinement is the process of transforming the initial requirements to

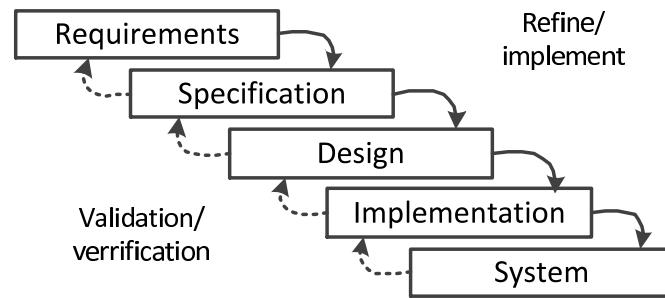


Figure 3.5: Waterfall model: Starting with the Requirements, in sequence, a system is specified, designed and finally implemented. The upward direction serves the purpose of validation and verification of the development process.

technical clear and sound descriptions of the properties of the system. This process is also called requirement engineering.

Requirement abstraction: It is not an actual process step during the development phase, but rather connected to the organizational structure of the development process. In order to express requirements, abstract models are needed. A formal requirement abstraction model is also key for any verification process. In this thesis requirement abstraction will be addressed in the development of security models.

System decomposition: During the system decomposition a system under development is broken down to smaller sub systems or components that in sum the expected global behavior is not changed. The final system consists of many atomic sub systems that can be easily understood and integrated. System decomposition is also referred as top-down development approach.

System aggregation: The process of combining sub systems to a larger system is called system aggregation, or system composition. The step of system aggregation is needed if sub systems should be re-used. This process is often called as bottom-up development.

All four operations are applied to variable extend in typical development strategies.

Top-down development

Top-down development flows in system development apply a straight flow starting with the refinement (specification) of the initial requirements to a complete specification which subsequently will be implemented. Three models implementing the top-down process are discussed in the following.

Waterfall model: A practical implementation of the straightforward sequential development process is the waterfall model. A representation is shown in Figure 3.5. It assumes to finish the specification phase before starting with the design that leads to the implementation.

In the magic square the waterfall model first goes from the start point to the top right corner, finalizing the refinement of the requirement specification. When this is done, the

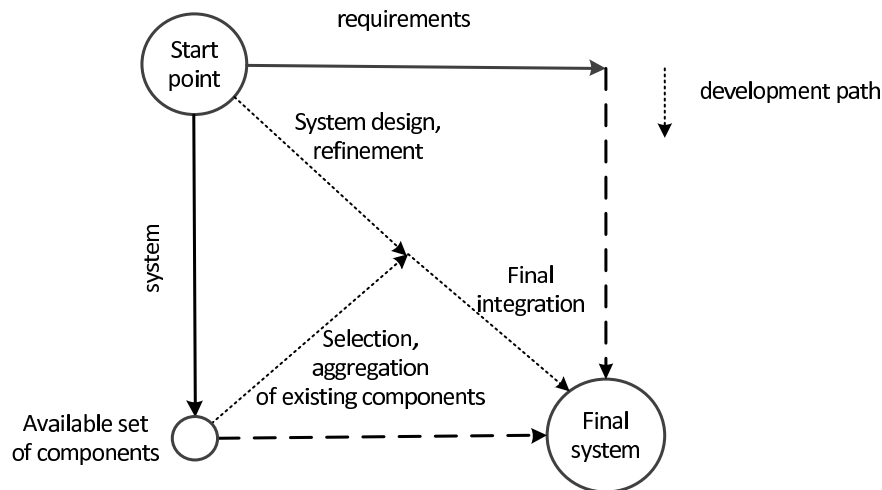


Figure 3.6: Adapted “magic square” with component reusability. In the bottom left corner starts a process step with a readily available set of components. Selection and aggregation of these components within the context of the system design accelerates the system integration.

design and implementation phase can start, which corresponds to a vertical arrow down to the final system.

The model has some severe weaknesses. It does not directly allow re-usability of components or systems. Another serious problem is the inflexibility of the big-step methodology. It is at least questionable to assume that all requirements can be refined to the uttermost extend before starting the software design phase.

Iterative Development Models: To reduce the step width models emerged which interleave design and specification phases. The result is what is marked by the dotted lines in Figure 3.4 as typical integration strategy. It is closer to a diagonal line in the square. That means that specification and design progress with similar level during the development process.

The *evolutionary development model* is one example for such iterative design. It is the goal always to have an implementation that respects the current state of system understanding. The *spiral model* [Boe88] is another example for such an implementation strategy. Analog to a spiral it starts from the inside and does the same steps of the development circle repeatedly with increasing level of detail. So it successively refines specification and implementation to narrow to the final system. The motivation of such iterative models is to reduce the risk of a project by seeing actual implemented progress.

Development models that foster reusability

Sequential processes are well-suited for developments starting from scratch. They however do not directly respect re-usability of existing components. Indeed, the top-down processes refine the system architecture until the component level is reached – and if there are reusable components that correspond to the required specification they can be used. However, while they allow it, sequential design processes do not foster the concept of reusability.

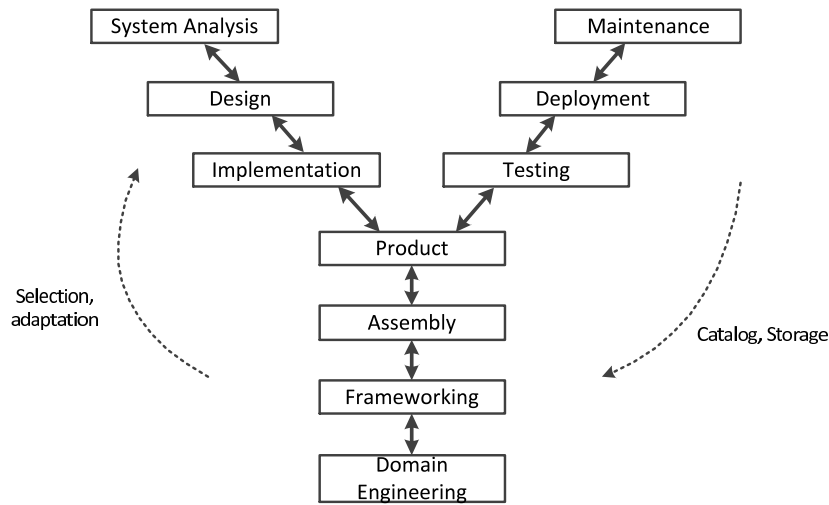


Figure 3.7: Y-process model for component based development [Cap05].

Reusability in fact needs an adaptation of the development path in the magic square. The development is not a straight process from the top left (Start) corner to the bottom right (Final) corner. Instead the system development starts from two positions: the actual requirement-defined start point, and the available set of implemented components. The adaptation of the square is depicted in 3.6. The integration process starts with a standard system design and requirement refinement process, but at the same time available components will be selected and aggregated. It fosters reusability of existing components and promises to accelerate the integration process. Only broad system architecture and completely new components have to pass the entire square from the top to the bottom.

[CCL06] presented an adapted V-model, that basically corresponds to our notion of the integration of reusability concepts in developments processes. Parallel to the requirements definition and system design steps of the process this model adds process steps to find, evaluate, and select suitable components.

[Cap05] further refined the process model to the Y-model. It is shown as Figure 3.7. The model does not only focus on the software creation but also emphasizes the creation, evolution, and management of reusable components. Additionally to the well-known process steps it introduces steps for:

Domain Engineering: Application domains may have a commonalities in vocabulary, processes and techniques. It is the task of the domain engineering to bundle components in context of the domain to reusable consistent packets.

Frameworking: A framework defines the structure and the interrelationship of components within an application domain. This typically is based on semantic relationships between components. Frameworking concerns the creation of such frameworks, but also the application of frameworks to accelerate the development process.

Assembly: It concerns the selection of the actual components or frameworks from the specific application domain. The model describes the direct assembly into the product as favorable strategy, while it also mention an alternative way of influencing the design and implementation phase with the selection and adaption of units.

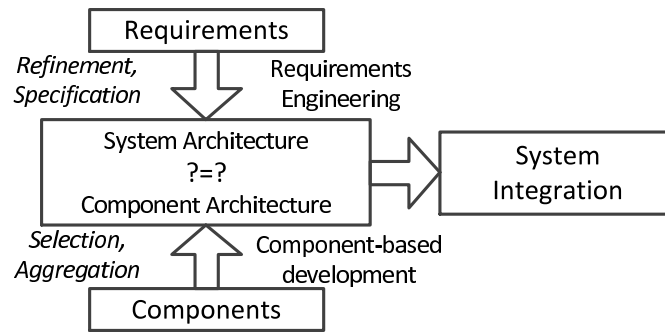


Figure 3.8: General model for component based development: Requirements will be refined to get a system architecture; existing components are selected and aggregated to a component architecture. If the components can satisfy the requirements, the system can be integrated.

Figure 3.8 shows a general model to foster reusability. Requirements and their specifications will be refined to get a system architecture. From the bottom, components are selected and aggregated to a component architecture. If both architectures match in a way that the component composition satisfies the requirements, the process can continue to the system integration.

The corresponding actions are requirements engineering and component-based development. They are described in the following sections.

3.3 Requirement Engineering

According to [Som05] requirement engineering is a structured set of activities to develop a consistent system model that represents the requirements and constraints given by the user. In the second half of this section we will discuss these activities. First, however, we pursue the question what requirements are.

3.3.1 Requirements

The IEEE Standard Glossary of Software Engineering Terminology definition [IEE90] defines requirements as:

1. A condition or capability needed by a user to solve a problem or achieve an objective.
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
3. A documented representation of a condition or capability as in (1) or (2).

While the first item is too general and the third item concerns documentation, the second part of the definition matches the view of requirements as we see suitable for the WSN engineering. However, this definition still considers requirements entirely as conditions or capabilities, while it ignores the effect of the environment. In particular

requirements in the domain of WSNs typically entail the technical requirements, but also qualitative parametrization of functions and a description of the environment in which the system should work.

In this context [Rom85] provides an extensive taxonomy of requirements and their engineering. It differentiates requirements in:

Functional requirements capture the nature of interaction between the component and its environment.

Non-Functional requirements restrict the number of solutions one might consider.

They are also referred as constraints. They may be qualitative restrictions, physical limitations, equipment constraints, but also regulations from government or standards.

As discussed in the previous chapter, in WSNs non-functional properties are the primary discriminator between systems. This is in particular true for security requirements which are mostly non-functional.

However, as [Gli07] explained in an example that non-functional requirements can be rephrased to functional requirements. They stated that the clearly non-functional requirement “the access must be secured” can be rephrased to a functional requirement, such as “the login function of the database needs a password”. In fact many security-related requirements have to be rephrased to functionalities during the design process. This however implies that there is no point in a strong differentiation between functional and non-functional requirements.

[Wie99] proposed another differentiation of requirements in

Business requirements are wishes and goals of the people in the environment.

System requirements are more technical nature and specify the environment, the behavior, and qualitative properties. They are derived from the business requirements.

In context of this thesis the emphasis is put on the system requirements. However, since it is a goal that eventually end-users and domain experts should be able to configure WSNs we have to consider the business requirements as well. As part of the requirements definition phase an automatic transformation from fuzzy business requirements to technical system requirements is proposed.

Metrics and Scales

To set, compare, and compute requirements metrics and scales are needed. This is true for all properties or attributes in the system. Depending on the phenomena the property or requirement describing such a domain can be represented in different shapes. For instance, the type of an application will be expressed differently than energy consumption. Even for the same subject sometimes different representations occur. Energy consumption can be expressed as absolute value (e.g. in Joule) or in some sort of classification (e.g. A, B, C classification for home appliances). The question on how to map real world phenomena on scales has been part of measurement sciences, which also originated the widely accepted classification of measurement types [Ste46]. A brief overview of the scale types is shown in Table 3.1.

Table 3.1: Overview on Scales of Measurement.

Scale	Supported Operations	Example
Nominal	Identity	application type (Agriculture, Medical)
Ordinal	Compare	Efficiency Class (High, Medium, Low)
Interval	Average	Temperature in Deg. Cel.
Ratio	Multiplication	Energy Consumption

3.3.2 Requirement Elicitation Techniques

The first step of the requirement definition process for the user is the entering of the requirements. For the elicitation of user requirements relatively few formal approaches exists. The most typical form of collecting requirements are text documents without formalism. Such product specifications are not applicable for automatic requirement processing. Methods for formal acquisition of requirements include:

Formal textual description: The required properties of a system are described in a formal configuration or system description file format.

Dialog boxes: Dialogs can help to enter a properties of the target system. They are easily accessible and understood. Since dialogs have to be defined, dialogs are typically static.

Interviews: Formal interviews or questionnaires are reliable means to collect requirement information. The user is asked a set of questions whose answers can be inferred to technical requirements. Such questionnaires can be static or dynamic. Dynamic interviews are often referred as Laddering [RM95].

Catalogs: To promote requirements reuse, specifications of requirements can be stored in catalogs. In such catalogs the requirement specifications are categorized in domains. Based on the user domain a specific subset of requirements has to be defined by the user. This process does not only favor a complete unambiguous requirement definition but can also support the following steps of the development flow, since the requirement types are well-defined and established. This method is also known as SIREN (Simple REuse of software requiremeNts [TNMG02]).

From these methods the catalogs are the most appealing way of defining requirements for WSN applications. In fact this approach has already been pursued for WSNs. For instance Römer and Mattern [RM04] defined a list of properties that allow the characterization of WSN applications. Their design space contains 12 major dimensions, some of which contain several sub-dimensions. The classification considers constraints, but also more technical properties of an actual deployment, like network size, network topology, and heterogeneity. Basically the WSN taxonomy presented in Section 2.1.2 is already a small catalog that allows a user to specify the system.

Another technique to collect requirements from users, which is extensively used in the configuration of software product lines are feature models, which are discussed in more detail in the following.

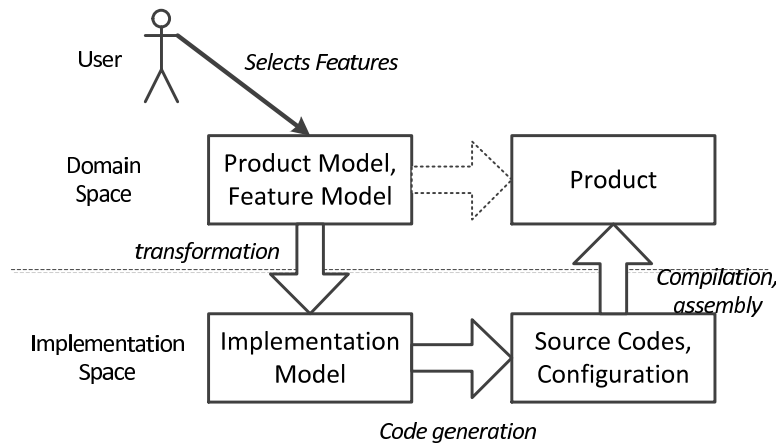


Figure 3.9: Product/Feature Models provide abstraction in the domain space. It allows users to select features to configure the product. Actually the configuration needs a transformation to a technical implementation space which generates compilable configurations from implementation models.

3.3.3 Feature Models

Features in context of Feature Models (FM) are prominent or distinctive user-visible aspects, quality, or characteristics of a system [EKMM10]. Czarnecki [CA05] defined a feature more general as “a system property relevant to stakeholders to capture commonalities and discriminates among systems.” It may denote any functional or non-functional characteristic at the requirements, architectural, component, platform, or any other level. The latter definition allows to consider all distinguishable system aspects as features. It includes quality aspects, security, hardware properties and algorithms. From this perspective features can clearly be applied in WSNs. However, in order to be applicable in our development model they have to support the composition in the implementation space.

The fundamental goal of FMs is to identify valid combinations of features in product lines. Since FMs are often applied to model product configuration, they are sometimes referred to as Product Models. Features of products usually work in the same domain level as the actual products, so that FMs are valid configuration tools for users. Figure 3.2 illustrates the idea. The user selects features needed for the intended product. The FM resolves dependencies of the features and identifies conflicts. Finally a product configuration is presented to the user. While in some domains a direct translation from FM to product is available, typically the FM has to be transformed to the implementation space. The logic in the domain space is similar to ‘Feature A needs Feature B’. The rationale in the implementation level is rather component and implementation-specific. As these implementation details should be hidden from the user, FMs are a suitable means for configuration of complex systems as WSNs. That is why FMs are applied in software product lines [PBVDL05]. There, FMs help managing large software systems with a variable set of common features.

FMs as composition tools:

Feature Models are able to maintain systems with a huge set of features, e.g. in the car manufacturing industry [CA05]. To manage such complex systems FMs apply feature diagrams to control the space of feature combinations. A Feature Diagram is a graph that performs a hierarchical decomposition of a system to the basic features while intermediate nodes are compound features. The relation between compound features and their children can be optional, mandatory or logically associated. The strict binary logic allows the application of straight propositional logic to evaluate the effect of selecting a specific feature. The result of this logical solving process is the description of a system as a set of features which may be implementable to a system. The result does not express how the system can be implemented. Therefore, FMs have to be considered rather as support tool to assist users entering requirements and to identify conflicts on domain level. For a small example it has been shown [LKL07] that FMs can be applied to configure a WSN middleware with a few features and components. However, generally at the current point of WSN maturity the systems cannot be considered as customizations of one general system that can be configured isolated from the actual implementation model. WSNs in our understanding are dominated by attributes that cannot easily translated into propositional logic. This issue becomes more relevant since the influence of environment also influences features and their quality. That is why a static model of features and their interrelations does not appear to be feasible for WSN systems in the moment. This may change in future when for toolboxes with a limited set of components and function a FM can guide end users selecting application configurations.

3.4 Component-Based Development

Component-based development (CBD) is the bottom-up part of the development strategy shown in Figure 3.8. There, the construction of the system is based on aggregation of existing components.

In this section we first define what components are and how they are represented. Since for composition purposes modeling the behavior of the components and the system is important, modeling aspects will be discussed in the following. This section is concluded with existing examples of CBD frameworks presented in the literature.

3.4.1 Components

This section defines the component concept as it is the basis for the CBD. Based on definitions given in [Bal01] and [SGM02], we define a component as:

self-contained building block with a coherent functionality, so that they can be deployed independently and are subject to composition by third parties,

with well defined interfaces, which provide the functionality,

which has explicit context interdependencies, meaning that the effect of the unit on system and parts of the system are documented.

Balzert [Bal01] also required that components are black boxes. We will see later in this section that components may have other encapsulations than black-boxes.

Practically a component is an abstraction of (implemented) functionality. They can represent software modules, such as functions, implemented classes, implemented algorithms, or services, but also hardware modules, which can be hardware systems (sensor nodes), hardware components (chips, sensors, memory extensions), or implementable hardware description modules. So, the granularity of a component can be very fine (one mathematical operation) or coarse (an entire system).

Many authors stress the concept of contracts components should provide. [DB00] discussed the following contract elements:

Context dependencies contain a definition of system properties which have to be present so that the component will work.

Semantics provide a full definition of the operations and the behavior of the component, including pre- and post conditions and inter-component interactions.

Non-functional properties describe qualitative properties of the component, such as space and time, but also for example security, reliability or throughput.

Configuration contains the parameters of the components in case it can be configured before instantiation.

In this thesis we use the term meta-information as generalized form of contracts. In Chapter 5 they will be expressed with properties and relations.

Interfaces

Components may use functionality of other components, or provide functionality to other components. Interfaces are the means to describe interaction between components. Interfaces can be considered as ports of the component. The ports provide the functions of the component, and other components have ports to use the functions. Many approaches consider interfaces as the complete description of the behavior of the components, for example in GRATIS [LBM⁺02] – a component framework discussed in detail below. In [CdAH⁺02] an extensive interface compatibility checking was shown. It contains a formal methodology to detect errors in the interfaces of software modules needs the interfaces and the behavior of the components modeled as state machines.

However, typical interface descriptions contain only syntactic representations of the data types, for example the nesC interfaces. Then the name of the interface and the list of named data types describes an interface. The semantic of the names and types should ensure correct usage, which is a rather pragmatic assumption.

Encapsulation

In CBD the knowledge a model contains about the inner workings of the components is indicated by a color.

Black box component models do not provide direct information about the inner structure and functionality of the component. For software it means that the source code is not accessible. All what is known about a component is determined by the interfaces and additional meta-information, such as documentation.

White box component models rely on the availability of the source code. All information about inner structure and functions of the component are accessible for and changeable by the developer.

Gray box component models, sometimes also referred as invasive component models [ABm03], allow to extend existing components after their creation, typically without access to the source code. The models are called gray box because the blocks are adaptable, so that they are not entirely black, but usually the full source code is not accessible so that the models are not white.

Invasive composition adapts and extends existing components usually by program hooks. The instantiation of hooks can vary between a "enter your code here" in the source code (which would be a white-box) and a well-directed extension of components over well-defined interfaces. Another way of implementing the gray box methodology in a less invasive manner is by customization properties of the components with parameters. Such an approach is pursued in JavaBeans [TS98].

[DB00] discussed another view on gray-box component composition by presenting a high-level view of the internal behavior to show environmental effects. This does not directly improve the composability of the component but bypasses the intransparency of the black box model without using the full source code.

Composition

Components can be aggregated in order to combine their functionalities in the system. In this thesis we use the term component composition and component aggregation as synonyms. Components are composed by connecting their interfaces. Generally, a component that uses a specific interface may be connected to another component that requires the same, or a compatible, interface. This implies that both components can communicate to each other via this shared interface. The semantic transition triggered by this composition depends on the component model. In the simplest case the composition contains the semantics and properties of the individual components. In most cases the properties may affect each other which makes the predictability of behavior and properties of composed component to a non-trivial issue. It will be discussed later in this section.

Predictability

One central issue in CBD is the reasoning about the behavior of the composed components. Considered two components, A and B, are well-specified regarding their structure and behavior, what can we say about the structure and behavior of the composition of A and B? Lau [Lau01] discussed a priori reasoning in the context of component based software development. He defined the terms a priori reasoning and a posteriori reasoning. *A posteriori reasoning* means that the correctness and the behavior of the program is determined after the program has been constructed.

A priori reasoning in contrast takes place during the assembly process. It allows to predict the behavior and its correctness before the actual construction of the program.

Obviously, it is the goal of any component composition process to infer as much information as possible before assembling components and testing the resulting system.

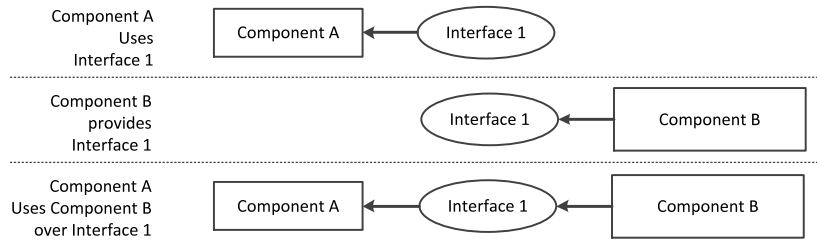


Figure 3.10: Graphical representation of components and interfaces: Components are rectangles, Interfaces are ellipses. The direction of the arrow between indicates if the interface is used by the component (arrow to the component) or if the component provides the interface (arrow to the interface)

In context of this thesis we define a priori reasoning as each assessment of correctness and behavior of the composition based on information that exclude actual source code and binary compiled code but includes all meta information about its structure.

It is further assumed that the meta information are significantly less complex than the actual code, and ideally can be processed automatically.

Graphical Representation

Components and their compositions can be represented graphically. Several ways exist to draw components, interfaces and their connections. It is common practice to represent components as rectangles. The representation of interfaces differs significantly. nesC typically is represented with direct arrows from providing component to using component, annotated with the name of the interface. In UML component diagrams interfaces are small balls called sockets. In hardware description models interfaces are ports illustrated in side the components. Figure 3.10 shows the representation we use in this thesis. It emphasizes that interfaces are individual, self-contained objects, rather than just a port of a component.

3.4.2 Model-Driven Software Development

Model-Driven Development (MDD) raises the levels of abstraction from implementation details. It increases the readability of a system and instead of analyzing complex source codes, users can view a system design made up of models, instead of specific programming languages, at a high perspective. So they provide an understandable view of complex problems and their solutions so that engineers can ignore details in functions and focus on their actual interests.

Moreover, it saves considerable time to modify systems at a conceptual level before actually putting a design into implementation. Thus, models can be considered as conceptualization of the real word, while source codes denote the actual implementation of a design. The separated handling of structure of the program on one side and model of the behavior and the properties on the other side is the dominating paradigm throughout this thesis. Therefore, MDD relies on the precondition that components are modeled. This enables partially automatically integration flows and allows checking of compatibilities and behavior before the actual integration. As conceptional example Model Integrated Computing [SK97] combines a domain specific modeling language with modeling tools

Table 3.2: UML four layer meta-model. Higher layer models instances of the lower layers.

Layer	Name	Description	Example
M3	Meta-metamodel	defines metamodels	MetaClass
M2	Metamodel	instance of the meta-metamodel defines the model	Property Component
M1	Model	instance of the metamodel model of the information domain	code size security degree
M0	user data or object	instance of the model	implemented system

for the given application environment. The concepts are motivated by the idea to provide tools and modeling concepts usable for generic application designer or domain experts.

Many researchers have recognized MDD and model transformations bridging the gaps between models at different abstraction levels or between models and source codes as a key challenge of the Model Driven Architecture. Intensive research has been conducted on the process of model transformation. Various transformation languages and tool suites have been developed, although most of them are at experimental stage yet to be applied to industrial practice. An example of such tools is the Eclipse Model Development Tools [MDT10] and the Eclipse Modeling Framework Project [EMF10]. Naturally, the model transformation process is complex, requires good expertise as the models are complex and some are still under development, and there are no criteria for ensuring the consistency and accuracy of the transformation between the different abstraction levels.

In recent years MDD has become subject for standardization activities in the industry. In particular the Object Management Group (OMG) [OMG11] is actively developing standard architectures for MDD.

Meta-Models

When working with models to drive development processes it is imperative also to define syntax and semantics of the resulting models. Following the standard four-layer meta-model architecture described in the UML specification [BRJ05] this definition is the layer M2: the meta-model. The general notion of the meta-model architecture, which is shown as Table 3.2, that each higher layer is able to describe the next lower layer in the hierarchy. The lowest layer M0 is the actual data or the real system. The highest layer is the meta-metamodel, which is able to describe metamodels, but also meta-metamodels. In the context of this thesis the layers M1 and M2 are most important.

3.4.3 Roles in the CBD

Component-based development involves several roles, participating in the development process. [Vit03] discussed four roles:

Developer develops new components,

Framework builder sets the infrastructure and the models and choses the set of domain-specific components,

Assembler selects components and assembles them into applications,

End user uses the resulting application.

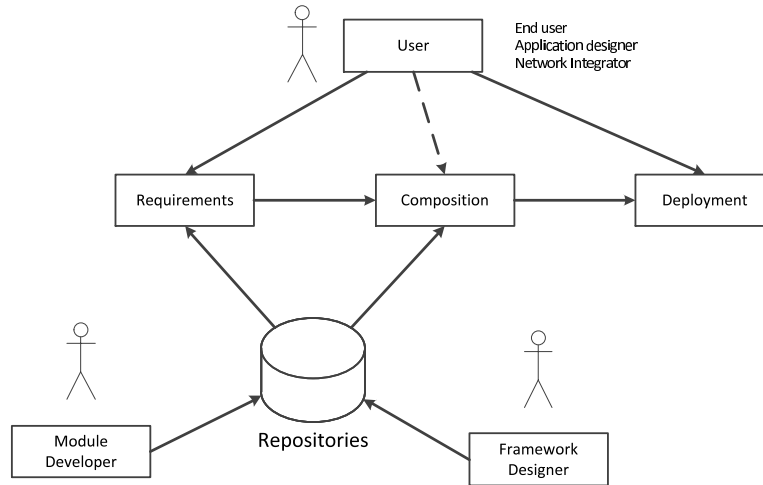


Figure 3.11: Roles in the component-based development: The user controls requirements definition and the (automatic) composition with the aim to deploy the network. Repositories for requirements and composition are managed by framework designer. They add the components developed by Module developers.

In [MSW⁺04] a fifth role was proposed with the end user developer. This role addresses no programmers, but technical and domain experts, able to tailor application at runtime. While in practice the differentiation of the user role may be important, in this thesis only three roles will be actively considered: The component developer, the framework builder and the user. Figure 3.11 illustrates the relations between the roles and the process steps in the CBD. The user in the context of this thesis is the person or organization who will compose and deploy the sensor network. In the future that can be an end user or application domain expert who wants to apply WSNs to accomplish a specific task. Today it is more likely that the user is a sensor network engineer who assembles the network for the actual end user. However, in the thesis we do not differentiate between end-users and their deputies.

3.4.4 Examples for CBD Frameworks

Literature shows a wide variety on CBD frameworks. In this section we focus on frameworks with a relation to the development of WSNs.

VEST [SZP⁺03] (Virginia Embedded Systems Toolkit) focuses on the development of effective composition, configuration, and the associated dependency analysis. The tool helps the developer select and compose software components to a product. The analysis part even allows checking security properties, though it does not provide formal proof of correctness. Rather it applies key checks and analysis to avoid many common problems.

HULOTTE - Meta-Framework [LPM⁺09] presents a framework which bases on a component model with a configurable tool-set. The component model is described by a generic meta-model. It allows reusability of the concept and tools in different domains. The focus of the work has not been the automatic composition but rather a tool-supported

integration process of components as well as architectural patterns.

While the eventual direction of HULOTTE is not directly applicable in context of this thesis, the generic meta-model is interesting also for WSNs.

A comparison of component frameworks for embedded systems was given in [HPB⁺10]. Since embedded systems are similar to WSN systems in several aspects the methodologies and properties of such component composition systems can be valuable also for WSNs. According to this study the most promising frameworks are THINK [FSLM02] and SOFA HI which is based on SOFA2 [BHP06]. Both frameworks are hierarchical component frameworks with a large tool-support. They both support the C programming language and contain a powerful meta-model that allows explicit formal description of the model behavior. THINK was originally developed as tool for component-based management of kernels from operating system, and now is a full toolbox for the development of embedded systems. However, according to [GLVB⁺03] its performance cannot compete with the results of nesC. Since SOFA with its powerful operating system which supports run-time configuration and Contrary to THINK, SOFA supports development of real-time applications. However, this is also reason for a particular run-time operating system needed for SOFA. Also THINK relies on its own compiler to generate byte code, which makes it less attractive for immediate use in WSN projects. Neither of both frameworks support automatic composition.

GRATIS [VMD⁺05a] (Graphical Development Environment for TinyOS) is a model-based approach to the component-based development of sensor network applications, and thus was already mentioned in Section 3.1. It bases on the Generic Modeling Environment (GME) [LBM⁺02] –a meta-model toolkit for Model Integrated Computing. It is able to map TinyOS code to GME concepts and thus provides a set of sophisticated graphical tools for the development of TinyOS-based applications.

Beside the graphical representation of TinyOS code, the main advantage of GRATIS is the ability to create and maintain models from real nesC code. The emphasis on these models is the definition of the interfaces. Contrary to classical interface description and compatibility checking, GRATIS does not only check static typing of the interfaces but also respects dynamic aspects of the component interfaces. The technique, based on Interface Automata [DAH01], allows verification and validation of component interfaces and their interaction.

GRATIS emphasizes interface specifications and thus is a great help for developers of TinyOS applications. Using the models and linked module source codes GRATIS can also generate actually compilable system code. However, since GRATIS does not model the actual behavior of the modules, it does not support the component selection process. Recently a new version of GRATIS was presented in [VSL⁺10]. GRATIS++ additionally to TinyOS supports integration of hardware components written in VHDL in the WSN development process. In this approach hardware and software modules are uniformly interchangeable. Within the hardware/software design space exploration, alternative implementations of particular components can be automatically evaluated and proposed to the user.

3.5 Conclusions and Implication for the Thesis

In the first part of this chapter we studied existing programming and development methodologies for WSNs. The discussed abstraction levels range from hardware abstraction to a single abstraction layer for the whole network. While many approaches are very interesting, they often focus on programming aspects which are not important for the task of configuration and integration. Additionally, it is notable that quality of service and security has not yet moved in the focus of the tool-developers.

From the properties exposed by a brief survey on existing development tools we can extract a list of beneficial attributes a WSN configuration tool should possess:

- Offline configuration and compilation is mandatory to achieve sufficient performance on the tiny nodes.
- Component based composition approaches are most suitable to foster reusability. Models connected with the components may help to assess non-functional properties of the system.
- An existing hardware abstraction is important to implement the components on the heterogeneous hardware platforms. Additional abstraction may be delivered with the middleware components.

In the second half of the chapter methods and approaches from the domain of software engineering were evaluated on their practicability in the context of WSN development. We found out that both top-down as well as bottom-up approaches have to be supported for the intended development flow.

The key aspect in the top-down process is the elicitation of the requirements and the translation to technical usable requirements. Among classic elicitation techniques we considered feature models as option to configure WSNs. Even though FM can be applied to solve feature dependencies on domain level efficiently, they are not considered as direct solution for the application composition problem in this thesis.

The bottom-up process steps concern the composition of existing components to systems satisfying the refined requirements. For this we studied the state of the art in component-based development. Result of this analysis is a understanding of the core concepts needed for the composition toolkit introduced in Chapter 5. The concepts include a general flow with assigned roles and component models with interfaces. An important aspect in CBD is the separation of structure and semantic. While the structure of a program can be described with components and interfaces, the semantic usually needs additional meta-information. These meta-information eventually can be used to describe properties that allow a prediction of the behavior of the resulting system.

A brief survey on existing CBD frameworks concludes the chapter. The examples do not contain a perfectly suitable framework for our intended configuration tool. Instead they could reveal valuable information on the practical application of CBD concepts such as meta-models, model checking and design space exploration.

The studied frameworks focus on support for programmers performing component based development. Automatic composition for user-given requirements is not supported. As answer to this issue, in Chapter 5 we further refine a holistic process combining the top-down and bottom-up approaches, discussed in the second half of this chapter, in order to realize an automatic configuration framework as it was motivated in the first half of this chapter.

Chapter 4

Security Engineering for Wireless Sensor Networks

Understanding of security is mandatory if it is the goal to configure security. However, security is a non-functional property that is non-tangible and cannot be measured directly. Engineering security –even for standard IT systems– is a highly complex task without a silver bullet. WSNs are additionally challenged by the diversity of potential attacks and the limited resources the nodes have to protect themselves. Therefore, this chapter does not only provide an overview on security objectives, threats, measures and models, but also describes specific attacks and countermeasures in the domain of WSNs.

In the first section of this chapter we study different views on the term security for IT systems in general and WSN systems in particular. While terms like confidentiality, integrity, or reliability are favorable to describe general security goals, the terms do not directly help developing or assessing secure systems. For the connection of the terms with actual properties of the system security models are needed. That is why we discuss models that combine technical properties of the system with goals of the user and goals and capabilities of potential attackers.

The favored security models relate the security of the system to the assumed attacker and the potential actions. This in first place requires an understanding, and therefore analysis, of attackers, the attacker motivation, and attacker goals. Such an analysis is given in the second section of this chapter. It concludes with an attacker classification model that is used throughout the rest of this thesis.

The following sections present an overview of actual attacks and general countermeasures in the domain of WSNs. As specific mechanism to combine functionality and security in WSNs, secure in-network aggregation (INA) is discussed in more detail. The introduction and analysis of specific algorithms to realize secure INA is the basis for the practical examples for security composition in Chapter 7.

In the final sections of this chapter, holistic approaches to configure security in WSNs are studied. Such security toolboxes combine several security mechanisms and promise to provide easily usable, pre-configured security. Finally, we extend the survey on methods to engineer security, from WSNs to general IT systems. Described as part of this survey, attack tree analysis and specific security metrics are valuable tools we will further apply in the novel security models introduced in Chapter 6.

4.1 Security Nomenclature

A significant problem of security engineering is that even fundamental terms are not clearly defined. Already the term security is not defined unambiguously. [HR06] considers it as the absence of unauthorized access to a system, while the majority of researchers consider security as composite of other security-related attributes [ALRL04] [And08]. More general definitions relate security to the degree of protection against dangers and loss. However, the actual meaning varies with environment, involved people, perspective and goals. While for politicians security depends on international relationships, and for cars security concerns the safety of the passengers, for computer systems security is mostly about dependability and data protection.

4.1.1 Security Attributes

Literatures [ALRL04] [Fir05] consider security as composite of primary and secondary attributes. The primary attributes are confidentiality, integrity and availability.

Confidentiality is the absence of unauthorized disclosure of information. It means that unauthorized subjects should not be able to access sensitive information. In many sensor network applications sensitive data is gathered. For instance in health care and in industrial applications data confidentiality is of uttermost importance. In some military and homeland security application scenarios it could be a goal to conceal the presence of the network.

Concealment, Privacy and Secrecy, in their meaning that specific information shall not be disclosed, are rather technical or semantical refinements of the term Confidentiality. Anderson [And08] and Hasselbring [HR06] among others, discussed partially conflicting views on the terms. In this thesis they are used as synonyms.

Integrity is the prevention of unauthorized modification, amendment, or deletion of information. If an unauthorized subject performs such actions the invalidity of the information should be detected. Since gathering of data is the primary objective of most WSNs, ensuring the correctness of the collected information is the most important security concern. Corrupt data may turn a network useless if its task is to collect reliable data. This is true even for habitat and animal monitoring networks which generally have low security requirements. However, data integrity becomes imperative in applications that use the data to control actively, for instance industrial or agricultural automation.

Availability is the ability to ensure that system and information are readily accessible all time within the required parameters. Availability does not concern data but the services. It means that no natural or intentional event should affect the readiness of the system to operate correctly. In WSNs availability is particularly important in applications that need the sensor input to monitor the health status of people. For example in the firefighter monitoring scenario the data should be transferred correctly even in a harsh environment with high temperatures and presence of water and steam.

The research community agrees on these three primary security attributes. Additionally, secondary attributes are described to extend, refine, or combine the primary attributes. Most common secondary security attributes are [ALRL04] [DKK07]:

Authentication is the ability to ensure that the source of information is accurate and unchanged. Basically it is the integrity of the source. Authentication is important in all WSNs. Without a certain degree of authentication everyone could access and reprogram the network.

Authorization checks whether the peer has permission to conduct specific action. The check is based on authentication, while the restricted operation may affect the confidentiality or the availability of the system. Authorization becomes important in WSNs with multi user access levels. For example when operators are allowed to configure the system while user may only use specific services.

Non-Repudiation is the non-deniability of an action. Currently Non-Repudiation is no major concern in WSN applications. This property may be important for future applications that include monetary services in the WSN or if the sensor readings are used for forensic data gathering to support insurance companies after accidents.

Additionally to the information-centric security terms, many security attributes concerning the correct behavior of the system have been proposed. A set of behavioral attributes to realize trustworthiness is discussed in [HR06]. They are:

Safety is the absence of harmful environmental consequences. It means that the network system including the monitored or controlled infrastructure must not effect events that could cause significant physical damage. *Safety* in WSNs becomes more important in actuator networks and in networks that actually control machines. Due to the physical real-world effects, safety cannot be expressed entirely as part of the IT system. However, a secure IT system is precondition for a safe system.

Correctness is the absence of improper system states. Further, a correct system works as it was specified, while each possible state was specified. This attribute mainly refers to the correct implementation of the system.

Reliability is the ability of the system to provide the required service consistently without degradation or failure. While *Correctness* refers to the correct implementation, *Reliability* describes the correct execution. Since the correct implementation is a prerequisite for the correct execution, *Correctness* is just a specific partial aspect of *Reliability*. Also *Availability* as described above is a partial aspect of *Reliability*, because *Reliability* requires the consistent service as it is defined for the *Availability* quality.

Performance describes the response time, throughput, and other time- and speed-related quality aspects. *Performance* basically is related to *Availability* and *Reliability* because in the moment the required performance is not met the availability requirements are not fulfilled.

Trustworthiness and Dependability: *Trustworthiness* is the assurance that a system will perform as expected. Anderson [And08] described a trustworthy system as a system that won't fail. *Dependability* is used similar to *trustworthiness*. In [ALRL04] it is described as composition of *Availability*, *Reliability*, *Safety*, *Integrity*, and *Maintainability*. For both *Trustworthiness* and *Dependability*, information security is a prerequisite.

SECURITY		
Confidentiality	Integrity	Reliability
<i>Outgoing data cannot be misused</i>	<i>Incoming data is correct and can be correctly assigned</i>	<i>The system does the things right</i>
Concealment Privacy Secrecy	Authenticity Authentication	Robustness Availability Correctness QoS Performance

Figure 4.1: Classification of security-related terms: Security objective can be decomposed in Confidentiality, Integrity, and Reliability, which each describe distinct objectives and can be decomposed to other attributes.

As result of this description, *Safety* and *Trustworthiness* are terms that are out of scope of the IT security. *Reliability* is a superclass for *availability*, *performance*, and *correctness*, which qualifies *Reliability* as primary security attribute.

Conclusions

Reusing the terms danger and loss to the information technology, we can state that security is the protection against the danger of an unintended behavior of the system, and protection against loss of information. Following this definition, the correct behavior of a system can be secured if it is ensured that no event can bring the system in an undesired state. The loss of information can be prevented if all outgoing information is protected against the loss.

Since events can be caused both in as well outside the system, this leads to three sub-classes:

- ensuring that outgoing information cannot be misused,
- ensuring that incoming information is correct,
- ensuring that the system does the right things with the information.

This basically correspond to the three primary security attributes Confidentiality, Integrity, and Availability. Other data-centric security attributes can basically be inferred from confidentiality and integrity as described above. However, to highlight the requirements concerning behavioral correctness of the system we replace the term Availability with Reliability. It is our understanding that availability is required for a reliable service. This extends the classic view on data-centric security by behavioral aspects such as correctness and assurance of qualities.

Figure 4.1 illustrates our security term convention: Security is the superclass of the security objectives Confidentiality, Integrity, and Reliability. Each objective describes a distinct property of the system. Each of the three objectives has similar or deduced terms. They are shown in the bottom row of the table.

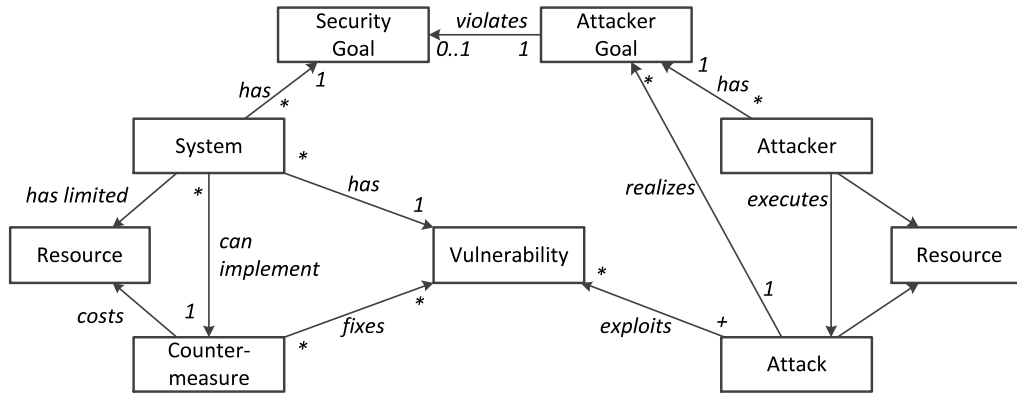


Figure 4.2: Ontology of the security terms used in this thesis.

4.1.2 Technical Security Terminology

The abstract security terms discussed in the previous section are important to find a common language that helps to express security-related goals. The terms however do not help to develop or assess security of systems. For this purpose more practical models are needed to describe security-related properties of systems. It is the goal of this section to introduce the entities and objects needed to discuss the security status of a system. Beside the actual objects the relations between the objects are interesting. We use ontologies to specify the relationship between the objects.

Security Ontology of this Thesis

The security meta-model that will be used in this thesis is shown as Figure 4.2. The following paragraphs discuss the objects, their possible instances and the relations in more detail.

System: The *system* is the IT structure that performs specific tasks and which should be protected. In this thesis the *system* is the wireless sensor network with all its hardware and software components.

Security Goal: A *system* has many system requirements. The subset of security-related requirements of the system constitute the *security goals*. Generally *security goals* can contain technical, organizational, or management constraints.

Attack: An *attack* is an activity with the intend to violate security requirements of a system. Typically *attacks* exploit *vulnerabilities* of the *system*. Examples are destroying the node's hardware, or draining battery by sleep deprivation.

Vulnerability: A *vulnerability* is a security-related weakness of the system. Typically *vulnerabilities* are exploitable to harm the system and to violate security needs. *Vulnerabilities* are properties of a *system* that can be exploited by *attacks*. Exploiting *vulnerabilities* is associated with costs for the attacker.

Countermeasure: *Countermeasures* are properties, methods, protocols, or components that fix or disable *vulnerabilities* of the *systems* or prevent *attacks*. Fixing means

either disabling the vulnerability completely or at least increasing the efforts to exploit the vulnerability significantly.

Protection Resources: Typically *countermeasures* have a cost for the operator of the system. Such costs are development and licensing costs for the countermeasure, maintenance, and additional system requirements such as memory, processing time and energy consumption.

Attacker Goal: An *attacker goal* is an intention of the *attacker* that violates a *security goal* of the *system*. In order to validate the security of the system we can assume that attacker goals are the inverse of the *security goals*. More specific attacker goals are discussed in Section 4.2.2.

Attacker Resources An *attacker* needs *resources* to execute an *attack*. Such resources can be money, technical equipment, time, knowledge, but also motivation. Motivation is a budgeted resource because an attacker needs sufficient motivation to execute an attack. *Countermeasures*, such as prosecution, can reduce the motivation. The effect is well-known for classic resources such as money or time, where *countermeasures* exhaust the *attacker resources*.

The ontology depicted in Figure 4.2 also contains cardinalities between the objects.

- A *system* has any number (*) of *security goals*, any number (*) of *vulnerabilities*, and can implement any number (*) of *countermeasures*
- An *attacker* has any number (*) of *attacker goals* and executes any number (*) of *attacks*.
- An *attacker goal* can be realized by any number (*) of *attacks*, and violates at least one (+) *security goal*.
- Any number (*) of *vulnerabilities* can be fixed by any number (*) of *countermeasures*.
- Any number (*) of *attacks* exploit at least one *vulnerability*.

This ontology does not directly address other common security terms:

Security Policy The term security policy is already used ambiguously. It may represent security goals of the system but often it is connected with actual mechanisms to implement security. While our notion of a security policy is rather the former one, we avoid that term and use security goal instead.

Anti-Requirement: The term *anti-requirement* was coined by Crook et al. in [CILN02]. An anti-requirement is a requirement of a malicious user that subverts an existing Security Goal. In our model *vulnerabilities* are *anti-requirements*, because they have to be present for an attacker to attack the system.

Risk: The concept of *risk* is used to express a degree of uncertainty. While technically neutral, risk is mostly associated with adverse events. Risk may be measured in percentages (probability of the event) or in money (expected loss).

Resistance: Resistance is a property of the system to withstand specific attacks. In our ontology countermeasures are resistances since their presence expresses the resistance of the system against an attack.

Threats: While an attack is an actual activity, a threat is the potential attack. In spite of this distinction, in context of this thesis, attacks and threats are used as synonyms. [Joh10] discussed the relation between vulnerabilities and threats. He noted that systems will never be 100% vulnerability free. But as long as the vulnerability cannot be exploited by a specific threat it is irrelevant for the security of the system. He further inferred that there is no one-on-one mapping of vulnerabilities to threats. This notion corresponds our understanding of attacks.

Other Security Ontologies

The security ontology was selected after reviewing several other security ontologies and meta-models proposed in literature.

[FF10] presents the IRIS (Integrating Requirements and Information Security) model for usable secure requirements engineering. They divided the model in five views: task, goal, responsibility, risk and environment. The task and responsibility models are focused on the persons and describe what is done by whom to accomplish the use cases. While for larger multi-user systems these aspects are essential for modeling security, for WSNs they can be ignored.

More interesting are the Goal and the Risk since they describe what is necessary to accomplish the goal and how it is threatened, respectively.

The concept of the environment is rather a filter than an active descriptor. For example the environment is used to combine only threats and vulnerabilities that apply for the same environment.

In [Cha05] a more simplistic ontology with just six classes on the top level is given. There an *actor* has a *motive* and gives *input* which initiates *threats* which result in *consequences* for the *system*. Each of the classes is further refined, for example the *consequences* can be loss of assets, illegal access or other outcomes like interruption of service. That way the model –despite its simplicity– is able to model complex relations between actions, threats and potential consequences. However, the model does not respect protection means. Also several terms do not concur with the terminology intended to be used in this thesis.

[EFKW06] combined the ontology describing the IT infrastructure with business processes of the application. The ontology allows to assess the potential damage of a successful attack and to simulate threats against the modeled company by processing the knowledge contained in the ontology. It is an option to extend the ontology applied in this thesis with company processes, while in the current point of research it is not a major objective.

4.2 Attacker Models

4.2.1 Attacker Classification

The security ontology described in the previous section needs as many classes to model attackers as it needs to model the actual system. This already justifies why it is a substantial task in protecting computer networks against attacks to understand the motivation of an attacker. It is the notion that attackers with different motivations react differently to countermeasures and the risk of exposure. A review of different attacker motivations for traditional computer networks is given in [RP09]. The authors differentiate attacker in seven groups. Ranging from Amateurs/Script Kiddies over Malware developers, to Criminals and State-Sponsored Hackers. While in PC systems amateurs have become a lesser threat due to the higher complexity and improved level of security of the systems, WSNs practically invite amateurs because of their low level of security and the relatively low complexity of the systems. Severe hardware constraints of WSNs are reason for systems and software that are usually kept as simple as possible and thus allow even amateur attackers to understand how the system works. The simplicity of the sensor nodes additionally allows to attack such a device with much smaller and less expensive tools and machines. Also the level of security in most WSNs is kept rather low in order to reduce the computation and communication overhead. This is in particular true for network logging and intrusion detection. Since it is possible to attack a network without using standard telecommunication access points, the risk of being detected is additionally reduced.

Technical Attacker Specification

Most work related to attacker schemes in wireless networks use technical characteristics to classify attackers. In the context of vehicular network, [RH07] provided a general classification of attackers that suits all sorts of wireless ad-hoc and meshed networks. A similar taxonomy has been given in [RSS06]. The attacker model contains five dimensions:

Insider vs. Outsider: An insider attacker has special knowledge or access to the network, while an outside attacker has neither special knowledge nor access to the system. A typical outside attacker is a laptop attacker in proximity of the WSN without initial knowledge about keys, structure, or mechanisms. In the context of WSNs the major discriminator between insider and outsider attackers is the presence of compromised nodes which participate in the communication as legitimate peers. Attacks may depend on this property. An outside attacker can turn to an inside attacker if nodes of the network could be compromised.

Malicious vs. Rational: Rational attackers pursue a personal benefit by attacking the WSNs. It may be competitors in industrial scenarios or a house owner tampering metering devices. They respect costs, efforts, and consequences before executing the attack. Malicious attackers have no actual monetary incentive. It is assumed they harm the network basically for fun and without an rationale, which makes their potential attacks much harder to predict. Vandalism is a specific subgroup of non-rational attackers. Another group are so-called Hackivists or Vigilantes, who have a political or idealistic agenda but are not interested in monetary gain [Fal05].

Active vs. Passive: Passive attackers only monitor the network without active interference. Such attacks pursue the disclosure of sensitive information. This information can be caught in proximity of the network while the actual evaluation can be done in a powerful lab. Active attackers may additionally interfere the network. Such interference may entail deletion, modification or creation of information. Thus, active attacks aim – additionally to the confidentiality – on integrity and reliability of the network. While active attacks typically are more complex and theoretically can be detected, the potential impact of such attacks makes them particularly critical.

Local vs. Extended or mote-class vs. Laptop class: A mote-class attacker uses sensor nodes with similar capabilities as the nodes in the WSN to execute the attacks. The nodes may be part of the network. A laptop class attacker uses devices with significantly more power than the sensor nodes. That performance advantage allows the attacker to execute attacks that rely on fast and complex computations. Possible attacks include sending false answers faster than the actual sensor nodes, or trying to increase the load on a set of nodes by frequently formulating complex requests. In the uttermost extreme a laptop attacker is connected to a distributed network of high-powered computers which can be utilized to break ciphers or to perform complex analysis. This is assumed to be an extended attacker.

Radio-only vs. Physical: Radio-only attackers attack the network only by means of radio communication. This corresponds to the Dolev-Yao model [DY83] usually applied to analyze security protocols on classic computer systems. The Dolev-Yao model assumes that the only point of attack would be the network. For most sensor networks that is clearly not the case since unauthorized physical access to nodes is very likely in networks deployed outside [WGS06]. However many mechanisms still rely on the radio-only attacker model and neglect physical attacks.

In this thesis we assume the default attacker to be a physical, laptop-class, active, malicious insider. That is the worst-case combination. Such an attack can be executed with equipment worth less than \$1000, considered standard sensor nodes are deployed unprotected in the field. Anyway, in security analysis we always have to consider the worst case, unless we can justify why a less-extensive attacker model should be used.

Qualitative Attacker Model

The attacker classifications presented so far help to classify the attacker types but they do not express the qualities of the attacks. In the domain of hardware design qualitative attacker models were first proposed by Abraham in [ADDS91] and Weingart in [WWAD90]. Meanwhile the classification scheme is well-accepted industry practice [Gra04]. The attackers are classified in groups depending on their abilities, strengths, monetary budget, and available resources.

Table 4.1 shows the four classes of attackers and the major discriminators of the classes.

Class 0 are no actual attackers but normal users that could violate security requirements by accident.

Table 4.1: Four-class attacker classification.

Class	Attacker	Tools	Budget
0	No actual attack	attack can succeed by accident	-
1	curious hacker	common tools	<\$10,000
2	organized attacker (academic, crime)	special tools	<\$100,000
3	large organized attacker (crime, government)	highly specialized tools, laboratory	>\$100,000

Class 1 is the group of amateurs or curious hackers with a limited budget and no special tools. They are not organized.

Class 2 is the group of organized hackers with a reasonable budget and time and special tools. Typically for this group are academic hackers who perform the attacks mainly for publicity. This group also includes hackers with criminal background motivated by an economic target.

Class 3 are large organized attacker groups. We can find them in large crime organizations, terrorist networks, or in the governments' intelligence structures. They have unlimited resources, outstanding knowledge and highly specialized tools and laboratories.

4.2.2 Attacker Goals

While literature is rich on classifications of attacker groups and their general motivation, there is relatively few focus on the actual goals. Mostly it is implied that attacks aim on stealing or disclosing protected data. However, possession of data or eavesdropping of secret information is only one of many possible goals. While basically the attacker goals depend on the actual application and environment, we still can classify general attacker goals that are valid in WSNs:

Disclosure of information: Attackers may be interested to extract actual data, but also the information of network activity or the existence of the network. Disclosing of information may be the actual target or only a partial goal to implement a more complex attack scheme.

Possession of nodes: Getting access on a computer system is an attack already known from standard networks. There computation power can be used for extensive computation operations, e.g. for breaking passwords or codes. Hijacked systems also can be used as proxy - either as relay to other trusted systems or for coordinated distributed attacks. While exploiting computation resources of WSNs can be neglected due to the limited resources, exploiting trust relations in a network is a feasible scenario.

With respect to amateur hackers, access on sensor nodes allows to do fancy things with the nodes. For example changing LED configurations to demonstrate what's possible, or to use sensors on the nodes for private projects.

Possession of nodes can be differentiated in physical possession and logical possession. Physical possession allows an adversary to steal the node or tamper the

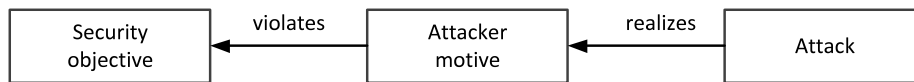


Figure 4.3: Context of attacker motivation: The attacker motive contradicts a security goal, while the attacks realize the motive and thus the violation.

hardware. The effect of logical possession concerns changed behavior typically achieved by reprogramming.

Disruption of the network: One potential attacker goal is the destruction of nodes or network. The actual motivation may vary between vandalism and directed attacks. Also the means may vary between physical destruction of nodes to logical disruption of the communication channel.

Harm to the monitored infrastructure: If the WSN actively influences the controls of a system, the attack on the WSN may be a means to harm the actual infrastructure. An example is the agriculture scenario. There competitors could have motivation to spoil the harvest of neighbors to increase the market price. In industry automation systems manipulation of the control system may have severe impact on the entire facility.

Forging events: In a water pipeline monitoring application a just-for-fun attacker could forge events pretending a hazard just to watch the maintenance team trying to fix a non-existent error. The same is considerable for automatic fire detection in forests.

Change of stored events: If data is stored in the WSN, attackers may be interested in changing the data. For instance information in a WSN that records the status of a road could be manipulated to hide misbehavior in a post-accident investigation [BH08].

Selective forwarding of information If an adversary is in control of a communication path packets may be forwarded or dropped deliberately. For example a trespasser is interested in dropping alarm messages indicating the intrusion.

Personal challenge/prestige: To some extend hacking has always been a sort of competition. Hacking a real-life system is exciting and can increase the social status in the peer group, so that it is clearly an incentive for the attacker.

The attack motivations clearly contradict possible security goals of the system, but they are no actual violation. Attacks are required to realize the attacker goals. Figure 4.3 illustrates the connections. In this context it is important to note that attacks are motivated and need the motive to be executed. Knowledge about potential motives is important to understand the actions of the assumed attacker and to decompose the motives to actual attacks. Tackling security by resolving the motives is a theoretical approach that most likely in practice will not work. Maybe social campaigns and strong prosecution of hacking and destruction attempts can help to reduce the motivation in particular for amateur hackers. Anyway, a technical solution is favorable. It needs an understanding of the actual attacks, which will be addressed in the following section.

4.3 Attack Space on WSN scenarios

The previous section introduced the general security goals of sensor network applications as well as the motivations of the attackers. In this section these general terms will be mapped on practical attacks on WSN applications. Basically, attacks on sensor nodes can be classified in active and passive attacks:

Passive Attacks: For passive attacks the adversary is assumed to do nothing but listening to the transmitted packets. Due to the passivity of the adversary such attacks are not detectable by the network. With well-designed antennas and powerful receivers passive attacks can be executed from significant larger distance than the actual extend of the sensor network.

Active Attacks This kind of attack assumes that the adversary is able to interfere the communication (i.e. to catch, destroy, modify, and send packets) or the nodes. Executing such attacks typically needs more resources and knowledge than passive attacks. Active attacks are detectable. As for the passive attacks, with powerful equipment attackers may execute their attacks on the radio with large distance from the network. Attacks on the nodes naturally need close distance and thus are easier to detect and to defend.

In the following typical attacks on WSNs are described briefly. Only the first two sorts of attacks are passive, while the other attacks are active attacks.

Eavesdropping: is the most common passive attack. It refers to listening to the communication in the network with the goal to extract classified information. The attack is often connected with decoding and deciphering of the messages. Eavesdropping is also precondition for most other attacks where knowledge about the status of the WSN is needed to perform the attack.

Traffic Analysis: Traffic analysis is a specific eavesdropping attack that does not eavesdrop the content of data packets but has the goal to extract valuable information from the presence and timing of such packets. In particular for event-driven systems the presence of an event may reveal exploitable information. For example a perimeter control that only sends alarms in case movements have been detected reveals valuable information to trespassers when a packet is sent. It is either a warning or tells the adversary which packet should be stopped from being transmitted (either by jamming or selective forwarding).

Traffic analysis can also be valuable to detect network structures and to identify important routing nodes. Such nodes are a valuable goal for node compromising attacks.

”Hole” Attacks: The sinkhole and wormhole attacks are active attack techniques executed from inside the network. The attacks compromise the routing structure in the network with the aim to attract network traffic to a node which is controlled by the attacker. In a sinkhole attack the attacker attracts the network traffic by sending forged routing messages. In wormhole attacks additional routes over faster links are embedded in the network by an attacker. Vulnerable routing protocols use the new powerful link (the wormhole) to improve the total network performance. The

problem is that the attacker has control over the link and can eavesdrop packets, control forwarding, but also can easily change or replay packets.

Sybil Attacks: In Sybil attacks [NSSP04] an attacker node incorporates multiple identities to tamper fault-tolerant or distributed algorithms. For instance reputation-based algorithms can be compromised by “ballot-stuffing” to improve the reputation of a specific node, or by “bad mouthing” to reduce reputation of other nodes. Routing protocols may be vulnerable to attacks in which many nodes inject faulty routes. In such a scenario the Sybil attack may be used to enable other attacks, e.g. sinkholes.

Adding false nodes: Adversaries may add nodes to the network that behave like normal sensor nodes in first place but pursue the target of executing malicious actions, such as injecting wrong readings or affecting the network traffic. Added nodes may be stronger than actual sensor nodes and thus are able to perform more complex attacks from the inside of the network.

DOS - Denial of service attacks: Denial of service attacks are basically attacks on the availability or reliability security attribute of WSNs. A variety of DOS attacks are considerable and have been discussed in literature [WS04]. The attacks may aim to Physically destroy the nodes, jamming the radio channel, tampering the network routing, or resource exhaustion.

Forge Messages: Attackers may be interested in the creation of messages and packets in order to inject false information. These information can be actual data but also control information which affect the data flow in the network. For example false routing packets can support “hole” attacks.

Message Modification: Similar to forged messages, existing messages may be changed. This modification can be a change of data content. For instance a periodic packet at a border control could be changed from one detected event to zero detections. Many stream cipher encryption approaches are vulnerable against this sort of attacks: It is not possible to forge a new message but to modify an existing valid message.

Replay Attacks: Replay attacks concern the malicious resending of previously sent packets. In WSNs this means a regular packet is sent at wrong time. This attack works in applications that use encryption or integrity codes without integer information of sequence or time.

Buffer Overflows: On the software executed on the nodes overwriting the end of a fixed length buffer may open the possibility to change data and to inject malicious code. Due to the dominant Harvard architecture in WSNs code injection with buffer overflows is less trivial. However a successful attack on such systems has been described in [FC08].

Node compromise: Physical attacks on sensor nodes bears many risks for the sensor network. A node captured by an attacker may reveal sensitive data, cryptographic keys, or implementation details that may be exploited in combination with other attacks. This can be performed by accessing the content of the memory, attacking the debug/reprogramming interfaces, or extracting cryptographic keys by side-channel

attacks. Such side-channels may be the timing an algorithm needs, but also electro magnetic emissions or the dynamic power consumption. Such attacks can target cryptographic accelerators or the microcontroller. Also data sent over the system bus may be accessed.

4.4 Protection Means - Countermeasures

To cope with the attacks presented in the previous section many approaches have been proposed in recent years. In this section the fundamental means of security and protection for WSNs are introduced. These basic means find application in many protocols to achieve particular technical goals. Table 4.2 provides a short overview of some examples.

Table 4.2: Examples of solutions for security goals, possible attacks and possible countermeasures.

Security goal	Attacks	Solutions
confidentiality	eavesdropping	encryption
integrity	replay, modification, "holes"	secure hashes, signatures
reliability	DOS, node compromise	redundancy, filtering
authentication	sybil attacks, added nodes	signatures

For example a standard means that is supposed to provide confidentiality is encryption. Confidentiality is attained when a plain message is converted into an unreadable form and a reversion is only possible with hidden secret information. Data integrity can be assured by generating a hash value over the content. When the content changes, intentionally or unintentionally, it does not match the hash value.

Authenticity can be obtained by digital signatures or challenge-response protocols. Both approaches are based on the correct encryption or decryption of a message with a key that is exclusively known to one person.

4.4.1 Cryptography

Cryptography (from Greek *kryptos*, "hidden", and *graphein*, "to write") is the technique of converting information into and from a format that is unreadable without secret knowledge. The fundamental cryptographic building blocks are symmetric ciphers, asymmetric ciphers and cryptographic hashes. Other protocols such as signatures or key agreement schemes build on the basic blocks. That is why in this section the cryptographic basis methods are presented.

Symmetric Ciphers

Symmetric ciphers or symmetric key cryptography uses the same key for both encryption and decryption. They can be separated in block ciphers and stream ciphers.

Block ciphers take specific amount of bits as one block. Encryption and decryption is always performed on blocks. The size of a block is typically between 64 bit (DES [FIP99]) and 128 bit (AES [FIP01]). In the domain of WSNs often smaller bit sizes

are used to reduce the computation efforts. State-of-the-art light-weight block ciphers with smaller key length are PRESENT [BKL⁺07], and Skipjack [HYN00], both with 64 bit block size and 80 bit key size.

Symmetric block ciphers can be used for several other cryptographic primitives, for instance for message authentication, hashes and signatures.

Stream ciphers: Contrary to block ciphers, stream ciphers are able to encrypt single bits of the messages. Typically the plaintext is XORed with a pseudo-random bit-stream generated by the cipher. Dedicated stream ciphers typically are faster and lighter than block ciphers, and they message sizes are smaller. Typical symmetric stream ciphers for WSNs are Grain [HJM07] and RC4 [Sch96]. Stream ciphers can be implemented with block ciphers in feedback mode [Dwo01].

Asymmetric Ciphers

Asymmetric ciphers realize a cryptographic approach with different keys for encryption and decryption. One key, the public key, can be published while the other one, the private key, is kept secret. Applying this approach, it is possible for everyone to encrypt a message with the public key and only the owner of the private key can decrypt it. Asymmetric cryptography is often referred as Public-Key Cryptography (PKC). public key cryptography is useful for digital signatures. To assure security, a public key cryptographic system must have the following properties:

- It must be computationally infeasible to determine the private key from the public key.
- It is not possible to reverse the public key cryptographic operation without the associated private key.

These properties can be realized with mathematical one-way functions. These 'trapdoor' operations are performed easily in one direction whereby the calculation of the reverse operation is very expensive. Until today two mathematical principles have been applied for PKC: Factorization and the Discrete logarithm.

The best known adaption of the factorization algorithm for PKC is the RSA algorithm [RSA77]. The best known approach for discrete logarithm PKC is Elliptic Curve Cryptography (ECC). Since ECC in comparison with RSA provides significantly better security strength at smaller key sizes and usually with reduced computation overhead, ECC is the preferred choice for WSNs [LV01]. Anyway the ECC computation is still relatively complex so that hardware accelerators are recommended [PLP08].

Hash functions

Hash functions are functions which map an input of variable length on a number of fixed bit length. This number is the hash value. These functions in particular find cryptographic application in data integrity and data authentication. It should generate a possibly unique mapping that provide the following conditions [Pre93]:

- The input can be any length
- The length of the output of the hash function is fixed
- Computing of the function must be easy

- The function must be 'one way' (preimage resistance). This means it should be computationally infeasible to find a message for a given hash value.
- The function should be collision resistant. This means it should be computationally infeasible to find two different messages with the same hash value.

Many different hash function have been developed in history. Well known is for example the 'Cyclic Redundancy Check' (CRC) which is mainly used for error detection in data streams or files. It is not recommended to use it to assure data integrity, since it is possible to create messages for a given hash values very efficiently. To ensure data integrity hash functions must provide a sufficient preimage and collision resistance. Hash functions that provide this properties are termed cryptographic hash functions. Popular examples of cryptographic hash functions are MD4 and MD5 [Riv92] which calculate 128-bit hash values and SHA-1 [EJ01] with a 160-bit hash value. Since these hash functions have been successfully attacked ([WYY05], [WY05]), stronger algorithms are recommended. Today, the SHA-2 functions [FIP02] with hash lengths up to 512 bit and the WHIRLPOOL hash function [RB01] are considered as safe.

Signatures

The digital signature is a method for authenticating digital information. The intention is to obtain a digital equivalent to the classic written signature on paper. It should verify that the signee has read and acknowledged the content. In WSNs the content may be data packets or larger documents such as code images.

In the previous section hash functions were already described, which assure that a received message has not been changed. Encryption of the hash is used to authenticate the issuer. Generally, there are two possible ways of performing this authentication: by symmetric or asymmetric cryptography. The shared key approach is connected with disadvantages. First, there is the problem of distributing the key. In addition the approach does not prove the originality to a third party. However they can be executed with low overhead. This is why symmetric signatures are used as *message authentication codes* also in WSNs, while the CBC-MAC [PR00] and HMAC [Kra97] are the most prominent representatives. They use standard block ciphers as AES cryptographic basis.

Public key based signatures are favorable because everyone can verify the origin since only the issuer has the encryption key. Additionally, distributing the public key is less security critical because it is public anyway. The problem with PKC-based signatures is the computation overhead. That is why in WSNs full sized signature approaches such as ECDSA (Elliptic Curve Digital Signatures Algorithm) are only used if the authentication is highly important.

4.4.2 Applied Countermeasures

Cryptography alone does not solve all security problems in the WSNs. This is why in this section a brief overview on applied countermeasures on all layers of the system will be provided.

Hardware

Tamper Resistant Sensor Nodes have the goal to protect the hardware against physical attacks. The aim is to prevent unauthorized accessing memories and key material on the

sensor nodes even in case they are hijacked by an attacker.

Basically two ways of tamper protection are considerable: active tamper protection and passive tamper protection.

Passive tamper protection tries to protect the nodes by impeding the access to the nodes by specific sensor packaging, housings, or fences. The advantage of passive protection is that no additional energy needed and no special node hardware. A variation of the passive tamper protection is the tamper detection by seals or surveillance. It does prevent tampering but allows to detect nodes that have been tampered.

Active tamper protection increases the resistance of the sensor node hardware by implementing specific circuits that protect the information on the chips against unauthorized access. This includes protection against direct memory access, side-channel attacks, and fault attacks. Tamper resistant devices still are technologically extremely expensive and thus not suitable for the application in WSNs. Projects such as TAMPRES [TAM11] have the goal to find suitable solutions, while generally a trade-off between security, costs and energy consumption has to be considered.

Rich Uncles are sensor nodes in the network with larger computation power. Such nodes can be better protected and work as trust center or gateway node for their region in the network. They are also suited to execute resource-intensive tasks and impersonate important security roles. Since the quantity of such powerful nodes in the network is rather low, their higher costs is acceptable. In most networks the base station is a *rich uncle*.

Network and Services

Prevention of Jamming Attacks by specific radio chipsets and protocols is a technical sound approach that typically needs additional specific hardware. The protection is achieved either by using wide-band communication to reduce the impact of narrow band jammers. First test implementations of ultra wide band (UWB) protocols for WSNs are discussed for example in [OKK09]. Nevertheless the technology still is relatively expensive and not perfectly matured.

Another approach to resist jamming attacks is changing the interfered frequencies if disturbances are detected. Such frequency hopping is part of the Bluetooth radio protocol. Software controlled frequency hopping as part of the medium access control is for example discussed in [ZHY⁺06].

Secure Routing: As described for the attacks in the previous section, compromising the network routing is a common and powerful attack in WSNs. To cope with these attacks several secure routing approaches have been discussed and proposed [KW03], [ABV06]. Typical countermeasures contain symmetric ciphers and light-weight hashes on link level to provide authentication to the neighbors. They work against external attackers while internal attacks cannot be detected. Multi-path routing is a probabilistic solution to reduce the risk of single point of failures, while the redundancy requires additional energy in the network. Thus the question for perfectly secure routing protocols in WSNs is still an open research issue.

Transport protocols for WSNs such as nanoTCP [JMR⁺08] and DTSN [MGN07] inherently provide a certain degree of reliability by session control, flow control, and data stream management. The transport protocols rely on secure network protocols and usually do not need explicit protection.

Key Establishment and Authentication Protocols received significant research attention in recent years. The actual problem is straightforward as it concerns only the question how do the sensor nodes get the keys they need to encrypt, decrypt or sign their messages. The established shared keys are the trust basis for the further operation. That is why the key establishment is basis for authentication, but also needs some authentication. Basically three general key establishment approaches have been discussed: Pre-deployment, ad-hoc based on asymmetric cryptography, and with symmetric keys and a trusted authority. *Pre-deployment* technically is the easiest solution since the keys are distributed before the sensor nodes are deployed. However it needs set up overhead. The overhead often is reduced by giving all nodes the same keys which turns the network at risk as soon as one node could be compromised. Changing the keys after such an incident then is connected with significant manual work. The strength of *key agreement based on symmetric keys* depends on the implemented protocols and ciphers. The trusted authority allows key renewal and revocation. However, it is also a vulnerable point of the network. *Key agreement based on PKC* is the most mature approach. Certificate signatures signed offline by a trusted authority can be verified online to realize the authenticated key agreement. The disadvantage is the computation overhead for the validation of the signatures and the agreement of the keys. A hybrid of public key and symmetric key protocols as proposed in [CKM00] seems to be a reasonable answer to the issue. The space of proposed solutions for key agreement schemes is huge and goes far beyond this brief taxonomy. Notably, one of the first tool-supported configuration tools for security in WSNs tackles key management systems [AR06] (see also Section 4.6.2).

4.5 Secure In-Network Aggregation

4.5.1 In-Network Aggregation

Reducing the total required energy in a wireless sensor network is an outstanding goal. Beside the power required for the computation on the nodes, the power needed for sending and receiving the data packets in the network is a significant factor. Sending one bit requires the same amount of energy as executing 50 to 150 instructions on sensor nodes [PLP06]. Thus, omitting as much network traffic as possible is a substantial task in the area of designing WSN applications.

A well known approach that reduces the energy required for packet transmission in the sensor-to-sink scenario is the in-network aggregation (INA). In many scenarios the sink does not need the exact values for all sensors but a derivative such as sum, average, or deviation. The idea of the INA is to aggregate the data required for the determination of the derivatives as close to the source as possible, instead of transmitting all sensed values through the entire network.

Figure 4.4 shows the data gathering sensor nodes S_1, S_2, S_3 sending their data to the intermediate aggregator node $A_{2,2}$ which aggregates the three values and sends one

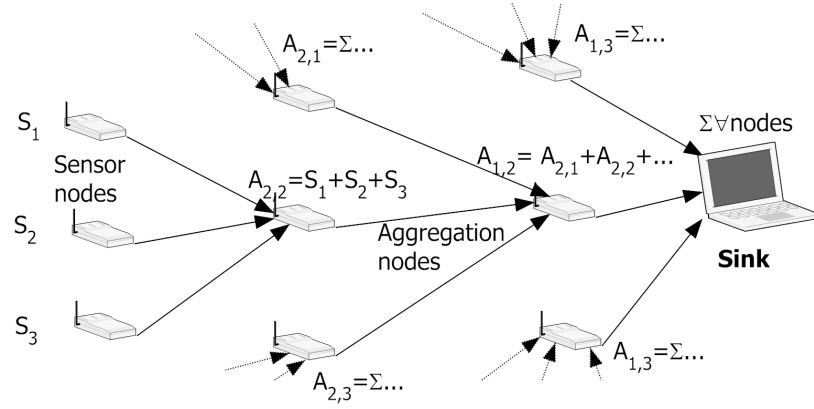


Figure 4.4: Principle of in-network aggregation: The sensor nodes $S_1 \dots S_n$ send the value to the sink. On the way to the sink the values are aggregated by aggregator nodes A_* .

instead of forwarding three packets. The next aggregator node ($A_{1,2}$) combines further values that are finally sent to the sink.

A serious issue connected with the INA is the security of the data [PWC10]. Considered that the data is transmitted encrypted, there is the problem that all aggregation nodes, i.e. the sensor nodes that perform the actual aggregation in the network, must have access to the decrypted values. Beside the lack of end-to-end (ETE) security, such a hop-by-hop (HBH) encryption as it is for example part of TinySec [KSW04] has the drawback that the data must be decrypted and re-encrypted on every aggregation node. An approach that promises the combination of ETE-security and INA is the concealed data aggregation (CDA).

4.5.2 Hop-by-Hop Encryption

Hop-by-hop encrypted aggregation encrypts the data sent over the wireless link so that potential eavesdroppers cannot extract the actual data. Each intermediate node decrypts the data so that it can be processed, i.e. aggregated. Finally the node encrypts the result and sends the encrypted data to the next hop. The HBH approach has the drawback that the data must be decrypted and re-encrypted on every aggregation node which is clearly unfavorable from the energy consumption point of view. Another critical disadvantage is the missing feature of end-to-end protection. It poses a risk if intermediate nodes can be accessed or compromised, which is a sensible attack scenario for WSNs.

However, using standard cryptographic libraries such as tinySec [KSW04] the HBH-INA approach easily enables secrecy over the wireless link while reducing the transmitted number of packets.

4.5.3 Concealed Data Aggregation

Concealed Data Aggregation (CDA) -the term was coined in [GWS05]- is an improved version of the INA, which, in contrast to HBH, ensures the ETE-privacy, i.e. the encrypted values do not need to be decrypted for the aggregation. Instead, the aggregation is performed on encrypted values and solely the sink can decrypt the result.

The background of CDA are privacy homomorphism functions that have the property: $enc(a + b) = enc(a) \oplus enc(b)$, i.e. an operation on two encrypted values has the same result as the encryption of the sum of the two unencrypted values. Assumed that such a secure INA exists, it has significant benefits compared to HBH and classic ETE encryption.

Network traffic: Since the data is aggregated in the network, the network efficiency is better than ETE without aggregation. In [CMT05] network configurations are described that reduce the network traffic by 85% due to CDA. In order to improve the network efficiency the packet size must be considered. Large encrypted packets could negate the positive network effect.

Network flexibility increases because with CDA aggregator nodes may change or can be replaced without touching keys.

Computation effort: Compared to the HBH-aggregation, the computation effort can be assumed as smaller, because there is no need for decryption and encryption on the aggregation nodes. Indeed, this is only true if the cryptographic algorithms that allow the concealed aggregation do not require too many additional computations.

Security A significant benefit is the improved security in comparison to the HBH-aggregation. Since the values are not decrypted on every aggregation node, there are less points where an adversary could catch the unencrypted values.

The fundamental basis for CDA are cryptographic methods that provide the privacy homomorphism (PH) property. An encryption algorithm $E()$ is homomorphic, if for given $E(x)$ and $E(y)$ one can obtain $E(x*y)$ without decrypting x, y for some operation $*$. The concept was introduced by Rivest et. al [RAD78] in 1978. The two most common variations of PHs are the additive PH and the multiplicative PH. The latter provides the property $E(x \times y) = E(x) \otimes E(y)$. Well known examples of multiplicative PHs are RSA and the discrete logarithm ElGamal. But since the multiplicative aggregation does not have apparent applications in the field of INA on WSNs, we restrict our search to an efficient PH to the additive PHs with the property $E(x + y) = E(x) \oplus E(y)$.

4.5.4 State of the Art

Several PH algorithms have been proposed in the literature. A large subgroup is the family of high degree residuosity class based cryptographic algorithms, for example from Paillier [Pai99], Naccache-Stern[NS98], and Okamoto-Uchiyama [OU98]. Even though these public key schemes provide the additive PH, we do not analyze them in this work, because they need very long keys that imply large messages and computation effort that does not suit the WSN scenario. Embodiments of these schemes, designed as public-key elliptic curve discrete-log encryption scheme, were introduced in [Pai00]. But according to [MGW06] this interesting approach still requires too much bandwidth and computation efforts.

Below we discuss three approaches that fit to the WSN world.

Domingo-Ferrer (DF)

In [DF02] Domingo-Ferrer introduced a symmetric PH (DF) scheme that also has been proposed as efficient PH cryptographic system for WSNs in [GWS05].

Domingo-Ferrer (2002) Algorithm [DF02]	
Parameter:	<i>public key: integer $d \geq 2$, large integer M secret key: g that divides M; r so that r^{-1} exists in \mathbb{Z}_M</i>
Encryption:	split m into d parts $m_1..m_d$ that $\sum_{i=1}^d (m_i) \bmod g = m$ $C = [c_1, \dots, c_d] = [m_1 r \bmod M, m_2 r^2 \bmod M, \dots, m_d r^d \bmod M]$
Decryption:	$m = (c_1 r^{-1} + c_2 r^{-2} + \dots + c_d r^{-d}) \bmod g$
Aggregation:	Scalar addition modulo M $C_{12} = C_1 + C_2 = [(c_{11} + c_{21}) \bmod M, \dots, (c_{1d} + c_{2d}) \bmod M]$

It has both the additive and the multiplicative PH property. It is a symmetric algorithm that requires the same secret key for encryption and decryption. The aggregation is performed with a key that can be publicly known. It is required that the same secret key is applied on every node in the network. The message size is $d \cdot n$ bit, so that for very secure parameter combinations ($d > 100$) the messages become very big [Wag03].

CMT - a Key stream based PH

A key stream based PH was proposed in [CMT05] by Castelluccia, Mykletun, and Tsudik. We denote it CMT, corresponding to the authors initials. It applies individual keys on every node and promises provable security with small ciphertext sizes.

The idea is to perform a modular addition of a classic stream cipher and with the sensed data. Every sensor uses a different pseudo random stream, for example RC4 or AES in CBC modus. For encryption, the plaintext is simply added to the current key of the stream modulo the length of the key space M . The sink has to subtract the corresponding key stream to obtain the plaintext again.

Castelluccia, Mykletun, Tsudik (CMT) Algorithm [CMT05]	
Parameter:	<i>select large integer M</i>
Encryption:	<i>Message $m \in [0, M - 1]$, randomly generated keystream $k \in [0, M - 1]$ $c = (m + k) \bmod M$</i>
Decryption:	$m = (c - k) \bmod M$
Aggregation:	$c_{12} = (c_1 + c_2) \bmod M$

Since the message size is determined by M , and only one modular addition is required for encryption and aggregation, CMT is very well suited for the application on WSNs. A problem is the decryption, which requires exactly the same key stream as at each sensor node. It is not only a potential computation problem, but the sink that decrypts the aggregated values must also know which sensor data is part of the aggregate. It has to subtract exactly the same key streams that have been used for the aggregation. The knowledge of these node-IDs is substantial for the algorithm which may result in a data overhead [PLP07]

Elliptic Curve ElGamal

In contrast to the both PHs presented so far, the elliptic curve ElGamal (ECEG) based PH is an asymmetric cryptographic approach. The benefit of this PH is that the encryption key may be publicly known, since it is the public key of the sink. As the name suggests the ECEG PH is based on the well investigated ECEG cryptographic algorithm.

ECEG PH Algorithm [MGW06]	
Parameter:	<i>private key integer x</i> <i>public key (G, H), G and H are points on EC, $H=xG$</i>
Encryption:	$C = [c_1, c_2] = [kG, kH + mG] = \text{tuple of EC points}$
Decryption:	$mG = (kH + mG) - x(kG)$ <i>demap: $mG \rightarrow m$</i>
Aggregation:	scalar EC-point addition $C12 = C1 + C2 = [(c1_1 + c2_1), (c1_2 + c2_2)]$

ECEG introduces a serious issue. The message text must be mapped on the EC. In [Adl00] and [MGW06] an approach has been proposed that multiplies the message text with the generator of the EC. It is also our preferred mapping algorithm, even though it causes some problems. The decryption leads again to the mapped point mG , but it is not trivial to compute m out of mG . Since it is the fundamental property of ECC that the point multiplication is not efficiently invertible, the only solution is a brute force computation that relies on a limited domain of the mapping. In most cases this approach is very reasonable. Please notice that without a valid key it is not even feasible to compute the point mG , so that the security is not interfered by the de/mapping.

Combination of CDA algorithms

The previous considerations indicate that each of the three described CDA algorithms has its individual beneficial properties. As we will evaluate in the next section none of the presented CDA algorithms provides all desirable protection features. An answer to this issue is the combination of CDA algorithms to a more secure cryptographic system. Two PH algorithms can be combined by cascaded encryptions:

$$E_2(E_1(a)) \oplus E_2(E_1(b)) = E_2(E_1(a + b))$$

Such a chain has some requirements on the encryption algorithms: both encryption schemes must be additive PH, and the ranges of results of inner encryption E_1 must fit to the domain of E_2 . Usually E_1 is a function: $E_1 : \mathbb{Z}_n \rightarrow \mathbb{Z}_n$, while $E_2 : \mathbb{Z}_n \rightarrow \mathbb{C}$, with \mathbb{C} as domain of the ciphertext. From the described PH schemes CMT is very suitable as E_1 function. Both, ECEG and DF can use CMTs results. As an example we demonstrate the combination of CMT and DF. In the previous section we described that CMTs only security weakness is the malleability, i.e. one can modify the content of a ciphertext without knowing the plaintext. Exactly that property is a strength of DF. Without the knowledge of the secret key one cannot modify the content of a single packet. Hence, this approach should result in the most secure CDA.

The advantage of this combination is that the aggregation requires exactly the same effort as the standalone DF. Since most security concerns are already covered by CMT,

CMT + DF algorithm	
Parameter:	public key: large integer M , $d \geq 2$ secret key: g that divides M ; r so that r^{-1} exists in \mathbb{Z}_M
Encryption:	randomly generated keystream $k \in [0, M - 1]$ $e_1 = (k + m) \bmod M$ split e_1 into d parts $m_1..m_d$ that $\sum_{i=1}^d (m_i) \bmod g = e_1$ $C = [c_1, \dots, c_d] = [m_1 r \bmod M, m_2 r^2 \bmod M, \dots, m_d r^d \bmod M]$
Aggregation:	scalar addition modulo M (like DF)
Decryption:	$d_1 = (c_1 r^{-1} + \dots + c_d r^{-d}) \bmod g$ $m = (d_1 - k) \bmod M$ where k is the sum of aggregated key streams

the DF parameters, especially d , do not need to be too big. But, with both encryption methods, we get the technical problems of both approaches. With $d > 1$ the encrypted message size increases and there is still the id issue to indicate not responding nodes.

4.5.5 Security Evaluation

The most important property of a cryptographic scheme is its security. In this section the security is evaluated by means of resistance of the CDA approaches against specific attacks. For this purpose we address the major attack scenarios for CDA schemes for WSNs and evaluate to what extent the considered cryptographic schemes have the desired resilience. The attacks described here are more specific than the general attacks described in Section 4.3

Passive attacks

These are the most common attacks. The primary security goal is that such an adversary is not able to gain any information by simple eavesdropping. Even though the primary goal is the concealing of the data, several individual attack scenarios are possible.

Ciphertext analysis: A very common attack is the analysis of encrypted packets. A secure cryptographic system must ensure that an adversary is not able to decide whether a encrypted packet corresponds to a specific plaintext or not. Applied properly, the three described CDA schemes are secure against this kind of attack. If the secret keys are hidden, there is no way to obtain information out of the encrypted data.

Known plaintext attack: In this kind of attack the adversary tries to determine the secret information with the additional knowledge of the plaintext. In a WSN scenario such an attack is likely since an attacker can obtain plaintext, e.g. by own sensor, physically accessing the deployed sensor, or manipulating the sensor readings. Studies [Wag03] show that in particular the Domingo-Ferrer PH is very vulnerable to known plaintext attacks. The both other schemes are immune to this kind of attack.

Active attacks

This kind of attack assumes that the adversary is able to interfere the communication, i.e. to catch, destroy, modify, and send packets. As we will see, such an attack is the most dangerous threat against the CDA approaches for WSNs.

Replay attacks: As introduced earlier, replay attacks emit a previously sent valid packet a second time with the intention that the packet is recognized as valid. In a movement detection scenario, a trespasser can keep sending the 'no movement' signal while he is moving in the protected area. From the discussed CDA approaches only the CMT resists this kind of attack, because it applies a new key for each message.

Malleability: The idea of this very dangerous attack is to change the content of a valid encrypted packet without leaving marks. For CMT and ECEG it is possible to alter the content of an encrypted packet, without knowing the plaintext. An adversary can alter the encrypted values of CMT or ECEG by adding natural numbers or multiples of the generator point, respectively. Consider an adversary wants to increase the value of an encrypted ECEG value by 10. Since the generator G is part of the public key, he could add $10 \cdot G$ to the original encrypted value:

$$[kG, kH + mG] + [0, 10G] = [kG, kH + (m + 10)G]$$

Unauthorized Aggregation: A threat of maliciously aggregated proper ciphertexts to a new valid but bad ciphertext. Similar to malleability. In case an adversary knows one ciphertext, he can use this packet as summand to add it, or any multiple of it, to any ciphertext without knowing its plaintext. DF and ECEG are vulnerable to this attack. CMT has a protection, because it uses a unique key for each message and expects exactly that key as part of the aggregate.

Forge packets: Actually, in case of ECEG there is no reason to alter existing encrypted data, since the pure application of ECEG with public keys allows everyone to create own ciphertexts with desired content. DF and CMT are resistant to this kind of weakness because the keys required for the encryption process are kept hidden.

Physical attacks

This group of threats includes all kinds of physical attacks against the node. Obviously, one could disable a node, but this would not implicitly be a threat against the security. A serious threat is the capturing of nodes. The access to the flash and memory may reveal key information that can compromise the entire network. In particular, symmetric encryption schemes that use the same key on every node are vulnerable. An example for such a scheme is the DF. Public key approaches like ECEG, as well as algorithms that can use different keys for each node like CMT, are resistant to the node capture attack.

4.5.6 Comparison of CDA approaches

Table 4.3 shows a brief evaluation of the described CDA encryption transformations regarding the set of properties and the described attack scenarios. Indeed, such an overview

Table 4.3: Comparison of CDA algorithms

	DF	CMT	ECEG	hCDA
Ciphertext size	—	+	<i>o</i>	—
Comp. effort encryption	<i>o</i>	+	—	<i>o</i>
Comp. effort decryption	<i>o</i>	—	—	—
Comp. effort aggregation	<i>o</i>	++	—	<i>o</i>
Resistance regarding				
ciphertext only	++	++	++	++
Chosen plaintext attacks	—	+	++	++
Replay attacks	—	++	—	++
Malleability	+	—	—	+
Malicious Aggregation	—	+	—	+
Forged packets	++	+	—	++
Captured sensors	—	+	++	+

cannot deliver a satisfying assessment for every situation and parameter combination. For example the ciphertext size of CMT is considered as positive. But the positive assessment is not justified anymore in case where many not responding IDs must be transmitted.

Another controversial point is the computation effort for EC-EG. Because ECC software implementations are known to be quite slow it is assessed with '-'. However, executed on hardware accelerators ECC is very fast. Moreover, in this case the power consumed during the computation is even smaller than it is required for the transmission of the encrypted data packet. Thus, if hardware accelerators are applied the computation costs for ECC can be neglected [PLP08].

Nevertheless, as result of our evaluation, CMT as the PH approach with the least computation efforts is also the most secure stand alone PH approach. Its only real weakness is the malleability. In combination with DF as hybrid CMT/DF even this weakness is solved. CMT/DF has its advantages in the security category, but on cost of efficiency. The message size is bigger and the computation efforts are higher.

However, in many application scenarios not all properties must be perfectly fulfilled. In case only a simple encryption is wanted and an active attack, which is connected with considerable expenses, is not a probable threat, all three algorithms are reasonable. In such a case side constraints could favor one algorithm or another. For example, if ECC is already part of the WSN, maybe for the key exchange protocol, ECEG is very reasonable. If malleability protection is important, DF should be selected. Of course, the security issues of every algorithm must always be kept in mind.

4.6 Holistic Approaches to configure Security in WSNs

The protection mechanisms discussed in the previous section typically tackle one specific sort of protection. These means have to be assembled and integrated in the actual application which usually requires a variety of protection means. In the description of the protocols in the previous section we could also see that many similar concepts are used for different mechanisms. That is why reusing concepts for protection is a reasonable approach to reduce the overhead for security in the system. This indeed needs a holistic view on the configuration of security.

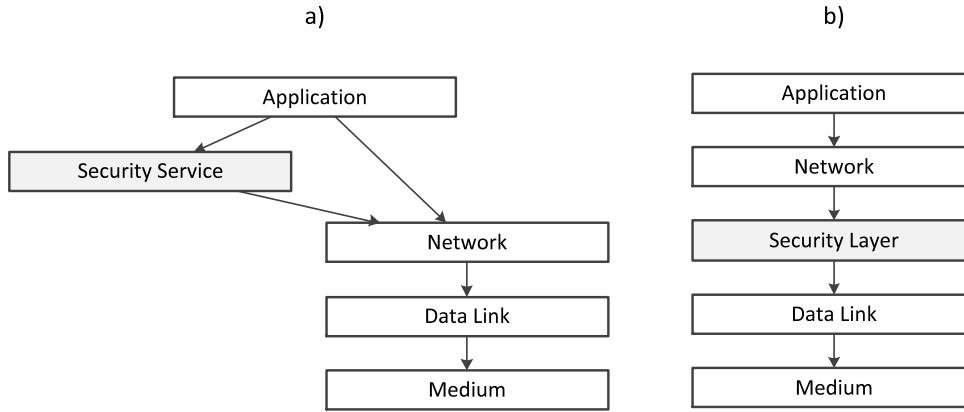


Figure 4.5: Different architectures of integrating security toolboxes: a) as security service that is explicitly used by the application, b) as security layer in the network stack.

That is why several toolboxes or security layers have emerged. They provide a preselected and preconfigured set of security mechanisms. That way they promise a certain degree of well-balanced security assurance that is ready to use. While the goal is the same there exist differences between toolboxes and security layers. Security layers usually are transparently integrated in the network stack, as illustrated in Figure 4.5 b). In contrast security toolboxes are explicit services that can be accessed by the application.

4.6.1 Security Service Toolboxes

LEAP [ZSJ03] (Localized Encryption and Authentication Protocol) and LEAP+ [ZSJ06] use multiple symmetric keys to communicate with different peers in the network. That means that communication with base station is performed with other keys than communication to other nodes in the network or in the neighborhood. This is supposed to reduce the impact of compromised nodes in the network. Important task of LEAP is the light-weight key establishment that also includes a light-weight authentication mechanism. LEAP solves a set of possible attacks in WSNs including wormhole attacks, sybil attacks and cloned or added nodes. However it relies on a short neighborhood discovery phase in which the network is utterly vulnerable. LEAP was implemented as service block for TinyOS. It uses the RC5 block cipher to encrypt messages, to compute the MAC and to generate pseudo random numbers. This code reuse saves memory and is a good example for efficient security engineering.

SPINS [PST⁺02] probably is the most prominent security tool suite for WSNs. It actually contains two services: SNEP (Sensor Network Encryption Protocol) which combines encryption and a MAC to provide confidentiality and integrity, and μ TESLA-a broadcast authentication protocol. Both protocols are building blocks in TinyOS and can be used by an application over the provided interfaces. SNEP allows applying different ciphers to cope with the security-performance trade-off. Thus it is a promising building block which still needs some expertise to use it.

4.6.2 Transparent Security Layers

A wide-spread approach to handle security is to add a security layer before sending data to the network. For Internet applications the Secure Socket Layer (SSL) is a well-known implementation of this notion. For WSNs several similar approaches have been presented.

[SSD07] presented a transparent security layer for WSNs. It is added to the link layer of the network protocol. The service is completely transparent to the user. It is just required to enable a flag for integrity, confidentiality, or replay protection if required. The ciphers and message integrity mechanisms are always embedded in the layer and are used or not - based on the runtime requirements of the user. TinySec [KSW04], SenSec [LWWB05], and SecureSense [XG03] are other link-layer runtime security composition frameworks for TinyOS. MiniSec [LMPG07] is a similar framework working on the network layer. The protocols and services are pre-selected by the designer of the framework and applied for each packet passing the network stack. Without significant rewriting of the frameworks it is not possible to add or remove services. Another problem is that most proposed solutions are optimized for a specific platforms (e.g. TinySec for Mica nodes, MiniSec for Telos nodes). This helps improving benchmark but does not foster reusability and comparability of the approaches.

It is notable that all these transparent security layers focus on hop-by-hop security. This is necessary to allow an application in-network aggregation but makes the implementation vulnerable against compromised nodes along the network path since no end-to-end security is provided.

4.6.3 Composable Security Approaches

Both security service toolboxes and transparent security layers follow the notion of readily configured assemblies of security means. The results are well-balanced and sufficiently analyzed security compositions that simply can be reused. However, all approaches make certain compromises to trade off security requirements and the highly constrained resources of WSNs. The user has to know and accept the compromises which means first, a certain experience and knowledge about security must be present on user side, and second, there is no means to adapt the security properties to fit specific requirements of the application. To cope with this issue, tool-supported security configuration approaches have been proposed.

KMS Guidelines

A tool that could be an example for a small solution library, entirely focuses on 'Security Through Usability' and has been published by [AR06].

It is a security mechanism selection toolkit to configure key management schemes (KMS) for WSNs. The key distribution and organization is a security critical operation which heavily depends on the structure and the environment of the network.

The authors categorized several key management schemes for sensor network applications. The work bases on the observation that every KMS gives "some priority to certain objectives, optimizing certain properties while neglecting others" [AR06].

A user starts the selection process by providing objectives that are important for the application scenario. After entering main and secondary properties, e.g. small memory, connectivity, scalability, resilience, the tool delivers a list of key distribution schemes

that fulfill the requirements. Additionally, the tool lists specific advantages and disadvantages of the algorithms, so that competent users have further information supporting the selection process.

The properties discussed in the work are memory footprint, security, network resilience, connectivity, scalability, communication overhead, energy, and localization.

However, KMS is no actual implementation support. It recommends approaches but leaves the developer alone with the integration.

Cionca Security Configuration Approach

Cionca and Neue [NCB10] proposed a security-related configuration tool [CND09] with usability as main concern. In a graphical user interface it allows the user to describe relevant parameters of the application. From these inputs the tool presents a set of security protocols that suit the requirements. Currently, they support four security primitives: key predistribution, key establishment, confidentiality and authentication. For each primitive several popular protocol alternatives with different security strength and footprint were selected and analyzed. The analysis concerns environment, topologies, communication patterns, security strength and power consumption, among others. The results are stored in a sort of database that can be accessed later to determine whether a protocol satisfies a specific property.

To release users from entering detailed security specification, application types are used to derive the environment security levels. Military applications have high security requirements, medical applications need high data security in a public environment, structural monitoring applications have low requirements while home applications are assumed to run in trusted environments.

The Cionca approach presented an envisioned design flow with final code generation and deployment phase. However it has not been implemented yet.

FABRIC

FABRIC [Pfi07] is a data-centric middleware synthesis tool for WSNs, already introduced in Section 3.1.2. A user defines needed data types and annotations. Based on this requirement the middleware compiler composes a service stack that provides a well-defined interface to the users' application.

FABRIC also respects security and reliability as part of the middleware. Security is an alias for concealment and is integrated as separate service layer on top of the transport layer. Aspects define the general security requirement (public or confidential) while the priorities define the required/provided strength. For instance, if a module provides high confidentiality, it has the aspect confidentiality with the priority high.

As result of the static structure of the middleware also a module of the security layer has to be selected even when there are no security requirements. Reliability is integrated accordingly as aspect of the transport layer.

In [RPF08] the authors introduced an extension of the security assessment process in FABRIC. The motivation of that extension is similar to our configKIT idea: A user or application designer should be supported by a tool which evaluates and proposes security configurations based on user-given application properties. The framework also follows the notion of integrating security mechanisms implemented by security experts. Considerable mechanisms are evaluated in the context of the application scenario. The results of that

evaluation are presented to the user who eventually decides which mechanism should be selected.

Inputs of the selection process are data types, application properties and attacker profiles. After an analysis which applies attack trees (see also Sec. 4.7.2) the considered attacks are evaluated concerning the efforts for the attacker and the risk for the application. For a subset of reasonable attacks the framework analyzes potential defenses and generates a list of suitable security setups. The setups are evaluated in context of the application and residual risk tables are presented to the user. Those tables contain the remaining risk for specific security-related properties of the application. They also contain textual explanations of the risks.

Unfortunately, the technical description of the automatic selection and evaluation process in [RPF08] and the related publications is not precise enough to understand or reproduce the results.

4.7 Security Engineering

Today's security engineering in the domain of WSNs is characterized by an analysis of application-specific threats and attacker methods which concludes in a puzzle-like integration process trying to combine various specific countermeasures and protocols. [PS08] provides an extensive overview on this matter. However, security and its engineering for software and hardware has been on the agenda of researchers and practitioners for a long time before WSNs emerged. In this section several more general methodologies from the domain of the security engineering are discussed.

4.7.1 Formal Security Models

Security models are an abstraction to represent the complex matter of system security. McLean [McL94] defined the term security model as “any formal statement of a system’s confidentiality, availability, or integrity requirements”. The term security model is often confused with security policy models. The most common application of security models is related to the modeling of access control schemes. Most famous is the Bell-LaPadula model [BL73] which is the basis for many access control systems. It labels objects and subjects with restrictions and clearances respectively. The security labels are security levels that range from unclassified (lowest) to top secret (highest). A read access to an object is only permitted if the subject has the same or higher clearance than the classification of the object. Subjects can only write objects that have the same or higher levels. The success of the model is caused by its simplicity, effectiveness, and provable security. If implemented correctly it guarantees that no sensitive information can be accessed by subjects without corresponding clearance.

Security models typically do not describe the needed mechanisms to implement these requirements. They rather base on the notion that the fundamental mechanisms have to be implemented as sort of security platform. Only then the security properties provided by the models can be achieved.

The techniques of verifying implemented models and their theoretical models is called model-checking [Cla97]. Several rather formal approaches [ABV06, PW01] follow the notion of defining an ideal (secure) channel and model a real system with the goal it can be proved to be indistinguishable to the ideal channel. Since no protection primitive is

perfectly secure, [ABV06] qualified this absolute indistinguishability to a statistical security which says that two models are indistinguishable if the probability that the primitive can be broken is negligible.

In [EWKL07] the authors discuss the integration of security aspects into a formal method based development of networked embedded systems. The focus of the security analysis language (SAL) is merely on information flow between networked entities. By that, it might be a way to model security requirements of applications and to verify whether or not the correct security modules were selected.

AVISPA [ABB⁺05] is a model-checking tool that allows the formal analysis of security protocols. Even though it is supported by tools, modeling is still complex and needs significant manual work. AVISPA-based analysis of WSN the protocols TinySEC and SNEP are presented in [TCC⁺07b] and [TCC07a], respectively. Anyway, the protocol analyzers typically only work on the Dolev Yao attacker model, i.e. only consider attackers on the radio link which makes them interesting for protocol design but do not support our view of WSN security.

4.7.2 Top-Down Security Engineering

More practical security engineering methodologies have been proposed in the domain of software engineering. There the typical top-down system design flow is supported with very specific security model extensions:

UMLsec

UMLsec [Jür02] is a security extension to the Unified Modeling Language (UML). It supports developers in the design process to define and control security-relevant information within the diagrams of a system specification. It applies the concept of stereotypes to define properties and environment of system, use cases, functions and subsystems. Thus, UMLsec allows to model properties of functions and subsystems from bottom up. Concurrently the system use cases and their security requirements are modeled from top down. Basically, UML checks that constraints in both directions hold. That is, to ensure that subsystem and functions are modeled in a way that is promised in the higher layers - and in the other direction to ensure that no sensitive data or function can be accessed in an insufficiently protected environment. So in first place UMLsec helps developers to avoid design errors, for example by forgetting the correct security properties of a function. Eventually, core of the security matching is ensuring that data with specific security criteria are not exchanged over channels that permit specific threats.

The three supported data security criteria are *secrecy*, which requires that no threat allows unauthorized reading data from the channel, *integrity*, which requires that no threat allows unauthorized changing data on the channel, and a *high security* requirement that neither unauthorized reading, changing, or deleting data on the channel.

The attacks that can be performed for a given environment are expressed by a threat function. The result of the function $Threat_i(e)$ for an assumed attacker type i and the environment e is the set of possible attacks. In [Jür02] the sets for the default attacker

are:

$$\begin{aligned} Threat_{default}(Internet) &= delete, read, insert \\ Threat_{default}(encrypted) &= delete \\ Threat_{default}(LAN) &= empty \end{aligned}$$

Thus, the default attacker represents an outside attacker with modest capabilities. Extending UMLsec to satisfy the needs of WSNs is generally considerable. For instance new channels (e.g. side channels) can be added to respect the fact that WSN nodes are subject to physical attacks.

SecureUML

SecureUML [LBD02] is another security-related extension of UML. The aim of SecureUML is to enable the development of secure systems even for developers without strong security background. Its focus is put on access control and authorization. It is possible to model user roles, and the relations of the user to data objects. By this it is feasible to express access control policies with UML precisely. Thereby, the role-based access control ignores the technical details. Since SecureUML does not provide answers in this matter, currently UMLsec appears to be a more suitable extension of UML. Anyway, a combined application of SecureUML and UMLsec is feasible.

SREP

SecureUML is used in the security requirements engineering process (SREP) proposed in [MFMP07]. The standard-based process deals with security requirements early in the software development flow based on the reuse of security requirements which are stored in requirement repositories. The process consists of nine process steps distributed over seven roles. While the process at first glance appears somewhat overloaded the process steps are reasonable. The process includes the identification of security objectives, dependencies which are used to derive potential threats. The latter are the basis for the elicitation of actual security requirements. Finally, model elements such as threats and actual requirements are stored in a repository to foster reusability. The process stops after the systematic engineering of the requirements but does not respect the system integration.

Attack Trees

Schneier [Sch99] introduced attack trees to describe security of systems and subsystems. They represent the attacks and show systematically the different ways a system can be attacked in an top-down approach. The root node of the tree is the goal of the attack. Leaf nodes represent the attacks. Intermediate nodes are refinements of the goal to the actual attacks which cannot be further refined.

For example consider the small attack tree shown as Figure 4.6. On the root it has the attacker goal 'Read data packet'. In the first branch it shows this can be realized by eavesdropping from the channel or by extracting the data from a node. The latter requires access to the node and the ability to read the content.

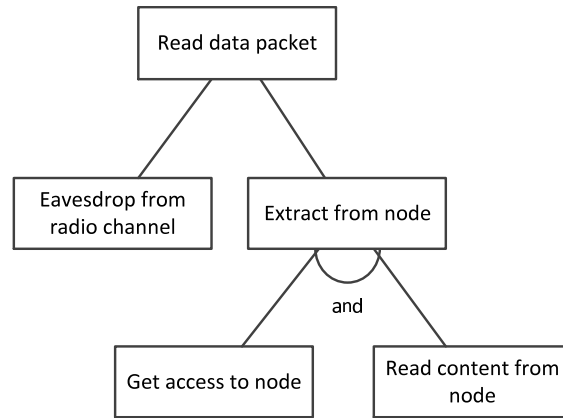


Figure 4.6: Attack Tree example: The attacker goal “Read data packet” can be realized either by eavesdropping from the radio or by extracting the data from the node. The latter needs access to the node and the ability to read from the node.

For analysis purposes, leaves, i.e. the attacks, can be assigned with attributes, such as the possibility, the required costs, time, or the need for special resources. The bottom-up aggregation of the properties in the tree then reveals a total possibility (or minimum costs, time or resources) for the attack goal to be achieved.

In the example, considered the access to the node costs an attacker \$200 and eavesdropping costs \$100, within the tree we can conclude that the attack goal to read the packet costs the attacker \$100. In the moment eavesdropping will be prevented, for example by good cryptography, the efforts for the attacker rise to \$200.

Since every node can have several attributes, the analysis helps to discover for example the cheapest low-risk attack or the most likely non-invasive attack, or the cheapest attack with the highest probability of success. A disadvantage of the technique is that the attack paths have to be known in order to assess the security implications. Novel attacks usually are not considered by the tree.

Anyway, attack trees are a valuable tool to assess the effect of specific attacks. That is why in Section 6.3.3 an application of attack trees for the assessment of security in WSNs is described.

Attack Dependency Graphs

Related to attack trees are exploit dependency graphs [NJOJ05]. Attack dependency graphs are directed graphs of the dependencies among possible attacks, weaknesses and preconditions and postconditions. Final postconditions are the attacker goals. The attacker goal can only be achieved if there is a path from the goal to an attack while all attacker requirements along the path are possible to be fulfilled. For the protection it means that to prevent an attacker goal it is sufficient to falsify the conditions on at least one node along all possible paths from the root to the leaves.

Dependency graphs are valuable for the analysis of complex attacks with several preconditions. A typical example is a public web service that due to a particular vulnerability can be exploited to get shell access to the server. With limited user rights such

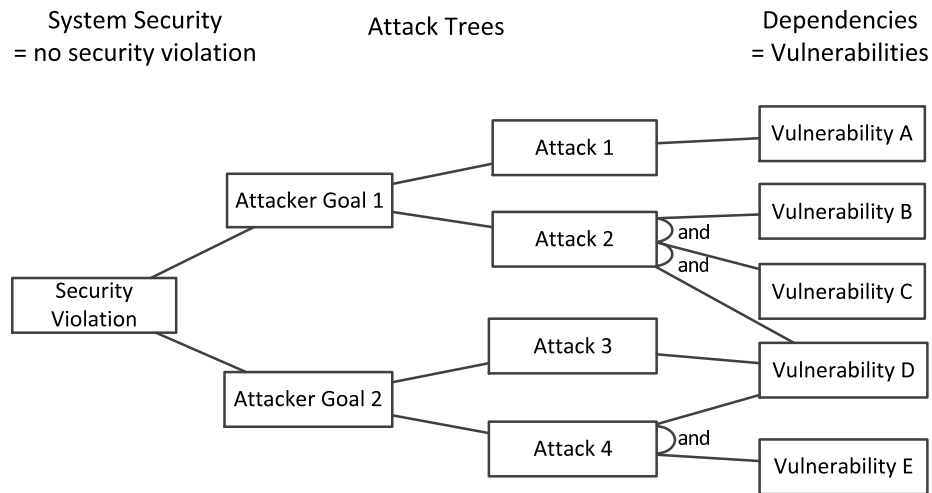


Figure 4.7: Combination of attack trees and dependency graphs with terms of the security ontology. The system security can be violated with the realization of attacker goals which can be realized by attacks which need the existence of a set of vulnerabilities.

an attack is not critical. If a second vulnerability in the system allows elevating the shell user rights to root level the system security has been seriously breached.

Extended Attack Trees (EAT) are a concept to combine attack trees and dependency graphs in one structure. The EAT, as illustrated in Figure 4.7, assesses the security by combining several attack trees that express individual attacker goals. The root of the EAT is a representation of the system security or a specific security objective. The children of the EAT are the attack dependency graphs. In this thesis we consider attack dependencies as vulnerabilities of the system. That means, an attack dependency is fulfilled if the vulnerability is existent in the system. Attacks realize attacker goals (i.e. the root of the attack trees), and attacker goals violate the system security. Thus, the EAT structure can support developers in assessing the system security based on the vulnerabilities and allows a direct application of the security ontology presented in Section 4.1.2.

The EAT is no tree structure due to allowed reconvergences of the vulnerabilities. In the example depicted in Figure 4.7 the Vulnerability D is needed for Attack 2, 3, and 4. If the system does not have that vulnerability, in this example only Attack 1 (which solely relies on Vulnerability A) can be executed.

In Section 6.3.5 a practical implementation of this approach is presented.

Security Patterns

A typical approach in the system design process is the decomposition of the system. This is also a valid methodology for security properties of the system. Similar to attack trees, protection goals of the system can be decomposed to less complex security objectives. The open point in this decomposition process is the final mapping from security objectives to actual implementation. That step is supported by security patterns. The basic idea

of security patterns is to provide a understandable and well-structured description of potential security solutions to foster the reusability of existing knowledge for developers.

Security architectural patterns are discussed in [Sch03] [YB97] and [RFMPG06]. Though the majority of the studies does not focus on the system level, studying the proposed terminology can help improving the definition of our intended solution library. For example the notion of a security degree as part of a pattern description as described in [RFMPG06] can be valuable for the objective security assessment process as it is required in a selection algorithm for security means.

[HSHJ08] proposed a method to measure the security degree of security patterns by decomposing the pattern to smaller sub-components which can be measured more easily. The pattern are associated with security objectives. Thus, security objectives can be measured.

The application of security patterns in WSNs has been discussed for example in [CEKGL08]. There the patterns are used to model an encryption-based access control to sensor data. Therefore, the authors combine the security patterns *sensor data encryption*, *data decryption* and *access control*. The patterns are realized by components which finally closes the gap to the actual implementation. The primary target of security patterns, however, is the support of manual development processes.

Application Profiles

In the domain of WSNs fixed integration profiles for application classes have been proposed. The idea is based on the notion that similar application types also have similar security requirements. This implies that a security analysis once made for one scenario can be reused for other applications. Considered this prototypical security analysis was made by an expert, such an approach does not only save time during the requirement analysis of the system but promises to result in better secured systems. This approach is partly similar to security pattern.

For example [SMK⁺06] argued that there are classes of WSN applications that are exposed to similar threats and have a similar organizational security background. The authors discussed the security requirements of two application scenarios: habitat monitoring and battlefield monitoring. For the scenario of habitat monitoring they identified the requirements of guaranteeing integrity and authenticity of the data, ensuring correct routing to the base station, and the prevention of forged packets. For the scenario battlefield monitoring additional security requirements are discussed, such as defending of attacks on localization and real-time requirements and the concealment of network topological information.

[PA06] discussed several application types and argued that each application can be represented by one of three security levels (low, medium, high). For each security level one composition of security means was proposed. For example WSNs in home environments are assumed to need low level security which means that no security but a low level access control is required. WSNs applied in medical scenarios need high security, implying digital signatures, strong confidentiality, a secure key agreement scheme, and secure routing. [CND09] further refined this approach in an automatic tool that selects protocols based on a security degree that was inferred from the application type.

[LDEH02] proposed an approach to profile applications based on boolean system parameters. The system parameters discussed in [LDEH02] are data confidentiality,

tamper resistance, public key cryptography capability, and presence of rich uncle nodes. The system profile for an application hence was represented by a vector of four boolean values. For instance sensor surveillance needs confidentiality, but no tamper resistance, PKC or rich uncles, resulting in the vector

$$\text{security vector} = [\text{true}, \text{false}, \text{false}, \text{false}].$$

Interpersonal communication additionally needs PKC and tamper resistance resulting in the vector

$$\text{security vector} = [\text{true}, \text{true}, \text{true}, \text{false}].$$

The notion of the authors is to develop (manually) ideal solutions for each of the 16 resulting configuration patterns.

4.7.3 Bottom-up Security Aggregation

Beside the top-down security approaches some ideas exist how security properties can be engineered from bottom up. Formal approaches use security axioms to construct more complex protocols. For instance [Can01] proposes a framework that breaks down the security protocols in atomic cryptographic tasks that can be combined to composed protocols. This process is related to the formal security models discussed earlier in this chapter.

In practice we have no security axioms but atomic security components that should be composed to a system. The issue is the inability to assess the quality of nonfunctional properties such as security, reliability, fault-tolerance, performance, availability, safety, etc. [Voa02].

Considered a developer assembles a system consisting of two components A and B. If A provides good integrity and B provides good concealment, what properties can we expect when we connect both modules? Basically there are three options:

- The composition sustains both properties (good integrity and good concealment) plus an uncertain additional value that has to be evaluated manually,
- The composition has both properties and some additional predictable properties, such as secure integrity or authentication,
- The properties of the composition is entirely uncertain.

[OMKD09] describes a rather practical approach to assess security assurance for systems of aggregated components. There the security assurance level of a entity is the result of an aggregation function of the assurance levels of the sub-components, i.e.

$$AL(\text{TopLevelEntity}) = AGF(AL(\text{Entity1}), \dots, AL(\text{EntityN})),$$

while AL is the assurance level and AGF is the aggregation function. The base assurance levels of the child components are probabilities for the resistance of the selected component against a specific attack. Based on the work in [PR07] three types aggregation functions are discussed:

recursive minimum: The security degree of the aggregating component is determined by the minimum degree of the used components.

recursive maximum: The security degree of the aggregating component is determined by the maximum degree of the used components.

recursive weighted sum: The security degree of the aggregating component is determined by a summation over the weighted qualities of the used components. This approach implicates the use of security properties measured in an interval scale.

[OMKD09] concludes that in most cases a weighted sum is the preferable option without providing a general recipe how the weights should be selected.

In Section 6.3.2 we will present an advanced version of this approach to assess security properties of composed systems.

4.7.4 Security Metrics

With the goal of developing security solutions in a objective manner, metrics are needed that allow to compare different algorithms and systems.

Security metrics –similar to software metrics– have requirements to be applicable reliably and systematically. In his book about security metrics Jaquith [Jaq07] expressed the criteria for security metrics as:

- Consistently measured without subjective criteria,
- Cheap to gather preferably in an automated way,
- Expressed as a cardinal number or percentage,
- Expressed using at least one unit of measure such as defects, hours or dollars,
- Ideally also contextually specific - relevant enough for stakeholders to take action.

While these requirements are valid in general, Jaquith mainly considered metrics that allow real measurements of systems. The majority of security metrics in literature focuses on measurements of security properties of implemented systems. The proposed metrics are motivated by the need to measure the security status of the system [Jan10], and to measure the effect of specific security means in practice. For instance [SA09] discussed applicable metrics for security measurements for mobile ad hoc networks. A typical example is the reduction of virus infections per month of computers in a network before and after the deployment of an anti-virus program. Another example is the ratio of bad packets received by the radio to the number of packets forwarded to the application layer.

However, the metrics do not implicitly allow to predict the result of applying the protection means in advance, which does not mean that such metrics cannot be applied in a composition process. With sufficient empirical data and corresponding confidence also metrics based on measurements can be applied at design time.

In case of virus scanners we know from experience that virus scanner A detects a specific percentage of virus infections so that we can apply that number what allows to predict the effect during design time. It works for virus scanners because their systems and the environments are relatively homogeneous and the security problem is well-understood. In WSNs each system differs significantly.

So what is needed are a-priori metrics that help to assess the degree of security of a system before integration of the system.

Software Metrics

Classic software metrics are not directly related to system security but they can indicate potential issues with the software that can lead to security problems. Software metrics measure the way software is constructed and may indicate implementation vulnerabilities. Tool-supported source code analysis with the aim to detect common software vulnerabilities, such as buffer overflows, has been presented in [CCS06].

While software metrics are valuable in general, they are not the sort of metric that helps to assess the degree of security of a system. The benefit of such approaches for black box component composition is limited. We presume that the software vulnerabilities are fixed before the software components are subject for the system composition.

Intrinsic Security Metrics

As addressed in [Jan10], the “development of computing components that are inherently attuned to measurement would be a significant improvement in the state of the art of security metrics”. Generally only few general security metrics are known. Most of them relate to the strength of cryptographic primitives. [JL97] discussed a set of metrics to assess cryptographic mechanisms. They include:

Key length: Symmetric ciphers are often directly compared by the length of their keys. For example AES has 128 bit keys and DES 56 bit.

Equivalent key length: A widely accepted metric for cryptographic primitives is the equivalent key length. It says what would be the key length of a symmetric cipher with the same degree of security. For example the cryptographic strength of RSA 2048 bit equals a 112-bit symmetric key [LV01]. Key equivalents for symmetric and asymmetric ciphers and cryptographic hashes are also proposed by NIST. The basis of the equivalent key size is the expected time needed to break the system.

Attack Time expresses required time to perform the fastest known attack. The attack time methodology also allows to assess the degree of security of non-cryptographic protocols and primitives—simply by predicting the time needed to break the scheme. To avoid the uncertainty of applied computer hardware the unit of the attack time may be given in million operations to break the primitive.

Assurance Levels are often considered as rationale metrics like percentages. They describe properties very precisely and can be integrated in many formulas. Unfortunately, it is non-trivial to estimate the assurance of a protection means as 95% – and what would be the difference to 94%? Today, methodologies to estimate any sort of security in such a precision are still missing.

Monetary budget is a general and well-understood metric that also allows to compare the system security. Such a metric indicates the monetary budget required to successfully perform an attack on a specific primitive or system.

Qualitative Security Classification Metric

Since absolute scales appear not suitable in all situations, in this section we propose a qualitative metric that still is expressive and consistent. The basis of the scale is monetary

Table 4.4: Qualitative Security Classification Metric

Class	Resistance against		
	Attacker	Tools	Monetary
0 - none			
1 - low	No actual attack attack can be succeed by accident		
2 - medium	curious hacker	common tools	<\$10,000
3 - high	organized attacker (aca- demic, crime)	special tools	<\$100,000
ultra	large organized attacker (crime, government)	highly specialized tools, laboratory	>\$100,000

budget and the attacker classification presented in Section 4.2. There we presented four classes of attacker with characteristic tools, knowledge and budget. The security metric we are proposing determines the security strength of a component or algorithm based on the resistance against attackers from the initial scheme. That means, an algorithm belongs to class c if it resists all attacks from attacker groups smaller than c . Due to this simple rule, this classification can also be applied as consistent general security metric. The result is a qualitative security classification scale (QSC) of attacker resistance as shown in Table 4.4. In practice this means that in the field of symmetric ciphers AES (128 bit) is supposed to resist even a class 3 attack. It is known that DES (56 bit) can be broken by a class 2 attacker (academic [KPP⁺06] in less than a week. A 32 bit cipher can surely not broken by accident, but an intended attack can break it easily. So an implementation without encryption would be an example for class 0, 32 bit cipher is class 1, DES is class 2 and AES is class 3.

We intentionally do not support a fifth class, i.e. could resist attack from large organized attacker, for two reasons. First, it is widely accepted that there is nothing like absolute perfect security. Further, it is impossible for us to perform a reasonable prediction about what a large organization like a government is able to break. Consequently, the assessment between class 3 and class 4 would be more a guess. So the meaning of class 3 is: if at all it can only be broken by very large organizations.

QSC can not only be applied to a holistic security assessment but to assess the security of individual security attributes of a system. For instance it can be applied to assess each of the major security attributes confidentiality, integrity, and reliability. As example a system provide have high confidentiality, but none integrity, meaning that even organized attackers should not be able to break the confidentiality mechanisms of that assumed system, while the integrity could even be compromised without an actual attack.

At the beginning of this section we cited five criteria a good security metric should fulfill. QSC cannot satisfy each aspect. However, it is relatively cheap to gather, it has an implicit monetary unit as objective measure, is easily understandable, and is contextually specific since the metric can be applied to assess specific security sub-criteria. A weakness is that the metric just represents an ordinal scale. A direct monetary metric would represent a rationale scale. The assessment of such a metric would be much more deliberately.

Since we are convinced that the QSC is a reasonable and suitable security metric it is applied as dominant security metric for the rest of this thesis.

4.8 Conclusions and Implication for the Thesis

This chapter studied security for wireless sensor networks and identified methods to assess security during system development. Therefore, first the term security was defined. Security in this thesis is used as umbrella for the three more specific security attributes:

Confidentiality concerns the protection of outgoing data,

Integrity concerns the correctness of data and information,

Reliability concerns readiness and correctness of the services.

The abstract security terms are important to find a common language that helps expressing security-related goals. The terms, however, do not help to develop or assess security of systems. For this purpose a general security ontology was introduced. It explains the relations between the system and its security goals to the attacker and its goals. The connecting elements are attacks, vulnerabilities and countermeasures. This security ontology is the key for several security models that will be implemented in Chapter 6 of this thesis.

The following sections presented a study of attacker motivations, actual attacks and general countermeasures in the domain of WSNs. This analysis is valuable to understand how security is threatened in such systems and how the mechanisms work to defend against attacks. It also practically validates the relations shown in the security ontology.

In the second half of the chapter approaches to configure security in WSNs are studied. Recent research activity presented security toolboxes and transparent layers to provide security in WSNs. Both approaches are characterized by a lack of adaptability, because the programmers preselected static set of protection mechanisms in the boxes, which appears not suitable for the heterogeneity of WSNs. More flexibility is provided by security composition approaches. KMS, the Cionca approach and FABRIC help integrators in configuring WSNs by proposing security methods suitable to user requirements. KMS and the Cionca approach are promising and are subject for integration in Section 6.1.

We extended our search for approaches to assess security to the domain of security engineering. While formal security models could not satisfy the needs for the intended security tool, some other practical approaches could be identified:

- Attack trees and dependency graphs appear valuable for evaluation of systems which are exposed to variety of threats.
- The concept of application profiles is reasonable since it is justified to assume that similar applications have similar security requirements.
- Bottom-up security aggregation is an interesting approach to assess the security strength of a system based on its individual properties

Finally we studied possible metrics to express the security strength in a consistent way. Intrinsic measures such as key length and expected time to break the system exist, but they are not applicable for all sorts of algorithms. As answer we propose a qualitative security classification that measures security strength basically compared on the monetary strength of an attacker.

In this way, the mechanisms and approaches identified in this chapter will be an important contribution for the practical security models introduced in Chapter 6.

Chapter 5

configKIT- A Component-Based Configuration Toolkit

In Section 3.1 of this thesis several approaches for software and system engineering for WSNs have been described. The survey concluded that, while the importance of systematic WSN engineering processes has been widely accepted, still clear answers on processes and their implementation in practice have been missing. To some extent this lack of implementation is caused by a fail to describe and accept a holistic design flow which starts with user requirements and finishes with an actual deployment.

In this chapter such a design flow will be introduced. Based on the properties of WSNs explained in Section 2.1.2, within this design flow we follow the notion of a lightweight application layer that relies on a highly flexible service composition. This process follows the general assumption that the sensor network nodes are configured once before deployment. Following the experiences in compile-time synthesis promises to result in the best performance.

We further decided in favor of a component-based approach in order to foster reusability of already available hardware and software modules. The ultimate goal –as also stated in the introduction of this thesis– is an automatic selection approach that selects the components suitable for the given user requirements. The corresponding selection algorithm is presented in Section 5 of this chapter. This process utilizes powerful data structures for the representation of requirements, components, system structure and system semantics. After an introduction of the design flow and the data structures, the structures are introduced in section 2 to 4 of this chapter.

This chapter concludes with a presentation of the currently implemented tool chain.

In this chapter we apply many concepts of system development introduced in the related work chapters. This may happen implicitly, while important aspects are referred with a pointer to the according subsection.

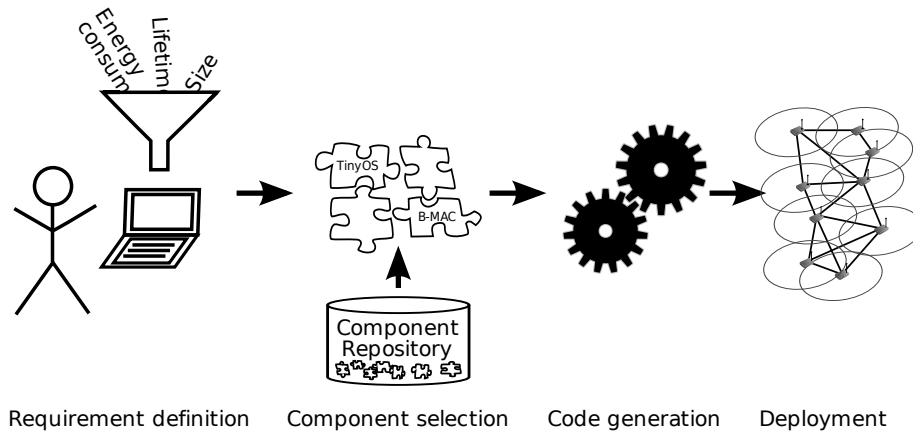


Figure 5.1: Stages of our envisioned partly automated WSN design process.

5.1 Design Flow

5.1.1 Four-step Development Flow

Figure 5.1 shows the general flow of a component-based composition-driven design process. The process steps focus on the actual development process performed by the user. It is assumed the user can execute the design process without expert support. It is a precondition of the process that the components are developed (by the component developer) and available in a component repository (assembled by a framework builder). The roles in the CBD process were introduced in Section 3.4.3.

The process starts with requirements defined by the user, for which components and software modules are selected that promise to satisfy the user's requirements. The selection and assessment process employs a repository containing models of components and their properties. Based on the resulting configuration, the actual source code is generated and compiled. Finally, the sensor nodes are equipped with the resulting code images and are deployed at the application site.

It can be seen that the requirement phase and a component selection converge in a process that leads to the final integration (code generation and deployment). Therefore, in its structure the process embodies an instantiation of the Y-development model discussed in Section 3.2.

The four major steps of this process are discussed in detail in the following.

Step 1: Requirement Definition

The goal of the requirement definition phase is the full specification of the WSN mission. The inputs given in this step must be sufficient to drive the following steps in the development flow. This is particularly true considered it is the goal that the following steps can be performed in an at least partially automated fashion. This means the requirements have to be complete, technically correct and unambiguous.

We presume that end-users are able to formulate abstract application requirements sufficiently and correctly. However, since we do not assume they can use technical WSN terms

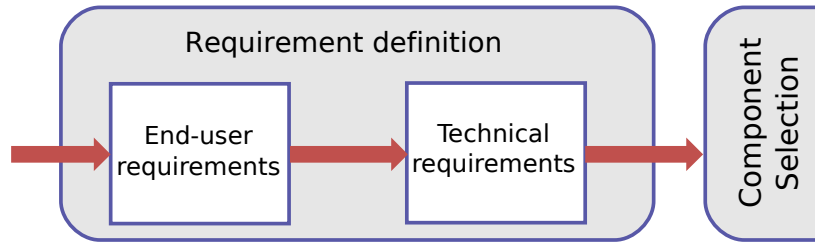


Figure 5.2: Two step requirement definition process: The end user first defines what the application needs. From this the technical requirements are inferred. Technical requirements are the input for the application composition process.

correctly, a deduction step is required to infer the technical terms from the requirements given by the user.

This leads to two general questions: First, how can a user enter the application requirements so that they are understandable to him and the WSN engineer, and also usable in a formal framework? And second, how to deduce the technical terms from the (partially fuzzy) end-user requirements?

The answer, further discussed in Section 5.3, is a two-step requirement definition process. It is illustrated in Figure 5.2. The user enters the application requirement in a language he understands. Technical means to acquire user requirements have been introduced in Section 3.3.2. Based on this survey we propose a catalog-driven requirement definition phase. From this the technical requirements of the application will be deduced. This output is the actual input to the following WSN engineering steps.

Step 2: Component Selection

The component selection concerns the automatic selection and composition of suitable and compatible software and hardware components to a system. The composition must satisfy the functional and non-functional requirements given by the user. Preferably the composition additionally performs optimization of non-functional attributes in order to improve properties such as lifetime, energy consumption, performance, security, or monetary costs. Beside the user requirements, a repository of components for common problems is the primary input for the component selection. Therefore we can express the composition step as mapping from requirements and repository to the composition:

$$f(\text{Requirements, Repository}) \rightarrow \text{Composition}$$

As known for standard component-based development (see Section 3.4) this mapping is non-trivial. Challenges are

- finding a language representation that covers requirements and components
- a-priori estimation of the properties of the composition
- definition of a selection algorithm with reasonable search efforts

We will discuss these issues in Section 5.5 while several concepts from the domain of component-based development (see Section 3.4) are applied.

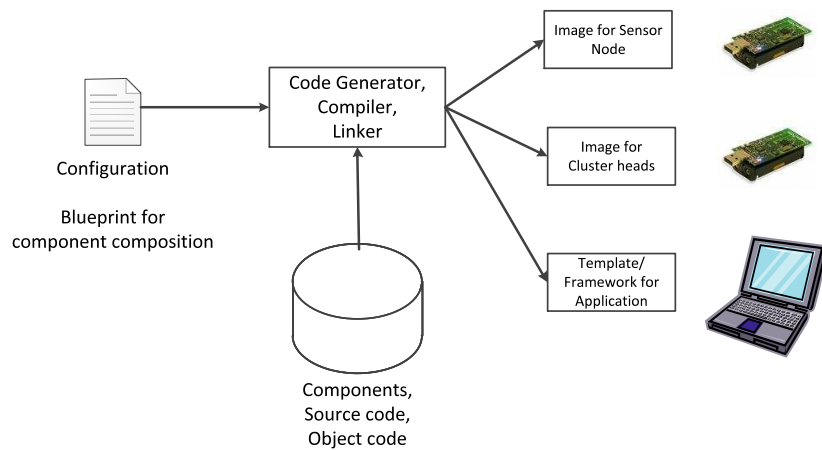


Figure 5.3: Code Generation process: Using a system configuration and source codes of the components, a code generator builds code images for the WSN. Due to the heterogeneity of WSNs individual (but compatible) images must be generated for different nodes and tasks in the network.

Step 3: Code Generation

The code generation finally translates the component composition described in a specific configuration to a system that eventually can be deployed. This process step receives a well-structured configuration and the components that should be combined. The general idea of this code generation process is shown as Figure 5.3. The figure also illustrates that the resulting code images for WSNs mostly differ in code images for sensor nodes and base stations. In many scenarios nodes with different tasks have different code images. For instance, head nodes of sensor clusters often have a different, more powerful hardware configuration which must also be respected in the software components.

The practical appearance of this step heavily depends on the applied component model. Black box component models instantiate the components in the configuration and just link the component objects to one system. It requires that the object code is already available as platform-depended byte code.

The diversity of sensor node platforms does not foster the application of object code. That is why the integration for these WSN systems works on actual source codes which are translated to platform-depended byte code during this process.

Several approaches such as SNACK [GKE04] or FABRIC [Pfi07] convert the component models and the glue code in a standard implementation language such as C or nesC and apply the established sophisticated compiler and linking technology already available for the target platforms. This methodology promises good performance and fast results. Disadvantages are the lack of maintainability since the generated intermediate code in most cases is not understandable. This eventually also affects the testability of the program because the standard debugger tools run on the intermediate code. Macro debugging as proposed in [SHH⁺09] is a promising first step, while the general applicability is not provided. A final problem of the merged code is that changing of components usually needs a step back to the component selection step, which generates a new general configuration. A change of components to runtime is not feasible with this methodology.

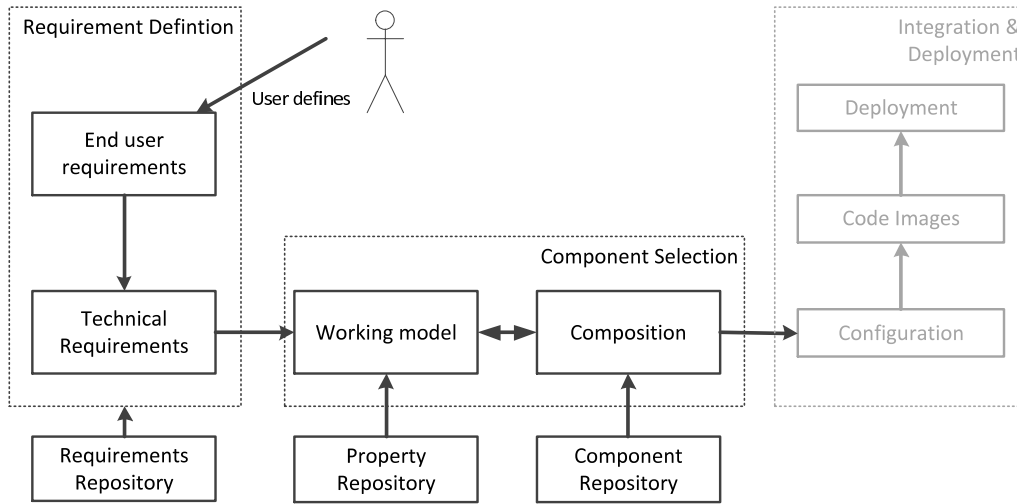


Figure 5.4: The basic configKIT development flow: A user defines the requirements which will be refined to technical requirements, as part of the Requirement Definition. The output defines a working model which controls the composition, as part of the component selection. The output is input for the integration phase. Requirements, properties, and available components may be preselected by the application domain space.

Step 4: Deployment

The final step in the WSN development process is the deployment of the nodes. This process step does not only entail the physical deployment of the nodes but also the programming, testing and debugging in the field. Later it also includes the maintenance of the network.

As of today, few approaches have been proposed to support this phase of WSN engineering. The support deployment process for example by graphical support of node placement is part of the TASK toolbox [BGH⁺05].

Test and debugging has been discussed in [SHH⁺09] and [RR07].

A rich survey of experiences during sensor node deployment is presented in [BISV08]. In fact many practical reports of sensor node deployment have at least a section of lessons learned and what went wrong. Most common problem is that the network works well in the lab but not in the field. Usually physical effects such as radio propagation or effect of temperature on the nodes were not correctly assessed in theory.

While in the configuration approach presented in this thesis we do not address deployment explicitly, the support of considering such real-world effects early in the development flow can reduce the impact of such potential obstacles.

5.1.2 The configKIT Development Process

Based on the general design flow, in this section we focus on the configKIT core development process consisting of requirement definition and component selection. Figure 5.4 shows the general steps in the dashed boxes, containing explicit objects needed for the corresponding step. Integration and deployment are combined and grayed out because they are not in focus of the configKIT design flow.

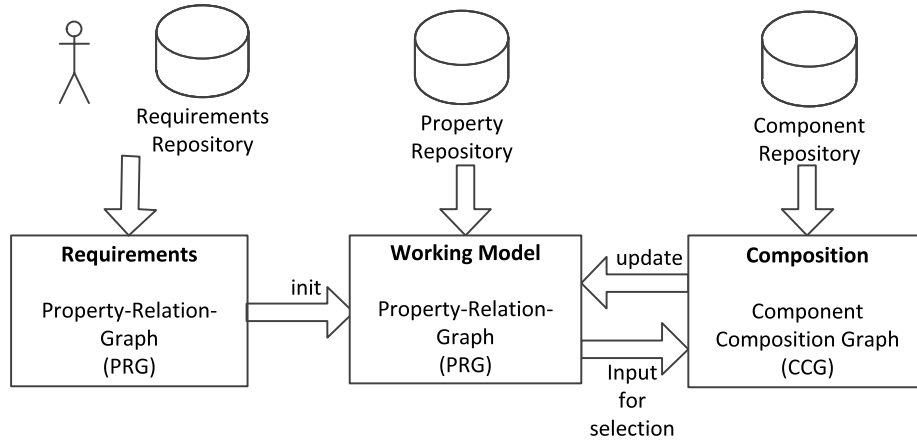


Figure 5.5: Major data structures needed for the configuration process: Requirements initialize the Working Model. The state of the WM is input for the composition which updates the status of the WM.

As already indicated in the previous section, the requirement definition phase is split in the definition of the end user requirements and the refinement to technical requirements. A user chooses requirements from a set of requirements defined in a requirements space. The requirement space may be tailored based on the development domain. The technical requirements provide inputs to the working model.

The working model (WM) contains all properties and information available for the development of the system. It is assumed that the properties and their interrelations are stored in a property repository, which also can be tailored based on the development domain.

The component composition, i.e. the actual component selection, works in close cooperation with the working model. The working model provides inputs and constraints for the selection and the composition updates properties of the WM. Finally the WM has the information to determine whether a composition of components satisfies the requirements and whether the implementation is technically sound. We will refine this fuzzy requirement in the following section.

It is the goal that all steps from the requirement refinement to component composition can be automated. Therefore we need a more precise understanding and definition of the data structures and information flows. Therefore, in the following, the data structures are introduced starting in a birds-eye view and zooming successively into details.

5.1.3 The configKIT Data Structures

The data structures of the global composition process and its data structures are depicted in Figure 5.5. It is a refined view on the configKIT process shown in Figure 5.4, isolating the data structures *Requirements*, *Working Model* and *Composition*. The structures in detail are:

Working Model (WM) is the model of the domain, and thus represents the semantics of the development process. The WM can be considered as a rule-book for the composition process. As such it defines one goal of the composition process: all

constraints as part of the model have to be satisfied.

Technically the WM uses a concept of properties and relations between properties to define the model. They are represented in a graph structure named Property-Relation-Graph (PRG). The PRG is described in detail in Section 5.2.

Requirements are initially provided by the user and transformed to technical requirements which eventually parametrize the WM. Requirements are represented as properties and relations between properties. The PRG structure realizes the transformation to technical requirements and enables an easy integration of the requirements in the WM. The requirement definition process is discussed in Section 5.3.

Composition is the model of the system under development. It is the composed application consisting of components. The composition is represented by the Component-Composition-Graph (CCG), which is a graphical representation of the system of components connected by interfaces. The goal within the CCG is to find a composition that is complete and can be integrated. Component model and composition structure are described in Section 5.4.

Between the data structures exist the following data flows:

Requirements → **WM**: The technical requirements parameterize the PRG initially. This step is performed once for the system configuration. Thereafter the PRG contains all information about the requirements. Together with the definition of the requirements the transition to the WM is described in Section 5.3.

WM → **Composition**: The current status or a subset of the status of the PRG is transferred to the (selection algorithm of the) CCG. There the information is applied to execute the component composition. The selection algorithm is discussed in Section 5.5.

Composition → **WM**: The WM evaluates the properties of the system under development in context of the requirements and the domain-specific model. Therefore, the updated properties and relations based on the current component selection are forwarded from the CCG to the PRG of the WM. In the WM the stimulus triggers an update of the status of the PRG system. The mapping from the CCG to the WM is discussed in Section 5.4.4.

All process steps rely on specific information which are stored in repositories:

Requirements Repository: The Requirements Repository is a catalog containing all possible requirements and constraints a designer can set for an application. This also means: if a property is not in the Requirements Database, that property cannot be set for the application. It is the notion that if a property is not part of a formal reusable database it can hardly be supported in the later composition, testing and reasoning steps.

Property Repository: The property repository contains domain- and model-specific sets of properties and relations between properties. These properties represent the working model and thus the understanding of system, requirements and environment. Assessment models for security are examples for optional sets of properties. The selection of the property subsets is domain specific and usually not done by the user directly.

Components Repository: The Components Repository contains all possible design options and the meta-information needed to assess the implications of the composition. Practically the database contains a list of components with their interfaces and individual properties.

5.2 Property - Relations

The working model represents the semantics of the system under development. The semantics in this model are expressed by properties and relations between properties, combined in a structure called Property-Relation-Graph (PRG). As we will see in this section the PRG a powerful abstraction that allows to manage requirements, express system properties, and maintain system models.

To realize these tasks the following requirement the be fulfilled:

Expressiveness: Properties as part of the PRG should describe the attributes of the system to full extend. This includes technical properties of the system, behavioral aspects, but also description of the application and environmental situation. This means that every information in context of the development and usage of the system has to be expressible within the PRG.

Checking of constraint and conflicts: One specific attribute of a system under development are conflicts that may occur during the assembly of the system or at runtime. For example if the application scenario demands a specific node distance, but the radio on the considered sensor node is not able to provide the distance in the required quality, it is a conflict that must be discovered in the PRG.

Extendability and Flexibility: Properties that have to be evaluated during the development process may vary with the applied domains. That is why it is important that the set of supported properties is not fixed. Furthermore, the incremental development of the PRG should be possible without the need for a central managing instance.

For example if a user or a component prefers to define frequency as cadence (e.g. as part of the domain-specific language), the new property Cadence can be added to an existing PRG, and connected to the existing system by a relation that transforms Cadence to Frequency.

5.2.1 Definition

Properties

A property is a characteristic or quality that describes an attribute of the system, or of a part of the system. Basically, a property is everything that can be expressed by a value. That concerns all sorts of attributes to describe the system, its components, its requirements, or its environment. It also includes information to describe functional or non-functional characteristics.

Basically a Property is a named value. Beside the value and a name, properties in our model have additional attributes:

Data type: Properties can represent various data types. Currently the following data types are considered:

- Numbers: Properties can be expressed as integers or floating point numbers. For example number of nodes, available memory, or sending power.
- Strings: Character strings are a means to express names of components. For example the name of a microcontroller or the name of a cryptographic accelerator.
- Enumerations: Enumerations are a list of options (typically strings) that can be set for the property. Examples are:
 - Application.type={Home, Public, Industrial, Medical}
 - Radio.band.type={433MHz, 868MHz, 2.4GHz}
 - Security.strength={none, low, medium, high}
- Sets: Several properties are expressed as sets. In practice these sets are typically based on strings. For example the property *measures*, which expresses the types of measurements in the system is such a set.
 - Sensor1: *measures* = Temperature
 - Sensor2: *measures* = Pressure
 - System(Sensor1+Sensor2): *measures* = {Temperature, Pressure}

Scale: The scale of a Property is important, particularly with respect to the comparison of properties. The scale type of a property is defined implicitly. Properties defined as strings or sets are nominal, so that only equity can be evaluated. Enumerations are considered as ordinal scale, because an enumeration is defined as ordered set of attributes. Properties defined by numbers are considered as ratio scales.

Unit: if applicable the unit of the property should be provided to prevent ambiguous interpretations of a property. For example the property energy is basically meaningless without a unit.

Relations

Properties influence and depend on other system attributes. For example the setting of an application type affects the required security strength. Another example is the required radio sending power that may depend on the distance of the nodes. For this purpose operations on properties are needed. In this thesis operations on properties are generally termed *Relations*. The semantic of relations is similar to a line of code in a standard imperative programming language. For example the operation $a := b + c$, which, in a programming language, assigns the value of the addition of variables b and c to the variable a , is in the configKIT context a relation that expresses the value of the property a as function of the properties b and c .

Similar to classic operations, Relations can be Assignments or Constraints.

Assignments define a property as function on other system properties. An assignment is for instance the formula

$\text{Cost_of_System} := \text{Cost_of_Nodes} * \text{Number_of_Nodes}$, which assigns the results of the multiplication of Cost_of_Nodes and Number_of_Nodes to the property Cost_of_System .

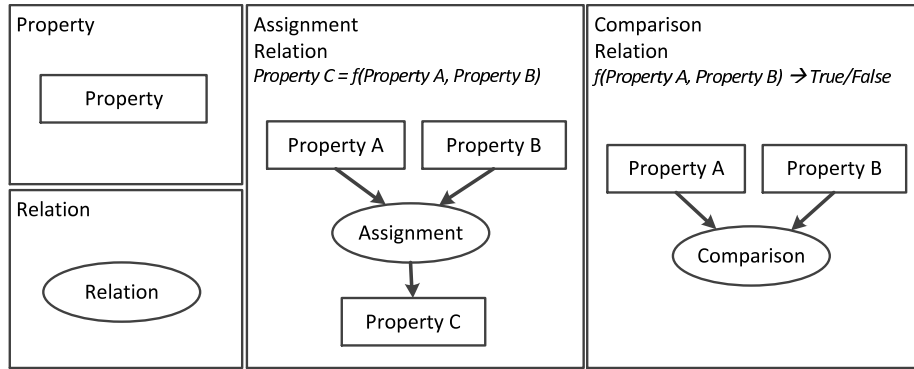


Figure 5.6: Graphical notation of Properties and Relations: Properties are boxes, Relations are Ellipses. The arrow between Properties and Relations indicates whether the connected Properties are input or output of a Relation. Relations with one output are Assignments, Relations without output are Comparisons.

Constraints are boolean functions on system properties. A comparison is for example $\text{Cost_of_System} < 1500$ which returns true only if Cost_of_System is smaller than 1500. In contrast to assignments, constraints do not assign result of the operation to a property. That is why not all comparisons are constraints. A relation which is an assignment of a comparison (e.g. $b := (a == 5)$) is an assignment and no constraint. Also conditional assignments such as $b := (a == 5)?x : y$, which assigns to b the value of x if a equals 5, and otherwise the value of y , are assignments.

While relations are always connected to at least one property, they are not attached to properties directly. Thus relations are independent data types. They inherently define which property should be written and select the required inputs.

Graphical Notation

Figure 5.6 shows the basic principles of the graphical notation of the PRG structure. Properties are represented by boxes, while *relations* are Ellipses. Arrows from *properties* to *relations* indicates that the corresponding *property* is used in the function computed by the *relation*. Arrows from *relations* to a *property* show that the result of the computation in the *relation* is assigned to the *property*. The associated *relation* is an *assignment*. *Relations* without outgoing arrow are *constraints*. The graphical structure of properties and relations is the Property-Relation-Graph (PRG).

Example

Figure 5.7 shows a small example of a Property-Relation-Graph. In this example Light gray ellipses are constraints, and the dark gray ellipses are assignments. The example illustrates a simple energy assessment model. The assignment uses the properties Time and Power to compute the Energy. The value is assigned to the corresponding system property. That Energy is checked against the Property Battery.Power by the constraint “Battery.Power > Power”.

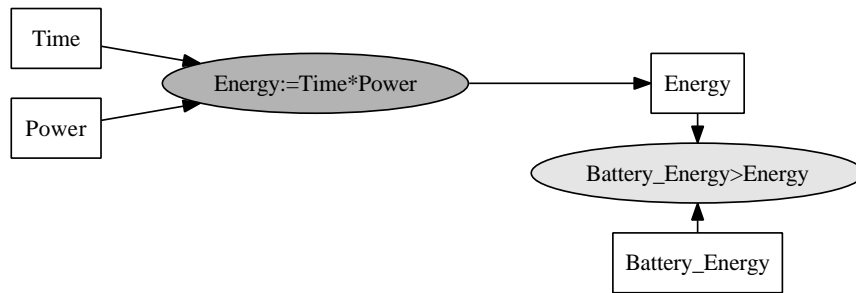


Figure 5.7: Example for a Property-Relations-Graph: The small energy model computes the needed energy and compares it to the available energy from the battery.

Data Model

Figure 5.8 concludes the introduction of the PRG data types in a data meta-model of properties and relations. It shows:

Relations receive input from any number of Properties. It can be refined to Assignments and Comparisons. Comparisons may be Constraints of the system.

Assignments are relations but additionally define one Property. The definition is based on the inputs of the Relation.

Constraints are relations that compute a function with boolean result.

Property delivers input to any number of Relations. May be defined by any number of Assignments.

PRG is composed of Relations and Properties. Both data types can also be in any number of PRGs. The PRG is the interface for Requirements, Components, and the System Model to access Properties and Relations.

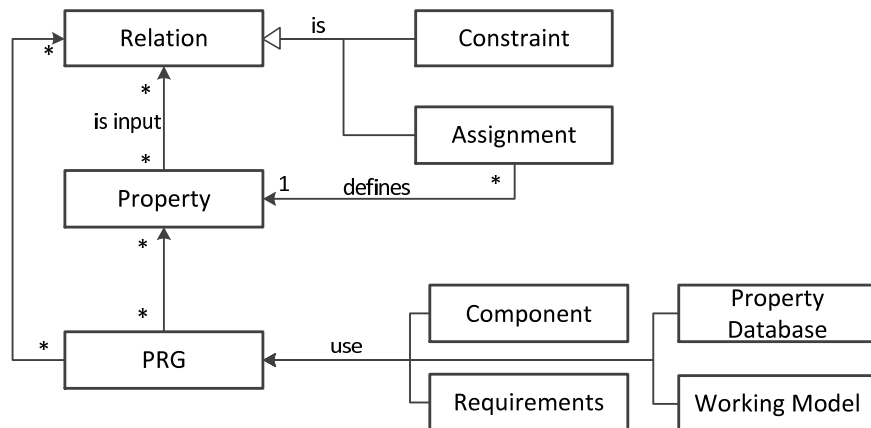


Figure 5.8: Meta-model of the Property-Relations: A Relation receives input from any number of Properties, and can be refined to Assignments, which assign a property, and Comparisons, which may represent system constraints. Properties and Relations may be defined by Requirements, Components and the System Model.

Later we will see that in fact components, requirements as well as property databases and the working model are instances of the PRG.

Discussion and Related Approaches

The idea of representing data dependencies in a graph is not new. For example *Data-Flow Graphs* as used in compilers represent operations (analog to relations) and variables (analog properties) in a dependency graph, illustrating the logical dependencies to compute complex operations or algorithms. Data-Flow Graphs just illustrate the data flow and do not actually work on the data. Hence they are also not applied to identify conflicts of the values.

Logic Truth Maintenance Systems [DK86] can be applied to evaluate the truth and thus the conflicts of the system. They are also called constraint solvers or satisfiability (SAT) solvers, since they are means to solve large systems of constraints and dependencies very efficiently. Typically they work on boolean values.

Recently integer constraint solving has been discussed [SMDD10], extending the typically boolean constraint models of product lines and feature systems by integer variables, which is a promising approach but still does not cover the variety of types considerable for system properties.

Logical programming languages like PROLOG [CM03] are able to maintain and solve complex truth systems. However, maintaining the logical truth of the PRG is just one aspect in the configKIT configuration tool. It has to interact with the component composition and the requirement definition. That is why we decided to develop the independent PRG structure with the intention to express the complex structure of relations of system properties. It is a rather intuitive tool to assess the system state. At later stage an adaptation or mapping of sophisticated SAT techniques is intended.

5.2.2 Formal Specification of the PRG

Before the practical specification of the PRG we need to define a general graph structure that is also applied for component graphs. The operations of the **cgraph** structure are subsumed in Table 5.1

cgraph Structure

The structure **cgraph** is a graph $\langle N, E \rangle$ while N is a set of nodes, and E are the edges $E \in N \times N$.

On this graph structure we can define the following operations:

Table 5.1: Operations of the cgraph structure.

Operation	Explanation
$v \times E^i$	set of reachable nodes in exactly i steps, starting with the set of nodes defined in v and following the edge function E
E^{-1}	inverse edge function
E^*	total reachability function
$G_1 \cup G_2$	merge operation of two cgraph structures. It is a union of nodes and edges.

- transitions from a given Node $n \in N$: $v' = n \times E$
 - $v' = \bigcup_{i \in N} \begin{cases} i & \text{if } E(n, i) = 1 \\ \emptyset & \text{if } E(n, i) = 0 \end{cases}$
 - the output is a set of all reachable nodes (within one step) from n
- transitions from a set of nodes $v \subseteq N$: $v' = v \times E$
 - $v' = \bigcup_{i \in v} i \times N$
 - the output is the set of nodes v' reachable from the set of nodes v within one step
- cross multiplication of two Edge functions $E_1, E_2 \subseteq N \times N$: $E' = E_1 \times E_2$
 - $\forall i \in N : \forall j \in N : E'(i, j) = \bigcup_{k \in N} E_1(i, k)E_2(k, j)$

Thus, the set of nodes that can be reached within two steps of E starting from a node $n \in N$ can be computed by:

$$v' = (n \times E) \times E \text{ or } v' = n \times (E \times E)$$

The first option would compute the vector or nodes reachable within one step first, and that result is forwarded a second mapping. The second option computes the second-step edge mapping first. Then n will be applied to the resulting new edge function. The results are equivalent, which demonstrate the associative characteristic of the cross mapping functions defined above.

Within the concept we can further define the following operations

- Inverse Edge function E^{-1}
 - $E^{-1} = \forall i \in N : \forall j \in N : E^{-1}(i, j) = E(j, i)$
 - the inverse E corresponds to the transposed matrix E^T
 - The mapping E^{-1} describes by which set of nodes the individual nodes are determined
 - generally we can write $E^{-n} = (E^{-1})^n = (E^n)^{-1}$
- neutral element of the edge function

$$- E^0 = \forall i \in N : \forall j \in N : E^0(i, j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

Often it is valuable to address all nodes that can be reached within a given amount of steps. E^s provides the transition mapping for exactly s steps, but does not give information on nodes

- we denote E^{*s} the transition function for all nodes in distance $\leq s$

$$- E^{*s} = \bigcup_{i=0}^s E^i$$

- E^* is the total reachability function. Since the longest path in a graph with n nodes must not be longer than the cardinality of the nodes $|N|$ we can assume $E^* = E^{*|N|}$

Consequently, $n \times E^*$ delivers all nodes reachable from a starting node $n \in N$. $n \times E^{-*}$ provides a set of nodes that precede the node n . Both operations are extremely valuable in the analysis of dependencies as they occur for properties (one property is a derivation from another one) and components (one component uses other components). In the latter context the operations help to identify all components that may use a specific component, or to address the set of components a specific component uses.

PRG as Instance of the cgraph Structure

Applying this CGraph structure we can define

- PRG as triple $\text{PRG} : \langle P, R, E \rangle$ as
 - $\text{cgraph} \langle P \cup R, E \rangle$
 - over Properties P , Relations R
 - with edges $E : (P \times R) \cup (R \times P)$,
which implies that edges in the graph can be directed from properties to relation $(P \times R)$ and from relations to properties $(R \times P)$.
- Properties
 - are attributes and represent values
 - values can be numbers, strings or selections
- Relations
 - are represented by the triple $r := \langle p, A, d \rangle$,
 - while p are the used properties, A is the assigned property $p \rightarrow A : p \subseteq P; A \in P \cup \emptyset$
 - d describes the mapping as a textual expression of a function that has to be parsed separately.
 - p and A are represented in the PRG:
 $\forall i \in p : E = E \cup (i, r)$
 if $A \neq \emptyset$ then $E = E \cup (r, A)$
 - that means, all incoming edges come from parameters of the relation. The optional outgoing edge (to A) points on the property that receives the assignment. If there is no assignment ($A = \emptyset$), the relation is a constraint.

Example

The expressiveness of the PRG structure should be demonstrated with a small example: Earlier in this section Figure 5.7 showed a small example of a PRG. Figure 5.9 represents this example with symbolic values for properties and relations. Expressed as cgraph the structure is:

- Properties $P : \{P1, P2, P3, P4\}$

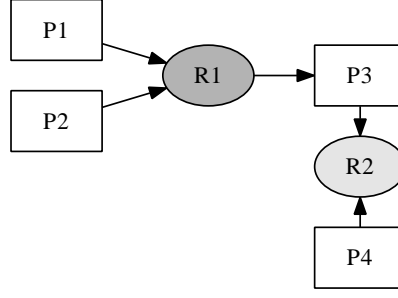


Figure 5.9: Symbolic representation of the example 5.7.

- Relations $R : \{R1, R2\}$
 - $R1 : \langle \{P1, P2\}, \{P3\}, "P3=P1*P2" \rangle$
 - $R2 : \langle \{P3, P4\}, \emptyset, "P4 > P3" \rangle$
- Edges $E : \{(P1 \times R1), (P2 \times R1), (P3 \times R2), (P4 \times R2), (R1 \times P3)\}$

Alternatively, the edges can be expressed as full matrix:

$$E = \begin{matrix} & \begin{matrix} P1 & P2 & P3 & P4 & R1 & R2 \end{matrix} \\ \begin{matrix} P1 \\ P2 \\ P3 \\ P4 \\ R1 \\ R2 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

On this structure we can determine the dependencies in the graph. The following table shows which properties are reachable from each node of the graph and which properties influence the value of a property. This is done with the operations $n \times E^* \cap P$ and $n \times E^{-*} \cap P$, respectively, while n is the considered node. In case of $n \times E^{-*} \cap P$, $n \times E^{-*}$ computes all nodes in the graph the corresponding node depends on, and $\cap P$ filters the properties.

n	$n \times E^* \cap P$	$n \times E^{-*} \cap P$
P1	{P1,P3}	{P1}
P2	{P1,P3}	{P2}
P3	{P1,P3}	{P1, P2, P3}
P4	{P1,P3}	{P4}
R1	{P3}	{P1, P2}
R2	{}	{P1, P2, P3, P4}

In this small example the results are not surprising. More complex graphs indeed are not that obvious. The operations may help to detect properties that influence a violated constraint. In our example all properties influence the constraint $R2$. Expressed in the 'real-world' notion of the symbols (Figure 5.7), it means that if the battery is not sufficient

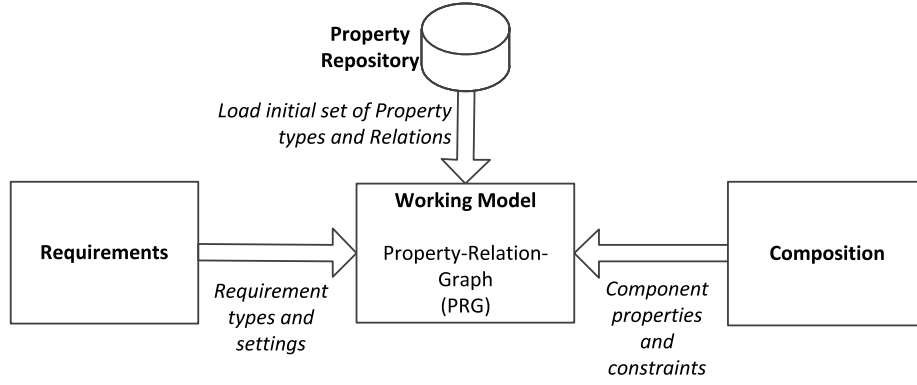


Figure 5.10: Dynamics influencing the PRG in the Working Model.

for the application one can change the battery, or influence the energy consumption by working on duty time or power consumption.

5.2.3 Dynamic Aspects of the PRG

With the static structure of the PRG defined, this section focuses on the dynamic behavior of the PRG. The static structure so far only expresses how the properties of the system are connected. But it does not describe how the graph may change, how actual data will be entered and how the data propagates through the graph. Basically we can differentiate two general dynamics in the PRG. The first affects the structure, i.e. the graph itself, and the second concerns the content of the properties.

Figure 5.10 shows the sources and sorts of modifications of the PRG that represents the Working Model. With its instantiation the WM loads the properties and relations from the property repository. These properties and relations are not application-specific but represent general knowledge and its context. This loading operation is structure-related. The structure of the PRG is not fixed after this initial loading. Specific requirements may be loaded into the PRG, and new properties and constraints of the composed system can be added.

Furthermore, components and requirements affect the content of the PRG. That is, they define the values of the properties, while the values are propagated through the graph driven by the relations.

In the following the structural and content-related dynamics are specified in detail. The definitions apply the PRG structure G which is defined as $G : \text{PRG} \langle P, R, E \rangle$, while P are properties of the system, R are the relations, and E are the edges.

Structural dynamics of the PRG

The structural dynamics consist of two general operations: adding properties to the PRG and adding new relations:

Adding Properties to the PRG is a union operation of the existing properties P and the new properties p . The new PRG G' after the addition of p is defined as: $G' = \langle P', R, E \rangle$, $P' = P \cup p$.

Relations and edges in the graph do not need to be adapted. If the properties are new to

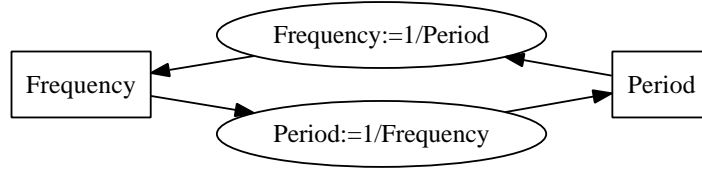


Figure 5.11: Example for a circular property relation.

the system the operation is trivial. Further definitions are required if p is already part of P , i.e. $(p \cap P \neq \emptyset)$. Conflicts in definition must be prevented. Conflicts of content will be analyzed separately, which is discussed below.

Adding Relations to the RPG is more complex, since relations work on properties and express the connection to the properties with the modification of the edges of the graph. To add the Relation $r : \langle p_r, A_r, d_r \rangle$ to the PRG G , we define the operation $G' = G + r$ as:

$$\begin{aligned}
 G' &= \langle P', R', E' \rangle \\
 P' &= P \cup p_r \cup A_r. \\
 R' &= R \cup r. \\
 E' &= E \cup \bigcup_{a \in A_r} (r \times a) \cup \bigcup_{p \in p_r} (p \times r).
 \end{aligned}$$

It adds properties and edges required by the relation. Similar to the addition of the properties, the actual challenge after adding the relations is handling the content of the properties.

Content-related dynamics of the PRG

The purpose of the PRG is not only to model the relations between the properties but to derive dependent properties and to identify violations of constraints. Therefore it is required to define and to propagate the content of the properties.

For this propagation several strategies exist.

- start with the newly set property and follow the edges of the graph recursively and update the corresponding properties. The propagation may stop if it does not change the values of the passed properties.
- Identify the relations that depend on the changed property $(p \times E^* \cap R)$, and update them all repeatedly until the computed values are stable, i.e do not change anymore.

In this propagation and assignment phase non-trivial structure may occur:

Circular assignments is one problem that may occur in the graphs. Figure 5.11 shows an example for such a circle: Period defines Frequency and Frequency defines Period. Such circles are reasons for an explicit termination of the propagation as expressed in both strategies above.

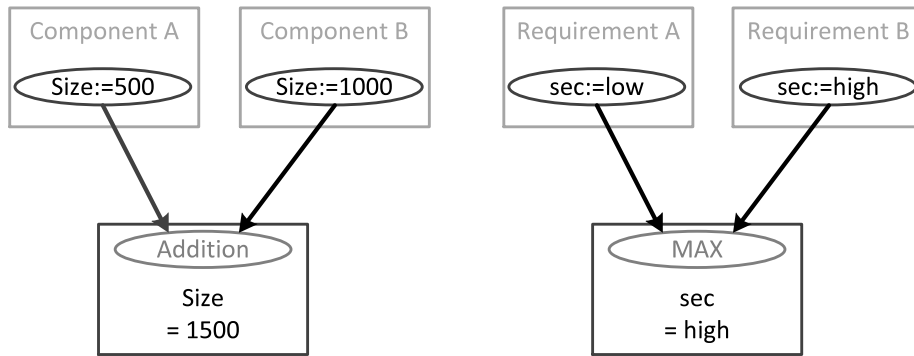


Figure 5.12: Implicit handling of multiple property definitions: A property can define how it handles multiple definitions. The Size property summarizes the multiple inputs. The assumed security (Sec) component choses the maximum of the inputs.

A further mechanism to prevent infinite circles is to flag properties that have been passed in the current update run. Such node will not be updated a second time.

Handling ambiguous assignments is a significantly more challenging issue. Properties can be assigned by other Properties, Components, or the user. Therefore, it is likely that some properties are set more than once. For instance data packet size can be defined differently for each service of the system. This behavior causes possible conflicts in the moment a property is driven by more than one source. It can be handled in different ways:

1. Forbid multiple definitions structurally: This would enforce that each property is always driven by exactly one source.
2. Forbid multiple definitions during runtime: The property is able to receive its value from different sources, but will generate a conflict if more than one source delivers actual data.
3. Attach implicit or explicit relations for properties: Instead of writing directly to the property, external sources write to a entry relation of the property. The entry relation contains the rules to decide what should be assigned. Examples are minimum or maximum, but also the summation. Figure 5.12 illustrates two examples: If for example one system component has the property size:=500 and another component has the property size:=1000 (both defined as relation), the property contains information to add the multiple definitions.

In case of security the general application requirement may be set to low, but a specific requirement demands high security. Then the multiple re-definition has to be resolved to the higher choice, implementing an implicit MAX operation.

Such definitions require the properties defined properly in the property repository.

Handling undefined values is another issue that has to be addressed with regard to the property propagation. If all inputs are defined, parsing and evaluating the relation terms is straightforward. However, the Property Repository contains many properties that are loaded in the PRG of the WM but do not need to be defined for each application.

It results in undefined Properties whose handling is not obvious. We considered the following strategies to cope with the undefined values:

1. Consider undefined Properties as set to a default value.
2. Define the output as undefined in case one of the inputs is undefined. This is the appropriate handling for pure mathematical terms.
3. Comparisons can be handled strictly logical, i.e. it returns true only if the condition is true. If it is unclear due to undefined values it is false. For example consider the assignment:

Security.Concealment := Application.Type=Military?high:low

It sets the required strength for the concealment property of the security to high if the application type is military and to low otherwise. In case the application type is defined, resolving that relation is trivial. However, even if application type is not defined the term can be resolved considered the equivalence operation returns FALSE if the comparison is not TRUE, which it obviously is not if application type is not defined.

We support option 2 for calculations and option 3 for comparisons. If a default value is given in the description of the Properties it is loaded in advance, so that the value is not undefined. An additional default value for Properties without default settings is not supported.

5.2.4 Conclusions

Now we have what we need to represent the semantics of our system as a set of attributes. The PRG is a flexible graph structure consisting of properties, which represent all forms of attributes of the system, and relations which use existing properties, either to define new properties or to implement constraints. The PRG realizes all requirements we stated in the introduction of this section: It is expressive since the properties and relations can maintain all sorts of values, descriptions and their interrelations. The interrelations are a means to fulfill the second requirement: Expressed as constraints the relations check the status of properties and can identify conflicts. The extendability –as third requirement– is supported by means of structural and content-related dynamics.

The PRG as introduced in this section is the underlying knowledge structure of configKIT. It is applied within the Working Model and its property repository. It is also used within the component description and represents requirements as introduced in the next section.

5.3 Requirement Definition

In the requirements definition step of configKIT all input information describing the intended application, its properties and environmental information needed for the development process are collected.

Among others the requirement information includes:

Functionalities: The description of the required functions contains information about what the application should do. This includes a full definition of the mission

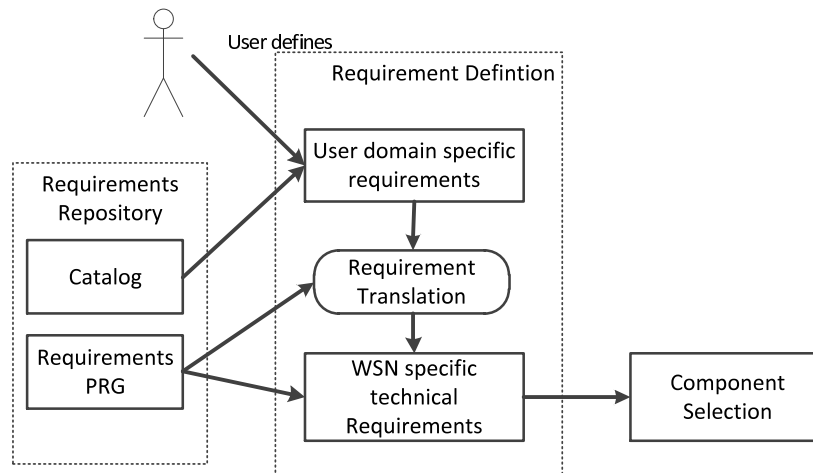


Figure 5.13: Requirement definition process: The user sets the requirements on a domain specific level by selecting and parameterizing attributes provided by a catalog. In a translation step the high-level requirements will be translated to the detailed technical requirements in the domain of WSN. The result of this step is the input for the actual application selection step.

and the use cases, but also technical and functional requirement such as a needed sensor types or specific event notifications. One functional aspect is the selection of the application module which can be chosen either explicitly (by selecting one application component) or implicitly (by describing the application).

Qualities: the qualities describe HOW the functions should be executed. This includes performance aspects. An important sub-category of qualities is security.

Environment: the environment as part of the requirement is a description of the execution conditions. This includes for example size of the network, available infrastructure, radio interference level and sort of accessibility.

Other technical and non-technical Constraints: Additional technical or organizational constraints may influence the design space. Examples are a pre-defined sensor node platform. Other constraints are costs for components and integration.

These inputs have to be provided clearly and unambiguously. It is a problem that end users and application domain experts are not familiar with the terms typically used in WSN engineering. They certainly can express the requirements precisely in their domain-specific language – under consideration of some experience-based hidden subtext. So what is needed is a translation of rather high-level, apparently fuzzy, domain-specific requirements to verifiable metrics that are understood within the WSN development process. The user sets the requirements on a domain specific level by selecting and parameterizing attributes provided by a catalog. In an translation step the high-level requirements will be translated to detailed technical requirements in the domain of WSN. This is executed within the PRG structure, which allows a seamless integration of the requirements in the following application design steps. Figure 5.13 illustrates process. To explain the requirement definition process we first introduce the data representation of requirements, before the user selection step and the requirement translation are discussed in more detail.

5.3.1 Data Representation

In order to reduce the translation efforts between the models it is the goal to express requirements as properties and relations so that they can be directly used in the property-relation-graph structure. The expression of requirements as sort of properties and their relations is quite natural. However, it is not directly clear how requirements are expressed technically. Requirements as they are given by the user can be considered either as properties of the system under development or as relation on existing properties.

To evaluate both options let us consider the following requirements and how they would be realized:

- Application_type is Medical,
- Energy consumption of a node per hour must be below 1J,
- key-distribution protocol is required.

Requirements defined as Properties would need the following properties in the system:

- Application_type,
- Energy consumption of a node per hour,
- key-distribution protocol.

A user could choose between the given application types, define the energy consumption, and decide if the key distribution is required, respectively. The corresponding relation (constraint) would be implicitly attached to each requirement (or property) if required. For *Application_type* no relation is needed since the value is already assigned. For energy consumption a relation would be needed that compares the selected value with the actual energy consumption of the system. For the key-distribution protocol a relation is needed that compares the properties of the modules of the system with the selection.

The advantage of this approach is the simplicity for the user. In case of the application type the potential selection can be shown. Generally, in most cases it is sufficient to make a selection or quantify a parameter directly. The disadvantage is the missing flexibility. New requirement types correspond to new properties with new implicit relations. Additionally in many cases it is necessary to implement properties for the selected value and the actual value.

Requirements defined as Relation needs explicit assignments or constraints to define the requirements. For the three examples it means:

- the application type must be set by an assignment ('Application_type:=medical')
- the energy consumption per hour can be computed and compared within a constraint that represents the requirement.
- adding a constraint that enforces the key distribution protocol must (or must not) be set.

The advantage of this approach is that new requirements can be easily formulated by combining existing property types. The disadvantage is that entering requirements is more complicated. Also type checks and value propositions (such as the selection of application types) are not implicit. It is rather a programmers' approach.

The requirements within the configKIT framework are a compromise between both approaches.

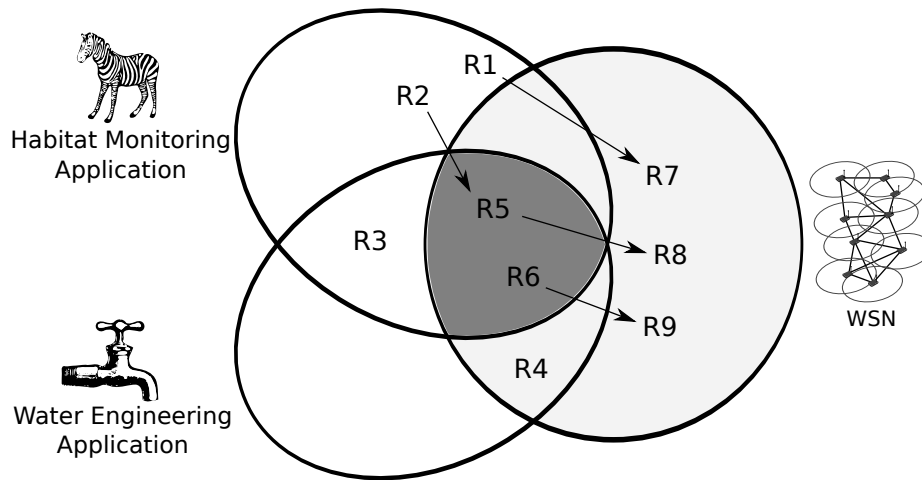


Figure 5.14: Requirement design spaces: Each ellipse represents the space of requirement types that are understood by corresponding domain experts. It is the goal to infer requirements understood by WSN engineers (R4 to R9) from domain-specific requirements (R1 to R3). Solutions are translations (arrows) or a common requirement catalog (dark gray area, contains R5, R6).

A requirement in `configKIT` is one parametrized property data type with a set of implicit relation data types. The relations may define other properties or constraints in connection with other properties. The name of the property may already indicate the sort of attached relations. For instance requirement `Minimum Lifetime` has an attached constraint comparing the requirement with the property `Expected Lifetime`.

As part of the requirement repository a set of requirement types are stored. Requirement types are types that can (but do not need to) be set for applications under development. If selected and defined the requirements (properties and relations) pose the requirements PRG – one graph structure containing all defined requirements for the system under development.

5.3.2 Domain-specific requirement elicitation

In the first step of the requirement definition phase an application designer chooses a set of requirements and defines the parameters. The question however is in what domain the requirements are formulated. Figure 5.14 illustrates the problem: Each ellipse represents the space of requirement types that are understood by corresponding domain experts, for example habitat monitoring designers or waterworks operators, on the left and the domain of WSNs on the right. Overlapping areas represent shared knowledge space. Based on this view requirements can be formulated in two general ways:

One requirement catalog shared by all experts:

The requirement catalog is the area that is shared by all experts. Such a catalog can be based on WSN taxonomies as presented in Section 2.1.2, or in related work such as [TAGH02],[RM04],[RG08]. All of these classification schemes have in common that they do not only consider requirements and constraints, but also more technical properties of an actual deployment, like network size, network topology,

and heterogeneity. These properties are useful for classifying existing WSN solutions, but are unsuited for structuring the requirement analysis. As solution, in [OP10] we proposed a catalog consisting of five general categories and 27 criteria in total. The catalog is an assembly of criteria even non-experts can define for the intended sensor network.

Individual catalogs for each domain: An alternative concept is the application of domain-specific catalogs. Each catalog supports the language and expressions of the domain experts.

The technical realization of both approaches is basically identical since a translation to the technical WSN domain is required, anyway. Whether this translation originates from a general or individual catalog does not influence the actual development process. It is rather an organizational question which is important for potential end users. In the following we focus on a general catalog while it can be extended or replaced by individual catalogs without technical change.

5.3.3 Requirement Translation

The actual intelligence in the configKIT requirement definition process is the translation from the user terms to the technical language of WSNs.

The fundamental idea of this translation is to apply the relations of the PRG to derive alternative expressions of already defined requirements. Since this approach practically adds new requirements, it may be referred as *requirement expansion*.

For instance in ZebraNet [ZSLM04] the nodes are deployed to zebras. A fact from which a biologist can easily imply other requirements based on the behavior of zebras as herd animals, while that knowledge is not common for computer scientists. On the other side, WSN engineers eventually need technical definitions that are beyond what end-users have to know, for example the need for and parameters of congestion protocols in the transport layer. From the fact that zebras are herd animals we can deduce that many similar nodes are in range, so that the density of the network is rather high, which can be inferred to multi-hop, short distance communication requirements.

We apply this forwarding methodology for deducing requirement types of high abstraction to technical terms (as R1 to R7 in Figure 5.14), but also within the technical requirement space (as R6 to R9 in Figure 5.14). The latter is motivated by the fact that the same requirement can be expressed in different ways. It is our goal that all possible expressions of a requirement are available in the subsequent automatic composition process.

For example the user may express one requirement as period (for example in seconds), which without translation is not applicable to components processing a frequency parameter (e.g. 1/s).

The requirement space expansion phase will expand the given requirements to all requirements that can be expressed with the given parameters. This is done by a relation-formula as part of the requirement entries in the database. Thus, all expressible requirements in the system form a network (graph) that will be parametrized with few requirements given by the user.

Practically this structure is represented by the PRG, while all possible requirements are expressed as properties. The translation is realized by relations attached to the description of individual requirements properties, so that the requirements already contain information on how they relate to other requirements. This allows to add new requirement types can be easily added without interfering with existing structures. In the habitat monitoring example, if a domain engineer wants to add a requirement without using the catalog (as R1 in Figure 5.14), R1 could be the species of monitored animals, and the translations describe how the behavior of the animals affects network properties. As conclusion of this process we can define the data structure of the requirements:

Requirements Repository REQ is a PRG structure $\langle R, T, E \rangle$ of

- Requirements R which are expressed as properties,
- Translations T which are expressed as relations, and
- Edges E of the graph structure.

The undefined requirements in the repository are also named *requirement types*.

Requirement selection: This is the parametrized subset of the requirements. The REQ graph is not changed. Only the values of the specific requirements are defined and the implications are propagated through the graph. The technical background of this content-related propagation was described in Section 5.2.3.

Parametrization of the Working Model is performed by adding the PRG of the requirements to the PRG of the WM:

$$WM = WM + REQ$$

It adds all requirements and relations so that the extended requirement space is integral part of the working model.

5.4 Component Model and Structures

Before in the next section the actual selection algorithm is presented, this section introduces the fundamental component and composition model. The model consists of two general parts: first, the static representation of the components and their properties as they are stored in the repository, and second, the logic and rule-set to compose the individual components to a system. That is why after a general introduction of the applied component model, in the first part of this section the structure and the individual parts

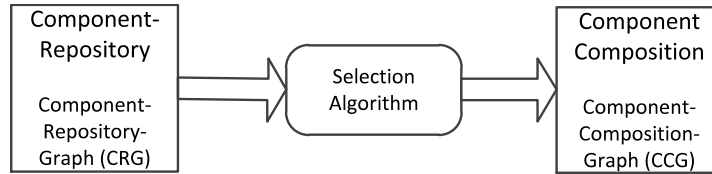


Figure 5.15: The Component Repository and the Component Composition are the major data structures containing components. Component Repository is a static structure that is used by the selection algorithm, the output is the component Composition.

of the used component model is defined and the required data structures are introduced. It concludes with a description of the component repository and its data structures. The component repository contains all components, their interfaces and properties as sort of a database. It is represented by the Component-Repository-Graph (CRG).

In the second part of this section the dynamic component composition graph (CCG) is introduced. The CCG represents the actual assembly of components and their binding to an implementable system.

As illustrated in Figure 5.15 the component repository is used by the selection algorithm, while the component composition is the output. The algorithm is described in Section 5.5. Both component-related data structures are introduced below.

5.4.1 Component Model

We use a high level abstraction for components, their interfaces and the properties of the components. Even though the model provides the abstraction from the actual implementation, it is not an entirely black-box model. As we will see below the model rather corresponds to a gray-box model, since the behavior of the components may be parametrized and extended as part of the composition process.

Components in configKIT are models of software, hardware or protocols and represent an individual behavior. Each component provides its function to other components via interfaces. Similarly, components may use other components via interfaces.

The behavior of the components is expressed by properties and relations. Therefore, each component contains a set of properties and relations (PRG structure introduced in Section 5.2) to express attributes, qualities, and supported functions of the component. Component constraints are expressed as relations.

Interfaces can be considered as ports of the components. Semantically interfaces in configKIT are service access points of the components and they implicitly represent a service description. This corresponds to standard semantics of interfaces, as introduced in Section 3.4.1. Notable is that we chose a rather high level of abstraction for the interfaces. This means that technical interfaces that may occur in a fine granularity are combined to larger abstract interfaces. For instance a message control component as described in [KHCL07] contains several detailed interfaces (send, receive, control, params) just to combine to a single other component. In such a case our model provides just one characteristic interface instead of explicit ports for sending, receiving and additional control. This single interface still represents that group of ports and functions and can be resolved for later integration. During composition and within the model, however, the additional abstraction improves the handling of the components significantly.

Abstract components and interfaces are abstractions of -usually of a set of- explicit components. Explicit components are actually implementable modules. Component abstraction inside the configKIT framework provides means to address classes of components without pointing on actual modules. By this, modules or users can express the need for a specific function without calling explicit components or interfaces.

For example an application can require a stream cipher module without knowledge of the actually available set of ciphers.

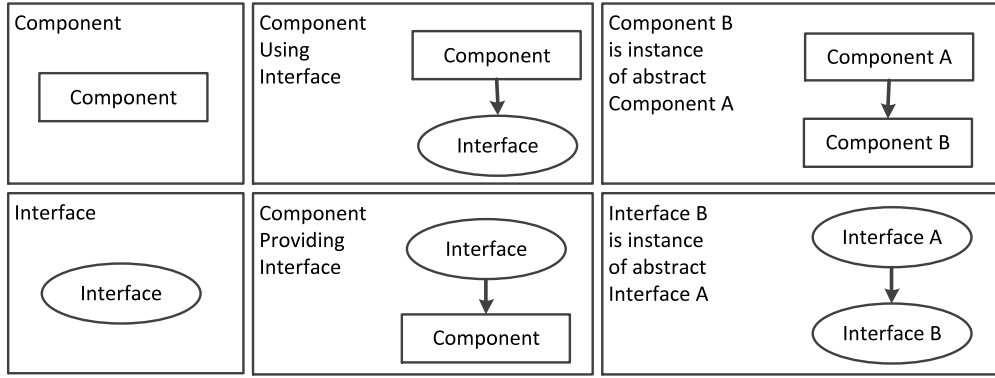


Figure 5.16: Graphical notation of static interface and components: Components are boxes, Interfaces are Ellipses. The arrow between Components and Interfaces indicates whether the interface is used or provided by the component. Arrows between two Components or Interfaces show instantiation.

As part of the framework not only modules but also interfaces can be abstract. Similar to abstract components abstract interfaces are generalizations of explicit interfaces. They allow to address a set (or class) of interfaces.

By this, functional aspects of a component may be defined in three ways:

1. Interface, e.g. the medium access control component sMAC provides the MAC interface.
2. 'Is' relations of the component, e.g. the sMAC is a MAC.
3. Property (evaluated in the PRG), i.e. the sMAC defines the Property 'MAC protocol'.

The three options are supported not only for flexibility reasons. Properties are the result of the requirement expansion. If it results in the need for a MAC protocol, the property has to be defined in the component accordingly. Other components that use the MAC typically specify the MAC interface. Therefore the interface should be defined. The abstract MAC component is not chosen directly, but it is valuable as blueprint for the specific MAC components.

Graphical Representation of Components

For graphical representation of components and their relations in this thesis we chose a notation that illustrates components as rectangles and interfaces as ellipses. The graphical notation is explained in Figure 5.16. Arrows indicate the relations between components and interfaces. The semantic of the arrows is as follows:

- Interface \rightarrow Component: the component provides the interface.
- Component \rightarrow Interface: the component uses the interface. (If dotted, the usage is optional)
- Component A \rightarrow Component B: A inherits from B (IS relation of components)
- Interface A \rightarrow Interface B: A inherits from B (IS relation of interfaces)

Figure 5.17 shows a practical example of the notation, illustrating the context of the TinyDSM component. Basically it provides the Tiny Middleware interface and uses

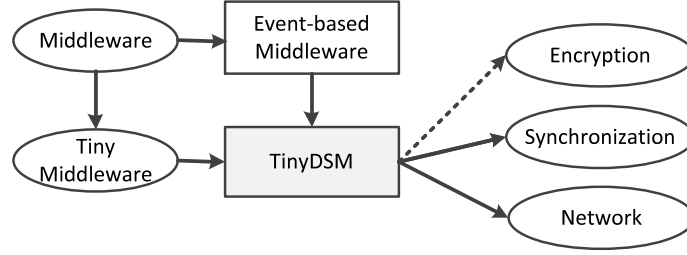


Figure 5.17: Example for the graphical notation of a single component: The TinyDSM component is an instance of the abstract Event-based Middleware component. It provides the tiny middleware interface and uses interfaces for network access and synchronization, plus optional Encryption. The Tiny Middleware interface is an instantiation of the general Middleware Interface.

interfaces for network access and synchronization. Additionally, it uses the Encryption interface optionally. TinyDSM is an instance of the Event-based Middleware which provides the general Middleware interface. This already means that TinyDSM provides the Middleware interface. This is emphasized by the definition that the Tiny Middleware interface is an instance of the Middleware interface.

5.4.2 Static Component Structure - The Component Repository

The component repository is a data structure similar to a database that combines all available components, interfaces and interrelations.

Formally the CRG is a triple $\langle I, C, T \rangle$, with

- Set of Interfaces I ,
- Set of Components C ,
- Set of Transitions $T : T = ((C \cup I) \times (C \cup I)) \rightarrow \{0, 1\}$.

It entails a cgraph structure $\langle (I \cup C), T \rangle$ (introduced in Section 5.2.2)

The transitions T are divided in four disjunct sub-groups T_{Ii} , T_{Ci} , T_{Cp} , T_{Cu} ;

$T = T_{Ii} \cup T_{Ci} \cup T_{Cp} \cup T_{Cu}$, while

- $T_{Ii} = (I \times I) \rightarrow \{0, 1\}$ is the subset of transitions expressing the Is-relations of interfaces. $(I_1, I_2) = 1$ means that I_2 is an instantiation of I_1 and that I_1 is an abstraction of I_2 .
- $T_{Ci} = (C \times C) \rightarrow \{0, 1\}$ is the subset of transitions expressing the Is-relations of components. $(C_1, C_2) = 1$ means that C_2 is an instantiation of C_1 and that C_1 is an abstraction of C_2 .
- $T_{Cu} = (C \times I) \rightarrow \{0, 1\}$ is the subset of transitions expressing used interfaces of components, $(C_1, I_1) = 1$ means that component C_1 uses interface I_1 .
- $T_{Cp} = (I \times C) \rightarrow \{0, 1\}$ is the subset of transitions expressing provided interfaces of components. $(I_1, C_1) = 1$ means that component C_1 provides interface I_1 .

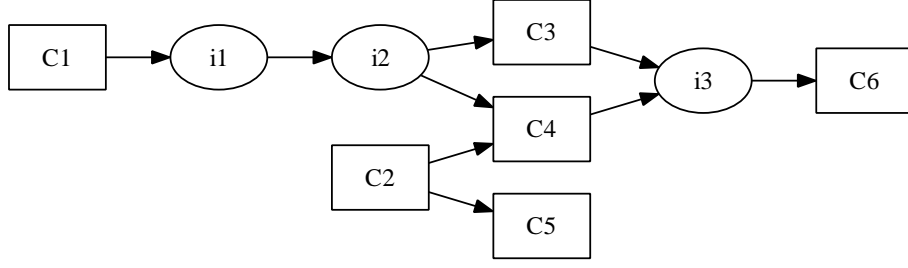


Figure 5.18: Example of a small Component Repository Graph (CRG) with symbolic components and interfaces.

The graphical representation –called component repository graph (CRG) – combines the notations of the single components while it merges interfaces and components of the same name. Structurally this means that all interfaces I are plotted as ellipses, all components C are shown as boxes. The transitions are plotted as arrows according to the rules explained in the previous subsection.

Please note, the graphical representation of component A uses interface X is provided by component B does not mean that component A and component B are practically connected. It rather means that they can be connected.

Figure 5.18 shows an example CRG structure for a set of components and interfaces. For brevity we use initials instead of actual component names. Even though the structure in the figure contains all sorts of transitions, it is not completely synthetic. As example it can be assumed that C1 is an application component that uses encryption (i1) which is instantiated by symmetric encryption (i2). C3 and C4 are symmetric block ciphers that both need a stream cipher (interface i3 and component M6). M2 can be generalization of a specific encryption standard and M5 is a specific stand-alone implementation. For this example, with the proposed CRG structure the following sets can be derived:

- if C1 is selected, the selectable components are $\{C1\} \times T^* \cap C = \{C3, C4, C6\}$
- components that may use interface i3: $\{i3\} \times T_i^{-*} \times T_{Cu}^{-1} = \{C3, C4\}$. The first step ($\times T_i^{-*}$) is required to compute all interfaces from whose i3 is inherited.
- components that provide interface i1: $\{i1\} \times T_i^* \times T_{Cp}^1 = \{C3, C4\}$.
- components that may (directly or indirectly) use C6:
 $\{C6\} \times T_i^{-*} \cap C = \{C1, C2, C3, C4\}$

The example shows how the data structure helps filtering useful components and interfaces in the composition process. For instance in the example C5 is practically isolated even though it is an instance of C2. The fact is visually observable in the CRG, as well as computable in the CRG structure.

Data Model

As conclusion of description of the static component model, Figure 5.19 shows the static meta-model of the component repository and hence illustrates the relations between the entities: They are described in the following:

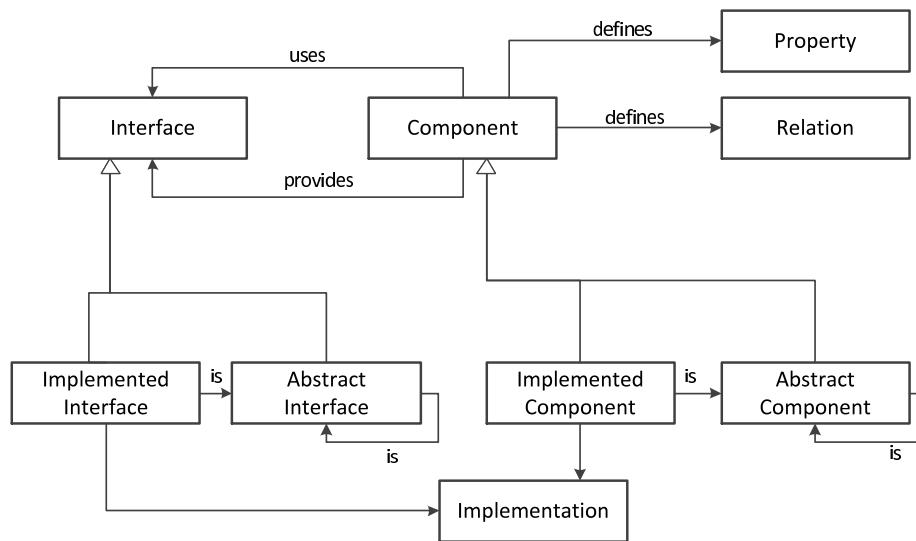


Figure 5.19: Meta-Model of the static interfaces and components.

Components and Interfaces are the core relationship in the component database. A component can use any number of interfaces and provide any number of interfaces. An interface can be used by any number of components and can be provided by any number of components.

Components can be either implementable components or abstract components.

- Implementable components may be associated with actual source code.
- Abstract components are not implementable but provide a skeleton of properties and associations to components that inherit these properties. Each component can inherit from any number of abstract components (via the 'is' relation). This means also abstract components can inherit from other abstract components.

Components can have any number of properties and relations.

Interfaces can be either implementable interfaces or abstract interfaces.

- Implementable interfaces may be associated with actual source code.
- Abstract interfaces are not implementable but provide a logical skeleton to interfaces which inherit these properties. Each interface can inherit from any number of abstract components (via the 'is' relation).

In the following this data structure is the basis the composition process. There, additionally to the selected components also their interfaces, properties and relations – as they are connected to the components – have to be considered in order to assemble a correct system.

5.4.3 Dynamic Data Structures - The Component Composition Graph

The Component Composition Graph is the model of the system under development. It is the composed application consisting of components. The goal within the CCG is to find

a composition that is complete and can be integrated. In this subsection the structure and the rules for the CCG are introduced.

Formally the CCG is a triple CCG: $\langle\langle C, I, T \rangle\rangle$, with a structure similar to the CRG.

- I is the instantiated set of Interfaces,
- C is the instantiated set of Components,
- $T : T = ((C \times I) \cup (I \times C)) \rightarrow \{0, 1\}$.
 $\forall i \in I : T = ((C \times I) \cup (I \times C)) \rightarrow \{0, 1\}$.

It entails a cgraph structure cgraph: $\langle\langle I \cup C, T \rangle\rangle$.

Contrary to the CRG no abstract interfaces or components are supported. The graph directly describes which components are bound using which interfaces.

For instance for two components $c1$ and $c2$: $c1, c2 \in C$ and an interface $i1 \in I$, the transitions $(c1, i1) = 1$ and $(i1, c2) = 1$ implicate that $c1$ uses $i1$ and that $i1$ is provided by $c2$, meaning that $c1$ uses component $c2$, so that $c1$ and $c2$ are bound via interface $i1$.

Composition Rules

The following description lists the structural composition rules for the components:

Each interface must be provided by not more than one component, otherwise it is not possible to bind the using and providing component.

Interfaces can be used by more than one component. Unless defined differently in the component properties, each provided interface may be used by a multitude of components. Exceptions may be stateful components. For example a checksum function for a data stream may store intermediate checksums internally which forbids multiple usage of the component.

Components can be instantiated more than once. As answer to the issue of stateful components, components may be instantiated more than once. Practically it is realized by attaching a number to indicate the instance. For example if the system contains two stateful checksum functions: one for incoming and one for outgoing traffic, the first instance is named 'checksum.1' and the second is 'checksum.2'. The same has to be done for the provided interfaces.

CCG is no tree: Often it is assumed a composition graph is a tree with the system as root and services as leaves. However, in the configKIT component model reconvergencies are possible (see case a) or c) in Figure 5.20), and even intended, since reusing a component multiple times promises to reduce the total code size of the system.

Interface consumption: After binding a component that provides an interface used by another component the composition of both components still provides the interfaces. The using side of the interface is consumed. This is a result of the provide-once-use-many-times paradigm in the composition process.

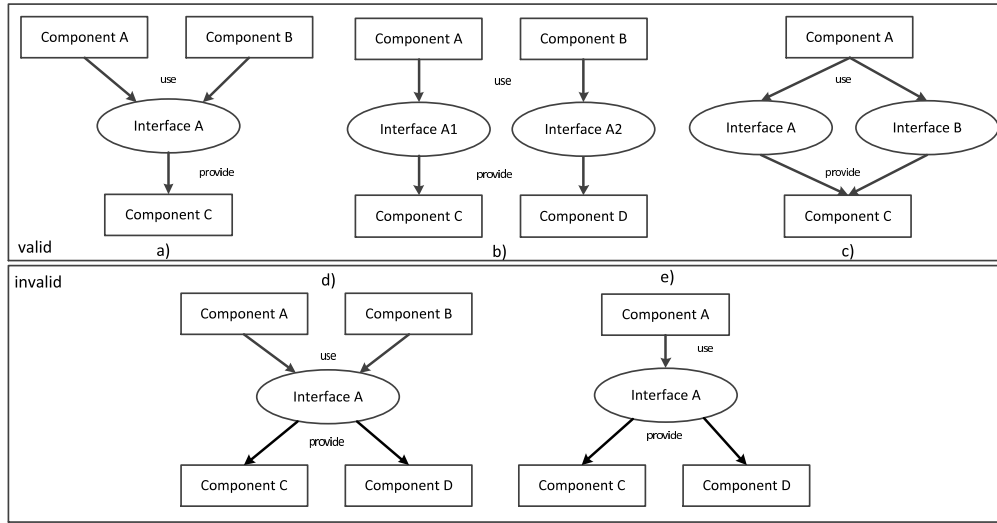


Figure 5.20: Examples for valid and invalid component-interface-connections.

Additional constraints emerge when respecting the behavioral aspects of the components. They are discussed in Section 5.4.4.

Figure 5.20 shows several examples for valid and invalid component-interface-connections in the CCG. The five cases are briefly discussed below:

- a) it is allowed that one interface that is provided by one component can be used by several components
- b) it is allowed that two instances of the same interface can be used by different components. That may be required if the component that provides the interface can only handle the states of being used by one component. Examples are stateful encryption units for incoming and outgoing data streams.
- c) One component can use more than one interface - and one component can provide more than one interface. The specific case where two components share two interfaces may be applied for complex interfaces that are split to improve reusability.
- d) and e) It is not allowed that one instance of an interface is provided by more than one component. The number of components that use the interface is irrelevant. A valid alternative would be setup b)

Structurally Complete Compositions

With the rule set defined in the previous paragraphs it is possible to compose systems consisting of components, while it is still unclear under which conditions a composition is complete, i.e. implementable. Basically, a composition can be considered as complete if there are no open dependencies.

Analog to the properties listed in [JDG10] the CCG has to comply with the following requirements:

Reachability: all components of the CCG have to be reachable from the main system component: $\forall c \in C_C : c \in (\text{'system'} \times T^*)$. 'System' is the name of the top

component. This requirement also ensures cohesion of the composition, because interfaces are the only means to reach other components.

Extensiveness: all components given by the user have to be in the composition: $\forall c \in App : c \in C_C$.

Completeness: All required interfaces of all components in CCG have to be satisfied by at least one component in the selection:
 $\forall c \in C_C : (c \times T) \subseteq C_C \times T^{-1}$

Implementability: All components of the composition must not be abstract but implementable.

Only if those four conditions are satisfied, the composition is complete and hence implementable. Beside the implementability, which is an attribute of the components, all rules can be computed within the CCG, which simplifies the check on completeness in practice. However not all syntactically complete compositions are free of contextual conflicts. Therefore the properties of the components have to be evaluated. This process is introduced in the following section.

5.4.4 A-priori Reasoning about Composition Properties

Propagation of semantic attributes of a composition from the CCG to the PRG is the operation of adding properties and relations associated with the corresponding component to the PRG.

In this section we first discuss general methods of mapping the properties from the CCG to the PRG. In the second part property models are introduced. The property models are needed to assess specific attributes of the systems on a rather high level before actually implementing the system.

Property Propagation

As introduced earlier, the behavior and attributes of the system are assessed by mapping properties from the selected components to the PRG. Initially, this mapping could be expected as direct projection of features, so that

$$properties(composition) = \bigcup_{c \in composition} properties(c) \quad (5.1)$$

However, not all properties can be mapped directly. Properties may also be defined as deduction of the composition. A third sort of properties describe structural properties and dependencies of the composition. Thus, the property assessment can be described with the non-trivial mapping function

$$properties(composition) = mapping(binding, composition). \quad (5.2)$$

This means that the eventual properties of a composition not only depend on the chosen components but also on the binding within the composition. It reflects that the same selection of components may result in different behavior if the components are connected differently. It also implies that for the same selection and the same binding we can expect the same properties.

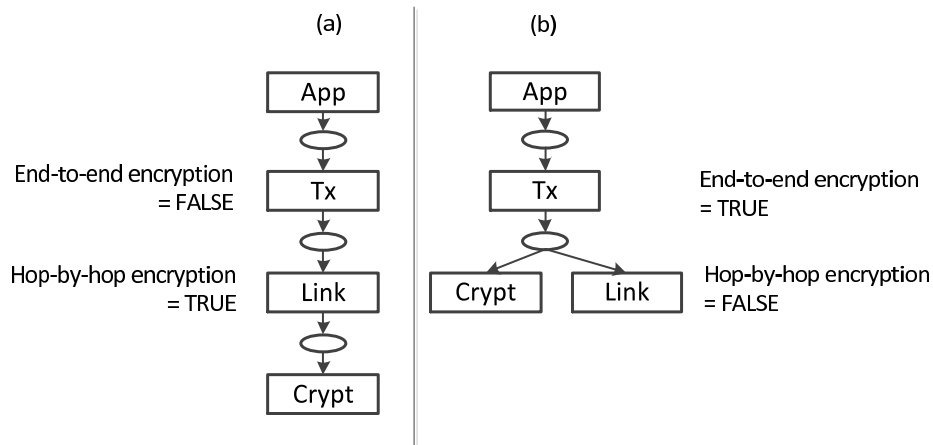


Figure 5.21: Example for direct properties: both compositions consist of the same modules (Application, Transport, Link Layer, and Encryption). Still the system properties are different. In (a) the TX cannot access the security services so that no end-to-end security can be provided, in (b) the Link layer has no crypto function to establish hop-by-hop encryption.

From a practical point of view it has to be differentiated between direct and derived properties:

Direct Properties are properties which are valid on the scope of the entire system once the corresponding component has been added. Many functional properties are valid on global scope because once the component is part of the system also the function is available. Thus, global properties are valid regardless of the actual binding of the component, so that Equation 5.1 holds true.

An example for global properties is the radio frequency which is set for the whole application once the radio transceiver is selected. This global parameter is the basis for many other properties in the system, but it is not affected by other components. Another group of examples for direct properties are direct function descriptions, such as the function Event-notification which is available if the corresponding component is included.

Derived Properties are influenced by the composition of the system and cannot be expressed directly but need a mapping in the context of the binding. For example the quality attributes of a network transport protocol depend on the qualities of the used network protocol.

As another example encryption on link layer does not provide the level of security as encryption on transport layer. Figure 5.21 illustrates both system architectures. Both compositions consist of the same modules (Application, Transport, Link Layer, and Encryption). Still the system properties are different. In Figure 5.21 (a) the TX cannot access the security services so that no end-to-end security can be provided, in Figure 5.21 (b) the Link layer has no crypto function to establish hop-by-hop encryption. So the same encryption component unfolds different system properties depending on the usage in the system stack.

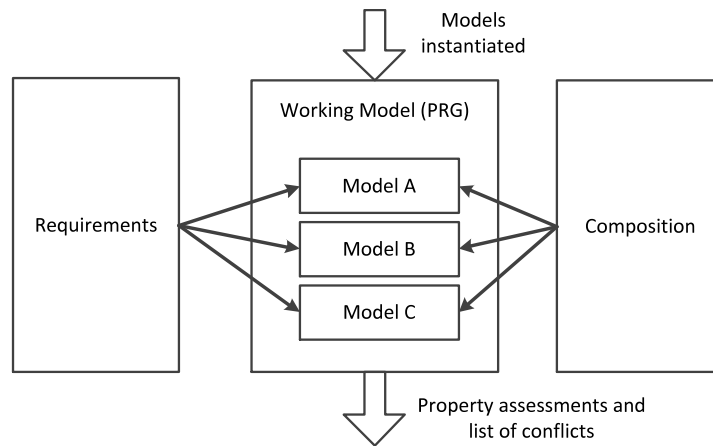


Figure 5.22: Property Models as Part of the Working Model: In the PRG of the WM sub-frameworks of properties and relations can be loaded to represent a specific property model. Like the global PRG the models are parametrized by requirements and evaluate the composition.

Such properties may be injected in the interfaces of the components. For instance if the transport protocol uses the encryption component the qualities of the cryptographic module are visible for the transport protocol via the used interface. Then the transport protocol may define the global property 'end-to-end-security' with a specific quality. If the cryptographic component is used only by the data link layer, the global property 'end-to-end-security' has to be set to 'none'.

5.4.5 Property Models

While some technical terms can be described directly in an ad-hoc manner, most properties are rather complex and need additional modeling and a general rule set. Such rule set determines how the properties have to be specified in the components description, how the requirements of these properties are formulated and how the information is processed. Within the configKIT framework various property models can be expressed within the PRG structure.

Figure 5.22 illustrates the principle: In the PRG of the WM sub-frameworks of properties and relations can be loaded to represent a specific property model. Like the global PRG, the sub-models are parametrized by the requirements and assess the composition based on the properties in the components and their binding. This approach allows to load specific more detailed models to assess properties of specific sorts of networks or applications.

In the following assessment models for general aspects for the development in the domain are discussed.

Memory: memory has to be differentiated between RAM and ROM. In particular RAM is severely constraint on most WSN nodes, while usually sufficient ROM in the flash memory is available. Another problem is that memory consumption differs with the applied hardware. A module compiled for the mica node will not have the same size as for an MSP430. Compiler optimizations or other optimization-effects

caused by the connection of the modules can additionally affect the result.

In the evaluation phase we use the straightforward addition of module sizes. We know that the result is not perfectly precise but is a good estimation that helps to assess the memory consumption. Consequently the memory property is a set of two numbers, i.e. RAM and ROM, that describe the module size.

Energy Consumption: Actually, energy can be measured and estimated precisely. For the applied hardware the needed energy per computation cycle is available. To determine the energy consumption, this can be multiplied with the number of cycles needed for one operation and the number of operations per time unit. This data can be gathered by simulations. The results, stored in the individual components, would allow to assess energy precisely. However, often such reliable data is not available and usually the actual number of duty cycles depends on the application scenario. It is considerable that in future work reliable energy models will appear for configKIT, but for now we follow an alternative approach.

In a this approach the energy consumption is assessed on a qualitative scale. With such a qualitative metric, i.e. '0' for bad and '3' for very good, it is easier to assess and also combine different protocols, and it is possible to compare similar protocols. Clearly, that approach is not perfectly satisfying, but it can help to filter suitable or non-suitable modules.

Expected Lifetime: The expected lifetime depends on many factors such as energy consumption, available energy, data distribution patterns. Assumed these input parameters are available it is a trivial operation within configKIT to determine the expected lifetime. Currently precise values for the energy consumption are missing.

Money: The price of the network is one important attribute for WSNs. for many tasks there is a trade-off between many small nodes or few larger nodes. With the knowledge of the single components that can be easily represented within a configKIT model. Variable and fixed costs for components can be stored in the meta-information, so that the money model only needs to accumulate the corresponding costs.

Security Degree: Since the gravity center of this thesis is about security, the separate Chapter 6 is dedicated to this issue of security models.

It is apparent that many properties need relatively complex models. While the small models presented in this section are rather brief impressions, the capability of the property models will be shown in Chapter 6 where several security models will be integrated with the presented structure.

5.5 Selection Algorithm

The objective of the composition approach is to find an assembly of components that satisfies all constraints in a model of the real world which is parametrized by the users' requirements. The process is illustrated in Figure 5.23. The selection process has the requirements and the component repository as input and computes a set of configurations that can be integrated and eventually deployed. The steps to realize this task are the following:

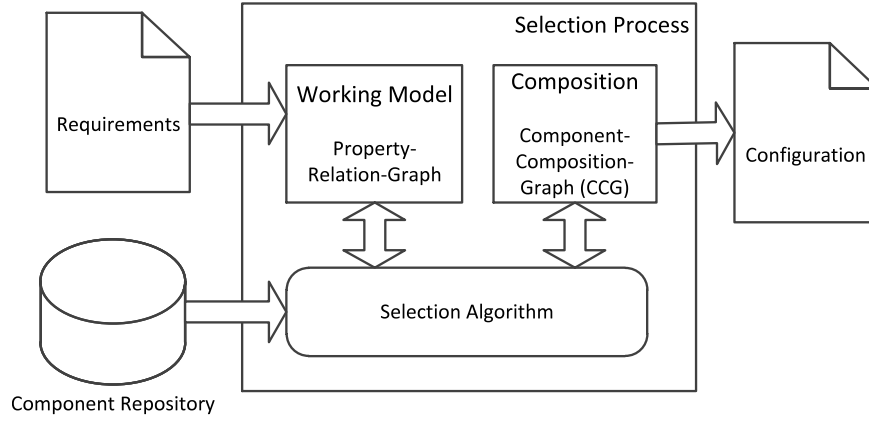


Figure 5.23: Structure of the component selection process: Requirements, loaded in the Working Model have to be satisfied. The selection algorithm uses the component repository to search a system composition whose properties solve the constraints in the WM. If such a composition is found it is stored as configuration of an implementable system.

1. The PRG in the working model is pre-loaded with the properties and relations of the property repository.
2. Requirements are loaded in the WM. They parametrize the WM. After this step the WM contains a set of unsatisfied constraints.
3. The composition, which represents the system under development, is initialized with an empty application.
4. The selection algorithm uses the content of the component repository to compose a component composition whose properties solve the unsatisfied constraints in the Working Model.
5. If a satisfying composition is found it is forwarded as system configuration to further integration.

This section focuses on the third and fourth point: about the component composition and the selection algorithm. Item one and two were discussed in the previous sections.

5.5.1 Initial Problem

The input data for the selection process are:

- Component Repository: $\text{CRG}\langle C_R, I_R, T \rangle$ with the available components C_R , the interfaces I_R and the edges T with its subsets T_{Ii} , T_{Ci} , T_{Cp} , T_{Cu} ; describing interface and component abstractions, provides and uses relations as introduced in the CRG definition.
- Each component $c \in C_R$ contains:
 - component description including interfaces, expressing the structure of the components,
 - Properties and relations to describe the semantics of the components.
- Working Model as $\text{PRG}\langle P, R, E \rangle$ with properties P , relations R and edges of the graph E .

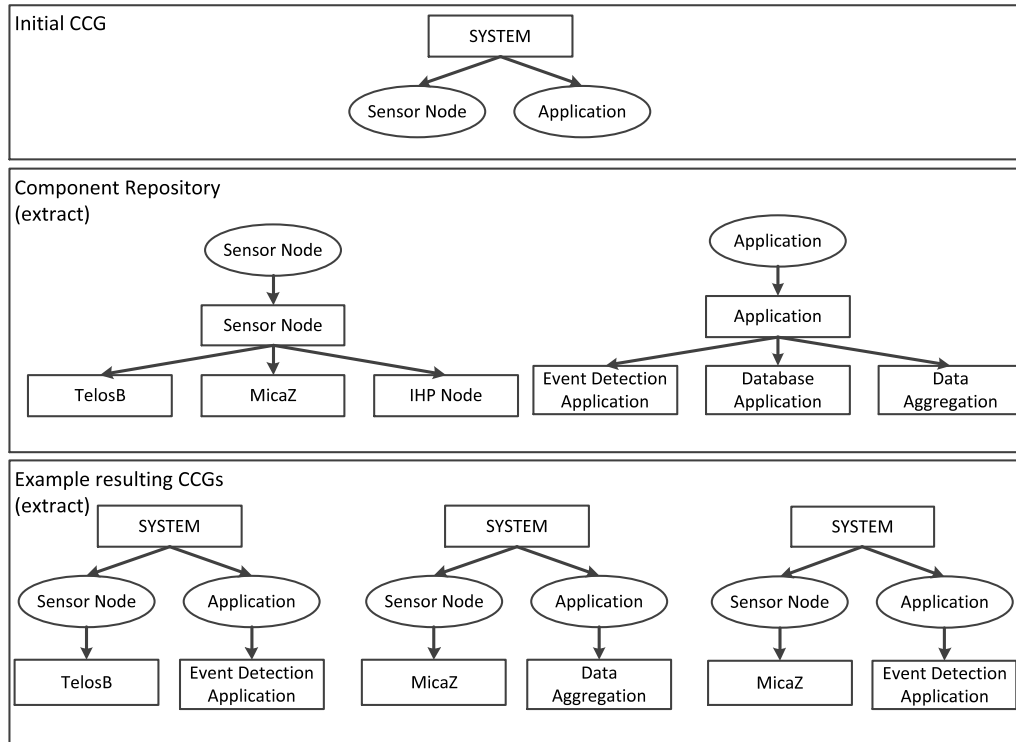


Figure 5.24: Top level system composition: The initial CCG contains the System component that needs Sensor Nodes and an Application. The Component Repository contains a set of components that are instances of abstract Sensor Nodes or Applications which provide the corresponding interface. The bottom box shows example compositions. The additional dependencies and properties of the components are ignored in this example.

- with models loaded to assess system properties,
- initialized with the requirements.

The resulting data structure is the

- Component Composition Graph: $CCG\langle C_C, I_C, T_C \rangle$ with the instantiated components C_C , the corresponding interfaces I_C and the binding between interfaces and components B .

The problem of the selection algorithm is to build a system consisting of software and hardware starting from a core system in order to satisfy the requirements. Figure 5.24 illustrates the process: The initial CCG contains the System component that needs Sensor Nodes and an Application. The Component Repository contains a set of components satisfying the requirements. Connecting those components to the CCG results in many potential compositions. From this set valid compositions form the set of applicable configurations.

A composition is valid if:

1. The composition is complete, and
2. There are no conflicts in the WM.

The completeness of a CCG is discussed in Section 5.4.3. Basically, a composition can be considered as complete if there are no open dependencies and it contains no unresolved abstract components.

The conflicts are an output of the PRG as explained in Section 5.2

With these two conditions we can rephrase the component composition mapping paradigm from the functional

$$f(\text{Requirements}, \text{Repository}) \rightarrow \text{Composition}$$

to a search-based

$$\forall \text{Composition} \in \text{Repository} : \text{Composition} \stackrel{?}{=} \text{Requirements},$$

which means that we search for all compositions in the that can be assembled using the components in the repository and which implement the system requirements.

5.5.2 Algorithms

In the following, three iterations of the algorithm to solve the problem is presented. The three versions are described on a high level to illustrate the operation of the search process. A more detailed representation of the implemented recursive backtracking algorithm is shown at the end of this section.

Algorithm 1: Straightforward Search

The first version of the algorithm follows the fundamental notion of model based composition. That algorithm has never been implemented due to its obvious inefficiency. However, it provides a first insight in the general practical property propagation and assessment.

1. For all compositions c
2. If c is structurally complete
3. Assess the effect of the Composition properties in the WM
4. If it solves all conflicts in the WM
5. A suitable composition has been found

The realization of this algorithm is straightforward: The CCG generates the component compositions. The properties and relations of the compositions are forwarded to the PRG. In the PRG the validity of the composition is assessed and a binary answer is returned to the CCG.

Algorithm 2: Respecting the Structural Dependencies of the System

The inefficiency of Algorithm 1 is caused by the fact that the search iterates through all possible compositions regardless of the actual structure of the composition. This algorithm can be optimized by exploiting the reachability requirement of the CCG. This means only components are considered that may be attached to the system under development.

The adapted algorithm is:

1. While composition C not complete
2. Find a component c in the Repository that may be attached to C .
3. Add c to the composition C
4. If c is structurally complete
5. Assess the effect of the composition properties in the WM
6. If it solves all conflicts in the WM
7. A suitable composition has been found
8. If no c could be found: remove last added component and continue search (backtrack)

This algorithm can be implemented by disabling all components for selection at the beginning. Components are only selectable if they solve a specific need.

From structural point of view, the only open demand can be either open interfaces or abstract components without implementation. So at any point in the composition process we maintain the sets of open interfaces I_O or abstract components C_O .

These lists already allow the implementation of an optimization. After adding a component c , with the mandatory interfaces I_c used by c ($I_c = c \times T_{cu}$), the solvability of i can be checked by verifying that $\forall i \in I_c : i \times T_i^* \times T_{cp} \neq \emptyset$. If for one interface no component can be found, it is clear that c can never be completely implemented in the current context of the application. A similar check can be performed on open components.

With the knowledge that the current composition with component c is not implementable the algorithm can backtrack immediately.

While, generally, we follow a depth-first backtracking, the completability check is a one-level breadth-first test. It controls the search space for the entire subtree, and thus identifies non-solvable configurations before starting a deep backtrack. This operation is an extension of step 2 of the algorithm.

Algorithm 3: Respecting the Semantic Dependencies

The first two versions of the algorithm checked the properties of the system only after a composition was complete. This is not the best solution since it can be assumed that many configurations and components can be ruled out early due to behavioral incompatibilities.

For instance a cryptographic implementation that uses assembler code to improve the performance only runs on specific microcontrollers. If the application demands a different microcontroller, the conflicting cryptographic component can be ruled out for the entire selection process. We avoid such a dedicated filtering process. Instead, we introduce a concept to identify such static conflicts automatically.

The basic idea is to update the PRG of the WM after every component that has been added to the CCG. The new component may cause conflicts in the PRG. A conflict is a violation of a constraint. Such conflicts may be static or resolvable.

Resolvable conflicts are constraint relations of the PRG which are false in the moment of evaluation, while they may change in further configuration process.

Static conflicts are constraint violations in the PRG which cannot be solved with additional components or settings.

A conflict f is static if:

$$\forall n \in (f \times E^{-*}) : (n \times E^{-1} \neq \emptyset) \text{ OR } (n \text{ is defined}),$$

meaning that for all nodes n of the PRG for which the constraint f depends on, either n depends on other nodes or the node is defined (either by a requirement or a component property). If that is the case, f cannot be resolved in the future.

Practically, it can be solved by forwarding a 'static' flag in the PRG. A property is static if it cannot be changed in the further process. Consequently, the result of a relation with all inputs defined as static is static too.

As result the main iteration of the algorithm is changed as follows:

1. While composition C not complete
2. Find a component c in the Repository that may be attached to C .
3. Add c to the composition C
4. Assess the effect of the composition properties in the WM
5. If C causes static conflicts
6. remove c and backtrack
7. If c is structurally complete
8. If it solves all conflicts in the WM
9. A suitable composition has been found
10. If no c could be found: remove last added component and continue search (backtrack)

This algorithm adds the steps 4 to 6 to assess the semantics of the composition, while the rest of the algorithm is still equivalent to algorithm version 2. Also this version can be optimized by the breadth-first test during step 2. With the additional coverage of semantic attributes in the WM it immediately filters out all components not suitable for the active system. For example the cryptographic implementation that conflicts the pre-selected hardware will be identified during the first iteration and never be considered again.

This test can be exploited also to gather which component may affect the PRG the most. Naturally we want to integrate components first that affect many aspects of the system instead of peripheral services. This optimization is respected in the actually implemented algorithm as it is presented in the following.

Final Selection Algorithm

The third version of the algorithm finally was implemented as recursive backtracking algorithm. Its core is a recursive function to add a component to a given interface. The function updates the PRG and the CCG, identifies complete composition, detects conflicts and iterates through suitable components to complete the composition.

It is presumed that the backtracking from the function restores the content of all variables and structures, including PRG and CCG.

Beside the component repository, the algorithm maintains two sets of selectable components:

SC - Selectable Components is a subset of the component repository containing only the components which may be used for the composition. This set contains all components that are not ruled out due to conflicts.

AC - Active Components is a subset of the SC, containing only the components that can be attached to the current system under development. If a new component is added to the system, other components may be added to the AC (due to new needed interfaces) or removed from the AC (since interfaces are satisfied by the new component).

function bool recurs_add (component c , interface i , PRG, CCG)

1. Add component to CCG and PRG and check conflicts
 - if $\text{PRG.conflicts} = 0$ then solution_found (CCG), return true
 - if $\text{PRG.static_conflicts} > 0$ then return false
2. update actively selectable set of components and open interfaces
 - add all components in the selectable set (SC) that provide one interface usable by c
 - $\forall c2 : c2 \in SC \wedge (c \times T_{CU} \times T_I^*) \cap (c2 \times T_{CP}^{-1}) \neq \emptyset : \text{AC.add}(c2)$
 - test all elements in AC on compatibility to current configuration, and remove conflicting components
 - if $\forall c2 \in AC : \text{PRG.compatible}(c2.\text{PRG}) = \text{false}$ or $\text{CCG.compatible}(c2.\text{PRG}) = \text{false}$ then
 - SC.delete ($c2$); AC.delete ($c2$);
 - else record the number of changed properties for the interfaces of all $c2$
 - if the remaining active set of components is not sufficient to satisfy the open interfaces: backtrack
 - return false
3. if c is a non-implementable component look for implementations
 - function *replace* removes $c2$ and calls recurs_add to add c
 - replace ($c2, c$)
4. select interface i with the highest average number of changes in the PRG
 - test set TS
 - For $\forall c2 \in TS$:
 - recurs_add($c2, i$)

The algorithm has several notable properties:

Resolves abstract interfaces: Abstract interfaces are resolved in the first substep of Step 2. The equation $c2 \in AC \wedge (c \times T_{CU} \times T_I^*) \cap (c2 \times T_{CP}^{-1}) \neq \emptyset$ addresses all components $c2$ from the set of selectable components which provide at least one interface that is compatible to a used interface of the new component c .

Resolves abstract components in step 3. There the abstract component is replaced by the corresponding implementation.

Evaluates the properties of the composition immediately. Therefore it is applying the PRG with its complex models.

Discovers and evaluates conflicts early: In Step 1 conflicts with the CCG and PRG are evaluated by adding the new component to the graphs. After adding they report the conflicts.

Additionally, within Step 2 all selectable components are tested on their compatibility. In case of incompatibility the latest added component will be removed from the set of selectable components (SC), which immediately reduces the search space.

Maintains only selectable components that can be added in the data structure AC and SC.

Terminates: In step 1, if a configuration with no open structural dependencies ($I_O = C_O = \emptyset$) and no semantic conflicts ($\text{PRG.conflicts} > 0$) is found, the configuration is stored within the `solution_found` subroutine.

5.5.3 Complexity and Optimizations

The presented algorithm works while the run-time with a large number of modules can be unsatisfying. The major reason is that the arbitrary composition process will always be a non-polynomial (NP) problem. It can be simply proofed by the fact that if a component needs n other components, and for each other component we have x alternatives with same parameters, so that no component can be eliminated, then we have x^n alternatives with equal quality that all have to be evaluated. Therefore it is the challenge for the selection algorithm to reduce the practical complexity by eliminating non-suitable design options early in the selection phase.

Basically there are three general option on how to reduce the design alternatives

- Removing components from the design space that do not fit due to the structure or due to conflicts,
- Remembering previous decisions in scope of the composition,
- Reasoning to shortcut the search.

Removing Components from the Design Space can be done depends on the context. If a static conflict is caught after adding one component it is known that the component is not suitable now or later in this tree or in the sub trees of this nodes siblings. This means it is favorable to test all possible components first on static conflicts before starting the deep backtracking. On the first level it means that all conflicts to pre-selected hardware and operating environment would be detected before starting the first backtrack. In deeper levels the context is extended to the environment and the current configuration. This is reasonable since on this level the active configuration in fact is part of the environment all new components have to fit in.

This optimization is included in the algorithm.

Remembering previous decisions applies the idea that if it is known that a specific path in the search process results in a favorable configuration, the results can be reapplied in similar searches. Similar searches are search problems for which for the currently considered open interface i all structural and semantical dependencies of i and its subgraph are identical to the previous search. This means that the decision has to be always stored

in context of the current application. For the assumed interface i , first the set of dependent components and properties (D) has to be computed that may be influenced by the configuration of i . For this set D the set of nodes in the graph can be computed that directly influence D , which is the context. If the CRG were a tree this context set contains exactly i . It implies the search for the subtree will always result in the same result. If D depends on a larger set of interfaces and properties, a previously stored decision must only be reused if the status of the entire perimeter is the same. We know that the CRG is no tree. However, the CRG contains natural clustering in the repository which favors the approach. For instance network protocols and specific service protocols have a reduced coupling to other sorts components. So once the search for the corresponding subgraph (interface) has been computed the chances are high that the result can be reused.

Reasoning In case the goal of the search is to find one solution which is not necessarily the best one, reasoning is an option to reduce the search efforts. This means, if a specific problem set can be identified in the PRG a direct rule points out components or abstractions of components, which solve the problem. This approach corresponds to the a human way of development. Based on personal experiences and a given requirement situation preferred approaches are selected while other methods are ignored. This is why reasoning has to be applied carefully since it bears the risk to reduce the design space without technical justification. For instance adding to the PRG rules such as

- radio + robustness \rightarrow Bluetooth radio,
- mesh network \rightarrow Zigbee radio,
- or high security \rightarrow AES

may lead to correct solutions faster, but they restrain less prominent protocols. That is why reasoning –even though technically supported– is not actively fostered as part of the process.

Related Work

SNACK [GKE04] (also see Section 3.1.2) met the same problem in their WSN composition construction kit. The authors noted that the design search space exploded if a trivial design space expansion is used. They reduced it to a manageable size by eliminating incorrect expansions:

- The compiler maintains a list of pairs of components that can never be shared, for example due to incompatible parameter constraints, incompatible interfaces, or contradicting rights.
- the compiler memorizes the smallest valid expansion subset to avoid repetitive work
- if expansions are invalid and the reason for the fail can be clearly detected, configurations with the same problem will not be pursued.

In the example described in [GKE04] by this means the design space could be reduced from 10^{36} combinations to merely 28.

[JDG10] studied the search space and the complexity of component dependency resolution. Therefore they provided a general but clear definition of the component dependency and composition problem, which concluded in a dependency graph structure.

Based on required properties of the target graph structure, such as reachability, completeness and cohesion, they could reduce the potentially huge design space by two orders of magnitude.

Typical composition approaches such as FABRIC [Pfi07] add logical layers in the composition process. Each layer is composed and parametrized individually. It reduces the total complexity to $\sum_{n=1}^i c(L_n)$ for n layers and an assumed complexity function c for the layer L_n . That is, the complexity of composing the application is the summation of the complexity of the individual layers. Considered that the complexity for each layer is linear, i.e. one component is picked from a set of design alternatives, the total complexity is linearly as well.

If the selection of one layer influences the selection of the other layers -as it is the case in the configKIT approach- the complexity is determined by the product of the layer complexities $\prod_{n=1}^i c(L_n)$. In configKIT a layer can be considered as all components that provide a specific interface. Thus, the number of layers correspond to the number of interfaces.

That means, the number of design alternatives is determined by an exponentiation while the power is the number of interfaces, and the basis is the number of alternative components. It also means to reduce the complexity one has to reduce the number of interfaces or significantly reduce the number of components for each interface. This conclusion also holds in practice since naturally the number of design alternatives is rather small if

- we have components with highly specialized interfaces. Then there are simply no design choices because most components do not work with most other components.
- we have few interfaces. Then the number of design decisions is reduced.

5.5.4 Conclusions

The previous subsections discussed several approaches to reduce the complexity of the search. Nevertheless, The worst case complexity remains $O(c^i)$, while c is the number of components and i is the number of interfaces. It is non-polynomial, and hence not a preferable solution. However in practice the complexity is acceptable. Reasons are:

Only reachable components and interfaces are considered: The complexity c^i relates to components and interfaces composeable for the system. Since we only look for components for which interfaces are needed by the current system, i is not determined by the interfaces in the repository but by the usable interfaces in the potential system.

High interface abstraction: The component model used in the configKIT approach applies rather high abstraction for interfaces (compare Section 5.4). This reduces the number of interfaces and thus limits the exponent in the complexity.

Eventual WSN systems have rather low complexity: While the design space is complex, the eventual WSN systems contain relatively few components (typically 10-20). This limits the total complexity of the search since only reachable components are considered.

Early disabling of components that are not suitable for the system: The applied algorithm uses structural as well as behavioral assessments to reduce the size of the design space.

Structure of components in repository is not worst case: In a typical component repository for WSNs components are not randomly connected but rather clustered. It allows to reuse decision made for one cluster independently from others which limits the exponentiation.

This justifies that for the current real-world systems consisting of 50 components and nearly 30 interfaces the computation of suitable systems takes few milliseconds on Pentium IV class processor. Synthetic worst case tests, however, let the computation time explode (>1 minute) with less than 20 components in the repository.

For the case that in future the complexity also increases significantly for real-world systems, above approaches were discussed to limit the number of design choices by restructuring the repository or by adding artificial reasoning. From the algorithmic side heuristic and fuzzy search algorithms, ranging from simulated annealing [KGV83] to genetic optimization, are considerable.

5.6 Practical Tool Chain and Implementation Details

This section first describes how the algorithm presented in the previous section has been implemented. The selection algorithm is only one part of a prototypical tool chain which is introduced in the second half of this section.

5.6.1 Selection Algorithm - Implementation Details

This section provides information on specific implementation details of the current version of ConfigKIT selection algorithm. It is not meant to be a complete technical documentation. Rather it explains how crucial design decisions have been implemented. The description is supposed to support understanding and possible reproduction of behavior and results of the program.

The selection algorithm has been implemented in the C++ programming language. In the current version it is a command line tool loading the requirements given by the user as well as the repositories. It executes the requirement expansion, the management of the PRG/working model and the actual selection algorithm. The results will be written in various files which can be processed by user interfaces.

Data structures

The data structures used in the implementation correspond to the structures discussed throughout this chapter. Figure 5.25 shows the class diagram of a part of the implemented program. Basically there are four main structures:

Variant: A class able to maintain flexible data types. It is bases for the data structures *Property* and *Relation*.

Module: A general class able to load and maintain *Requirements*, *Interfaces* and *Components*. It provides an interface to an XML library and is basis for flexible lists

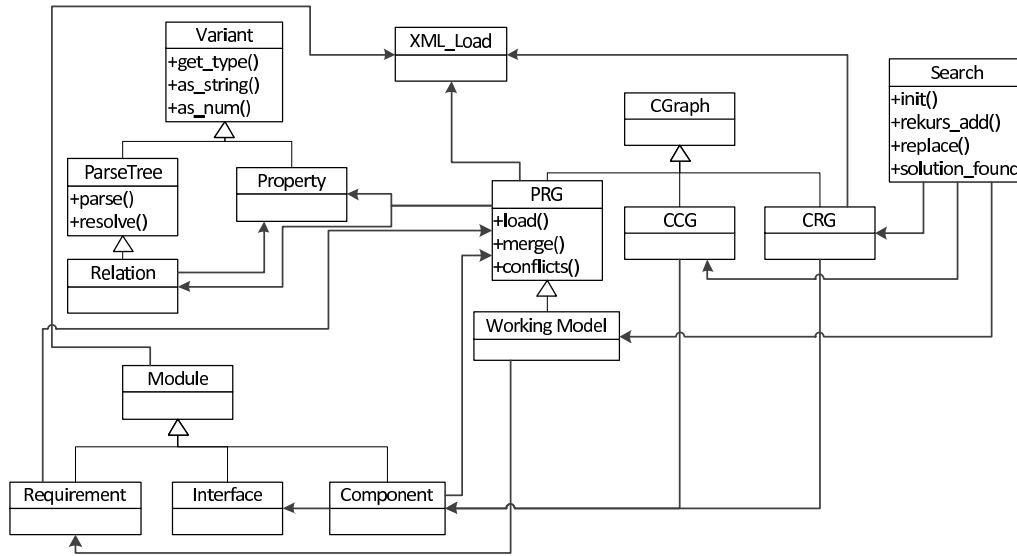


Figure 5.25: Class diagram of the implemented configKIT selection algorithm. The diagram shows a subset of the implemented classes and methods.

and stacks (not part of the class diagram). *Components* and *Requirements* use the PRG which manages the properties and relations.

CGraph: The cgraph structure is the general graph structure in configKIT. Maintaining properties and relations the class establishes the PRG which is the basis for the working model. Managing static components it is the component repository (CRG), and using instances of components it is the component composition (CCG).

Search: It is the main managing class of the program. It instantiates the other data structures and implements the recursive search method. The search basically uses the three major data types CCG, CRG and working model, which implicitly allow the access to the actual components and properties.

In the following some implementation aspects will be briefly described.

Abstract components and interfaces: The class diagram does not explicitly addresses abstract components and interfaces. They are attributes of the corresponding classes. Abstract interfaces are valuable for the modeling process since they increase flexibility substantially. However, during the composition process it has been challenging to combine the properties of two or more interfaces or several components to the one actually implementable interface or component. The solution as part of the current version of configKIT is to add dynamic relations pushing the properties from the abstract component to the concrete component and vice versa. The disadvantage of the solution is that we still have two or more interfaces with corresponding overhead of translation rules. The benefit however is that it is not required to change existing relations in the system.

Relations As described in Section 5.2, relations are functions $f(P)$ over a set of properties P : $p_1 \dots p_n \in P$. As part of the description language, relations are string-based

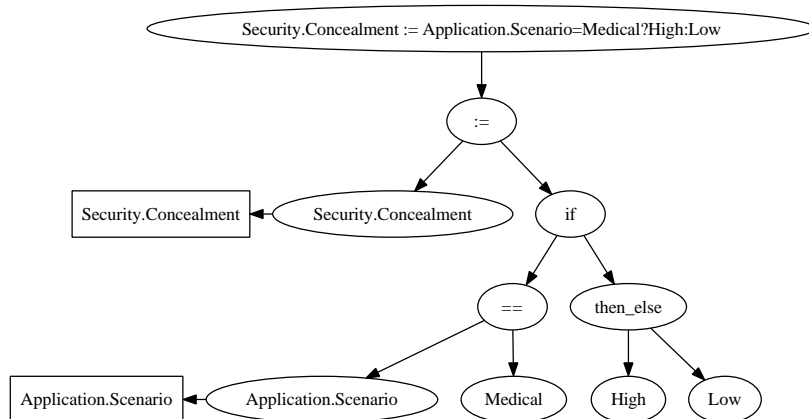


Figure 5.26: Representation of a relation as a parse tree. Ellipses show nodes in the tree, boxes indicate a link to a property of the system.

formulas while the variables in the formula correspond to properties in the system. This requires a flexible parser for the relations. For the implementation we chose binary parsing trees. The tree translates the relation as

(*R1*) `Security.Concealment := Application.Scenario=Medical?High:Low`

to a tree as it is shown in Figure 5.26. The diagram shows the actual parse tree as ellipses and already illustrates the link to actual properties illustrated as boxes.

String interpretation: One aspect of the relation parsing process is resolving literals. Literals can be resolved to numbers (floating point), strings or links to properties. For example in the relation *R1* introduced in the previous paragraph five literals can be identified in `Requirement.Security.Concealment`, `Application.Scenario`, `Medical`, `High`, and `Low`. After the first parse step only `Security.Concealment` can be clearly linked to a property, since a value is assigned to it. The other four strings are considered as strings. The result of the relation is `Low`. This value is unchanged until the user or the composition process assigns a value to `Application.Scenario`. Assumed the user defines the requirement `Application.Scenario` as `medical`, then the property `Application.Scenario` will be created and assigned with the string value `Medical`. During that assignment process configKIT recognizes the unresolved string with the name `Application.Scenario`, and consequently will redefine the string to a link to the property. Then *R1* will be resolved to `High` because the condition is true.

That string resolving approach has the advantage that neither strings nor properties have to be defined explicitly inside the relations. It improves the usability and extendability of the relation significantly since the formal overhead is kept to a minimum.

Resolve Expressions: Each literal of a relation is an instance of a variant class. This class can contain strings, numbers, booleans, links to properties, or can be undefined. The variant class can also contain information about domain, scale and intervals of its values. Since all processed values are instances of the same class, operations on the values can be implemented more easily.

It is also possible to perform operations over different types. Most inter-type operations result in the undefined state. Only compare operations return a boolean value that is false unless the operation is true.

Actual Selection Algorithm: With the existing data structures and the recursive algorithm presented in the previous section, the implementation of the algorithm is straightforward.

A recursive function takes the environment, the repository and the current module selection as parameter and calls itself to evaluate the effect of adding a new module to the composition.

Standard backtracking algorithms send the parameters in a call-by-value manner, that is, all modifiable parameters are put on the stack. The great advantage is that the actual program does not need to care about restoring the values, because each recursion step gets its own copy of all parameters.

The problem in the configKIT implementation, however, is the enormous size of environment and potential module composition. Since each new module can change the environment it would be required to store the complete environment before adding a module, and restoring the environment.

As solution for this issue we implemented a separate stack structure that stores only the objects (components and properties) that are actually changed in the recursion level. When entering the recursive function the current position on the stack is stored. When leaving the function (typically by backtracking) all objects saved on the stack are restored. This approach allows to work with one PRG and CCG without being forced to copy the entire content.

Listing 5.1: Component Type XSD description

```

1<xs:complexType name="ComponentType"><xs:sequence>
2  <xs:element name="Id" type="xs:string"/>
3  <xs:element name="Name" type="xs:string"/>
4  <xs:element name="Description" type="xs:string" minOccurs="0"/>
5  <xs:element name="Is" minOccurs="0" maxOccurs="unbounded">
6    <xs:complexType><xs:sequence>
7      <xs:element name="id" type="xs:string"/>
8      <xs:element name="name" type="xs:string" minOccurs="0"/>
9    </xs:sequence></xs:complexType>
10 </xs:element>
11 <xs:element name="Provides" minOccurs="0" maxOccurs="unbounded">
12   <xs:complexType><xs:sequence>
13     <xs:element name="id" type="xs:string"/>
14     <xs:element name="name" type="xs:string"/>
15   </xs:sequence></xs:complexType>
16 </xs:element>
17 <xs:element name="Uses" minOccurs="0" maxOccurs="unbounded">
18   <xs:complexType><xs:sequence>
19     <xs:element name="id" type="xs:string"/>
20     <xs:element name="name" type="xs:string"/>
21   </xs:sequence></xs:complexType>
22 </xs:element>
23 <xs:element name="Relation" minOccurs="0" maxOccurs="unbounded" type="xs:
    string"/>
24</xs:sequence></xs:complexType>

```

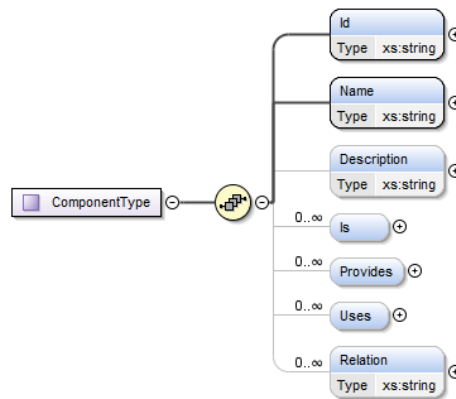


Figure 5.27: Graphical representation of the XML component scheme.

XML Description Language

For the description of components, properties and requirements we used the Extensible Markup Language (XML) [Wor04]. This standard was chosen due to its flexibility, expandability and platform independence, with plenty of tools available.

The schema defined is a prototype with emphasis on flexibility. It supports an incremental description process where information can be added when it becomes available or when it is required.

Listing 5.1 shows part of the current component XSD (XML schema definition), which defines the file format of the XML files. Beside an ID and a name the scheme allows to define abstractions, used and provided interfaces for the structure and the relations to express the semantics. Figure 5.27 shows a graphical representation of the schema. Similar XSDs are available for interfaces, properties and requirements.

5.6.2 Tool support

This section presents a set of tools that embed the configKIT selection algorithm in a prototypical toolchain which implements part of the design flow introduced in Section 5.1. Figure 5.28 shows the current tool chain, identifying the three main operations:

- setting up the repositories as task (by the framework designer) that has to be done before the actual WSN engineering starts,
- selection of the requirements by the user,
- and the actual composition as part of the application engineering process.

The three operations are discussed briefly in the following subsections.

Set up of Repositories

One major problem we found for most model-driven analysis approaches is the absence of tools to set up and extend the fundamental databases. As introduced earlier, in our decisions process we need three databases: The requirement repository, the property repository, and the component repository.

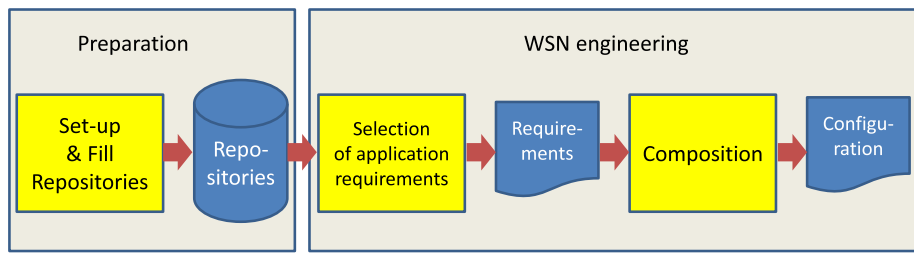


Figure 5.28: The WSN engineering tool chain: The preparation phase, executed by the framework designer is followed by the actual engineering process that can be executed by the user.

Compared to other modeling languages, the XML scheme presented in the previous section is already not too complex, and the XML syntax is supported by many editors. We are convinced that it is naive to assume that external developers would start to learn the semantics to add their modules. This is why we spent efforts to develop a GUI that allows to fill out the databases in a convenient application front-end. Figure 5.29 shows the GUI to edit the components database. On the top a user can see the components and their relations to other modules and interfaces. The modules can be selected to edit their properties in the bottom windows. The corresponding XML file will be generated and stored in the database so that it is available for the future selection processes. Similar front ends have been designed for the requirements database and the reasoning database. Our evaluations show that they ensure creating and maintaining database entries that are syntactically and semantically correct. Still there are some weaknesses that could be fixed in future. For example entering long formulas for property computation without further assistance can be frustrating and error-prone. Also specific support for models, such as the security models developed in this thesis, is not yet supported by the GUI.

User Interface

For the definition of requirements and the presentation of the results a web front-end was used. The user can define the requirements in a dialog as shown in Figure 5.30. The user interface is an XSL representation of the requirements XML database. The dialog allows to add requirement types from a catalog (requirement repository). After adding them to the dialog, the requirements can be defined. On the top, the application designer can chose the domain of the parameters, the select the parameter and add it to the list of requirements. The parameterization will be done in the input field of the list. The dialog posts the selections to a perl [PER] script which generates the requirements description. This application requirements statement is the input for the selection and evaluation process.

After the requirements are written, the perl script starts the configKIT executable. It uses the property database and the component database to compute suitable configurations. For a preferred solution (usually the smallest) the architecture of the intended system is represented as graph, using the graphviz library [Gra11]. Alternative solutions with key figures are represented in a table and stored on the server, allowing a user to select them.

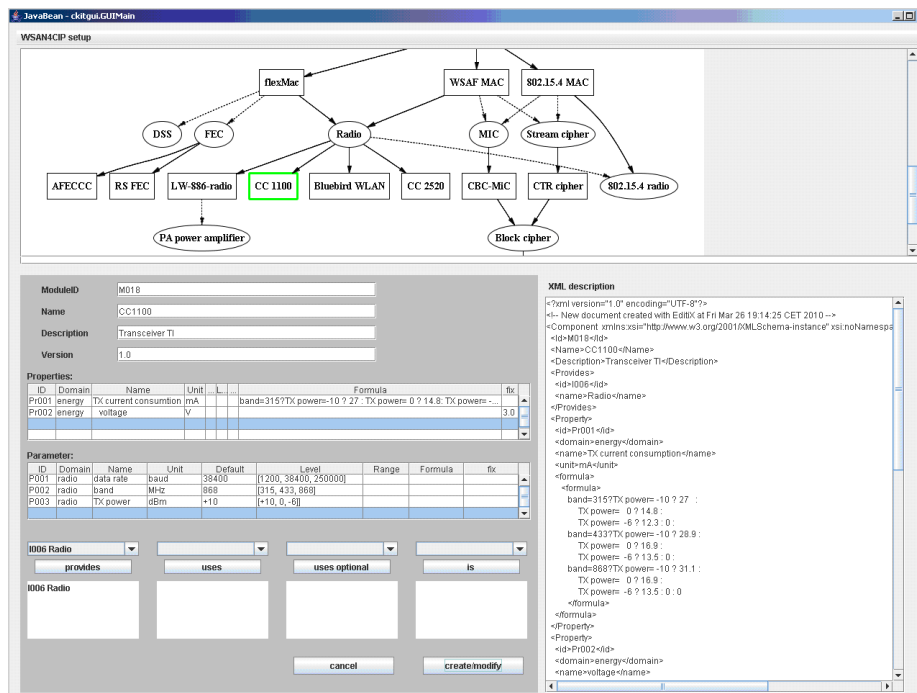


Figure 5.29: GUI of the components database editor.

The results are presented on a result page as shown in Figure 5.31 after configKIT finished. Typically the entire computation process takes less than one second.

5.7 Conclusions

This chapter presented the component-based development framework configKIT. It is no specialized CBD tool but a general tool chain supporting the development of systems from the requirement elicitation to the automatic composition of systems satisfying the

The screenshot shows the 'definition of application requirements' window. It features a form with the following fields and options:

- domain:** security (dropdown)
- type:** reliability (dropdown)
- add requirement** (button)
- network / topology:** tree (dropdown)
- mechanism / in-network aggregation:** enabled (dropdown)
- security / concealment:** none (dropdown)
- parameter / energy requirement:** medium (dropdown)
- hardware / node:** MicaZ (dropdown)
- submit** (button)

Figure 5.30: User interface for entering the application requirements.

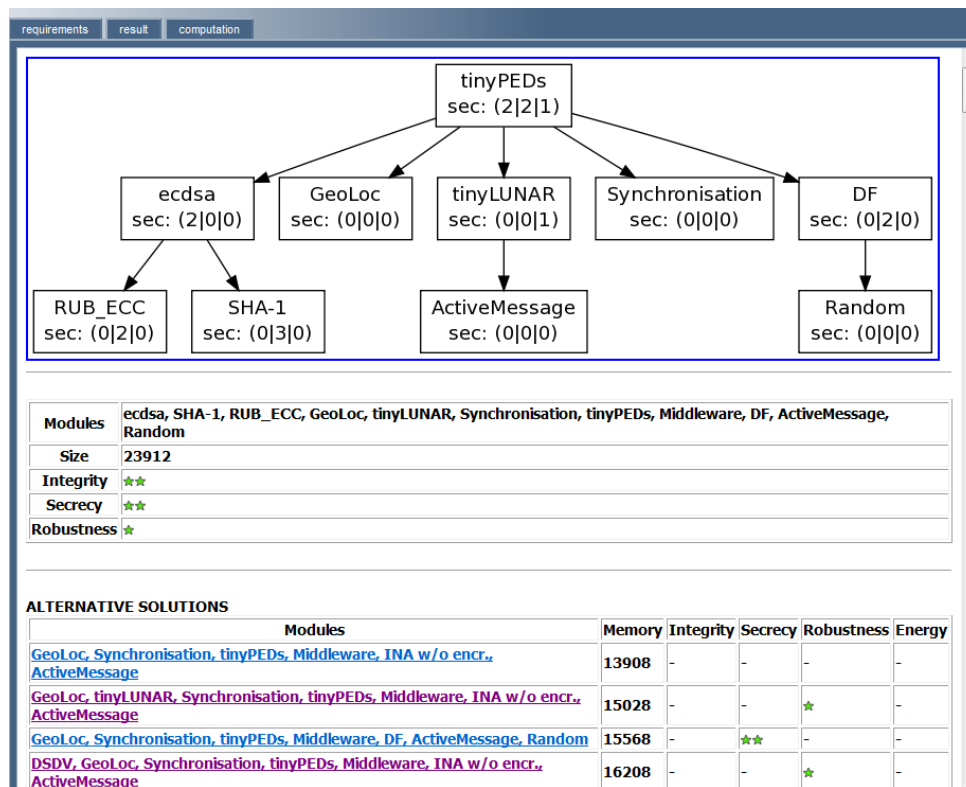


Figure 5.31: Screen shot of the result page of the configuration process.

given requirements. As result, every system composed of components for which the behavior can be modeled with a set of interrelated properties basically can be configured with the configKIT framework.

The most significant difference to state-of-the-art CBD frameworks is that configKIT covers behavioral aspects of the composed system. For the processing of the behavior configKIT uses a Property-Relation-Graph (PRG) that maintains all properties from requirements, environment, and the system under development. For the connection and the evaluation of the properties individual models can be instantiated in the PRG. By this, even complex models for the assessment of memory or energy consumption can be processed. Several dedicated security models are proposed in the next two chapters.

For the composition of the system a rather high level component model was introduced. It describes the interfaces and the properties of software and hardware modules which are represented by the components. The interfaces control how the components can be assembled to a complete system while the properties of the components –added to the global PRG – allow the assessment of the behavior of the composed system and to detect potential conflicts.

It is the task of the selection algorithm to find a composition of available components to a system free of conflicts that satisfy the initial requirements. This chapter described the development of a practically efficient selection algorithm for that purpose. Finally, the selection algorithm was implemented and integrated in a practical tool chain supporting developers and users in the assembly of WSN systems.

Chapter 6

Security Models

Basically security is just another property of the system under development. Therefore security should not be treated differently than other functional and non-functional system requirements. It should fit in the general development flow without the need for additional steps or tools.

However, security is not as tangible as other functional properties of the system, which can be described as existent or not. As discussed in Section 4.7 security features lack measurability in general and the means to assess it a-priori in particular. Therefore models are needed that provide abstraction to the term security in a way that they can be handled inside the configKIT toolchain.

The model-related parts of configKIT are the requirements, the component composition (CCG), and the world model (WM) containing all properties and relations. In context of security their task is:

- The requirements provide the translated security requirements or the assumed attacker model, i.e. all parameters from outside the actual system.
- The CCG provides the security related properties of the composed system.
- The WM combines both inputs and derives a model-based assessment of the awareness of the system in the context of the requirements.

This general security model assessment architecture is illustrated as Figure 6.1.

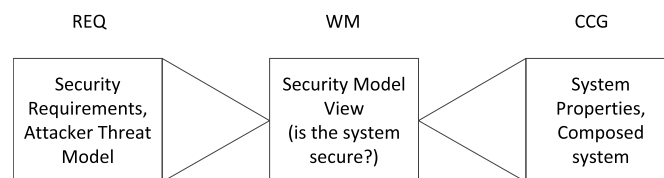


Figure 6.1: Security model assessment architecture: The security model view assesses the security status of a system based on inputs describing the system properties (right) and the model of attacker and/or security requirements (from left). Involved data structures are REQ, WM, and CCG.

This rationale is first applied to security assessment approaches presented in the literature. As result all approaches suitable in the domain of WSNs that are known to the author of this thesis could be integrated in the configKIT framework. The results, however, also indicated that the existing models lack generality.

As answer to this issue general security models are developed based on specific views on the security ontology presented in Section 4.1.2. The views on the ontology provide a good abstraction and allow to explain security properties. However, a mapping from the views to the property-driven configKIT framework is not obvious. That is why the view-based models are further refined with focus on deduction of requirements and composability in the component model. Finally five implementable security models could be developed and integrated in configKIT, while each model provides a different abstraction of security. The differences of the models, beside the abstraction quality, are the needed efforts to describe and maintain the data for the assessment process.

6.1 Integration of Existing Approaches

This section describes in detail configKIT adaptations of the two most advanced security configuration tools available for WSNs. They were already introduced in Section 4.6.3. The results of the integration are evaluated by comparing the results of examples presented in the corresponding papers with the results of the same examples in configKIT. The adaptation demonstrates the practicability of the general security assessment approach within the configKIT framework. It also demonstrates the capability of the framework to handle various different models.

6.1.1 KMS Guidelines

KMS Guidelines [AR06] is a security mechanism selection toolkit to configure key management schemes (KMS) for WSNs. A user starts the selection process by providing objectives that are important for the application scenario. After entering main and secondary properties, e.g. small memory, connectivity, scalability, resilience, the tool delivers a list of key distribution schemes that fulfill the requirements. The properties discussed in the work are memory footprint, security, network resilience, connectivity, scalability, communication overhead, energy, and localization

In [AR06] totally 24 KMSs were evaluated and classified concerning their individual objectives. Table 6.1 shows an extract of the analysis.

Integration in configKIT

For the integration of the KMSG approach in configKIT each objective (i.e. connectivity, resilience, ...) is represented as a binary property (with undefined as third state). A positive objective is encoded as property 'TRUE' - a negative objective is encoded as property 'FALSE'. Otherwise the property is undefined. A more extensive encoding using several qualities is considerable but not required in this case. The basic notion is that the user defines the required objectives in the requirement definition process, the components provide their individual properties as meta-information

The Requirement Definition has to allow the user to enter the required primary and secondary features directly. For each of the objectives the user can define whether the

Table 6.1: KMS CRITIS Guidelines - extract from [AR06]. The table lists four key distribution mechanisms and their beneficial (+) and negative (-) properties. Evaluated properties are Connectivity (Conn.), Resilience (Res), Communication (Comm), Scalability (Sca), Location-dependency (LOC).

Mechanism	Properties
Closest Pairwise	+ : Conn., Res, Comm, Sca - : LOC
Grid Based Predistribution	+ : Res, Conn., Comm, Sca - : Ext., Mem, Energy, LOC
Polynomial Key Predistribution	+ : Res, Comm., Sca - : Mem, Energy
Symmetric Design	+ : Conn., Comm - : Sca., Mem., Res

objective is a primary objective, a secondary objective or should be avoided. Therefore we defined the following transition:

- If a property is a primary objective the property has to be true in the selection
- If a property is a secondary objective the property has to be not false in the selection
- If a property is an objective which must be avoided in the selection the property has to be false

The advantage of this definition is the replacement of the fuzzy secondary requirements. The must-be-avoided-constraint is required to support properties such as the need for pre-deployment localization information.

The Components provide an interface each, offering the KMS service to an application. Figure 6.2 shows the reduced component composition design space graph. The actual diagram has more design alternatives than the four depicted KMS approaches.

Each KMS component contains a list of properties according to the data of the KMS analysis (see Table 6.1). For instance the “symmetric design” approach provides the properties:

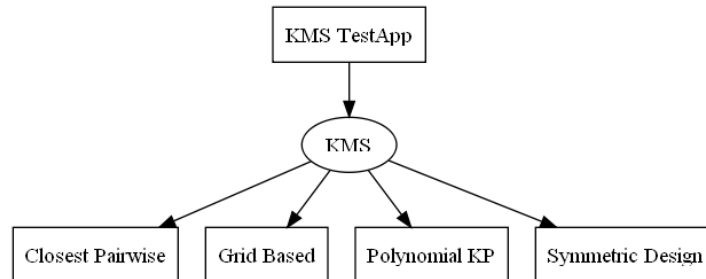


Figure 6.2: Component Composition Design Space diagram for the KMS example: One application uses the KMS interfaces, which is provided by several optional components.

connectivity := true
resilience := false
communication := true
scalability := false
memory := false

The other objectives are implicitly undefined.

The WM needs matching constraints between the required objectives and the properties provided by the selection. Due to the addition of the avoidance condition a direct comparison of the required objectives and the application properties is not possible. Thus we need the following condition for each objective:

(required.objective = primary and application.objective = true) OR
(required.objective = secondary and application.objective! = false) OR
(required.objective = avoid and application.objective = false) OR
required.objective = undefined,

to implement the logic described above.

Examples

[AR06] provided several rather informal examples to demonstrate the decisions made by the KMS Guidelines process. One is an application scenario for the monitoring of aging infrastructures. According to the authors the connectivity is the most important property for such scenarios while resilience and security are secondary objectives. In the paper these requirements are best satisfied with a specific hybrid approach, while a second solution is also considerable. The requirements applied to the configKIT implementation of the KMSC approach resulted in five solutions satisfying the requirement, including both selections presented in the paper. What filtered the preferred selections of the authors is not traceable. A special weighting of the secondary objectives that is not discussed in the paper could be an explanation.

Another example in the paper is a Self-Powered Communication and Diagnosis System. In that scenario the primary objectives are Resilience, Connectivity, and Scalability. The secondary objective is security. The authors find two solutions: grid based predistribution and polynomial key predistribution (both in Table 6.1). The configKIT implementation can only find the grid based predistribution, since the polynomial key predistribution does not provide the required connectivity. A reason for the discrepancy could be a typo in the property table in [AR06] or another fuzzy rule not described in the paper.

But in principal the results presented in [AR06] could be reproduced.

6.1.2 Cionca Security Configuration Approach

Cionca and Neue [NCB10] proposed a security-related configuration tool, which in a graphical user interface allows users to describe relevant parameters of the application.

Table 6.2: Security protocol assessment as presented in [CND09]: For different protocol categories specific protocols with their constraints and properties are listed. EpP is Energy per Packet

Protocol category	Protocols	Properties, Constraints
Key Predistribution	Pairwise	Hostile Environment Nodes < 100 Meshed, Static Network
Key Predistribution	Master	Trusted Environment Nodes < 100 all Networks
Key Predistribution	Server shared	Hostile Environment centralized Networks
Key Predistribution	Probabilistic	Public Environment Nodes > 100 Mesh Networks
Data Security	TinySec	Cipher=SkipJack, strength= 2^{16} EpP= $0.176\mu\text{AH}$
Data Security	MiniSec	Cipher=SkipJack, strength= 2^{64} EpP= $0.167\mu\text{AH}$
Data Security	SNEP	Cipher=RC5, EpP= $0.188\mu\text{AH}$

From these inputs the tool presents a set of security protocols that suit the requirements. The tool supports four security primitives: key predistribution, key establishment, confidentiality and authentication.

To release users from entering detailed security specification, application types are used to derive the environment security levels. Military applications have high security requirements, medical applications need high data security in a public environment, structural monitoring applications have low requirements while home applications are assumed to run in trusted environments.

The Cionca tool presents a selection algorithm that processes direct network parameters and constraints, as well as the environmental security parameters derived from the application type. The selection algorithm starts with the full set of potential protocols for each of the four security primitives. Applying the database containing the results of the initial analysis each potential protocol is checked on constraints or properties that do not suit the application requirements.

For this purpose for each protocol p and each parameter r a constraint is defined as function of the required value v :

$$c_{p,r}(v) = \begin{cases} 1 & \text{if } p \text{ has value } v \text{ for } r \\ 0 & \text{if } p \text{ has different value than } v \text{ for } r \\ 1 & \text{if } p \text{ does not have a value for } r \end{cases} \quad (6.1)$$

This means that the constraint is satisfied if the function returns 1. A protocol is suitable in context of an application if all constraint function of the protocol return 1. Hence, all protocols that do not satisfy all of their constraints can be disabled during the selection process.

Examples

In [CND09] a scenario explains the configuration process. In a considered hospital every room has a WSN deployed on patients to monitor vital parameters. Each patient is a mobile cluster of sensor nodes. The data should be reported every second for a runtime of up to one month. It is assumed that standard nodes with two AA batteries are used.

The Reasoning process concluded that the environment is public which excluded protocols that are either not suitable or not sufficiently secure. From the key predistribution protocols in Table 6.2 the Master key approach will be disabled. Pairwise and Probabilist KPDs cannot be applied due to the hierarchical topology of the network. Similar reasoning was made for the key establishment.

The reasoning regarding data security reveals another interesting capability of the selection approach. Based on the lifetime (one month) and the communication period (1 second) the number of packets over the lifetime is computed (in this case 2,678,400). This number is used to disable security protocols that overflow earlier. Following the parameters presented in Table 6.2 TinySec has to be deselected. For the remaining two protocols the expected energy consumption for the expected packet count is computed and compared with the energy available from the batteries. Since both protocols (SNEP and MiniSec) pass this test, they both are recommended while the authors prefer MiniSec because it has been implemented.

Integration in configKIT

According to Table 6.2 each algorithm has a set of properties and constraints which determine whether the algorithm is suitable for the given requirements. The notion of the integration in configKIT is to add these properties and constraints to the relations attached to the components. Then the component is suitable for a system only if the relations cause no conflicts.

The precise technical information is derived from the user inputs with the requirements step.

The Requirement Definition: To replicate the inputs of the Cionca tool, additional inputs are needed to enable the definition of the network parameters. The new requirement inputs can be similar to the GUI shown in [CND09], allowing a user to enter the Application type and network parameters such as number of nodes, topology, available energy, needed lifetime and communication frequency. While energy, lifetime and frequency are numbers, the other properties need individual enumeration scales. For instance the application type is a selection over [military, medical, industrial, home].

Since different explicit physical units are used, the units to be translated as part of the requirement expansion process. For example the number of packets has to be computed from the packets per hour and the lifetime.

The Components provide an interface for the protocol category they represent. Figure 6.3 shows a partial component composition design space graph. The actual diagram has more design alternatives.

The properties provided by each component correspond to the properties listed in the analysis table (Table 6.2). However, due to the heterogeneity of the characteristic of the

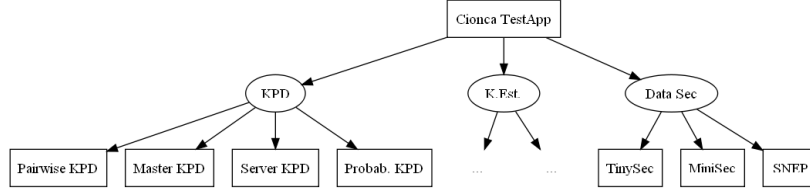


Figure 6.3: Component Repository of configKIT for the Cionca example: One application uses several interfaces. The selection is made for each interface separately. The graph only shows components for Key-predistribution (KPD) and Data Security

component attributes not all of them can be expressed as direct property. For example the server shared KPD runs on centralized and hierarchical topologies. In the configKIT framework this issue can be solved by constraints attached to components.

For instance, for the “Server Shared KPD” approach the relational properties are expressed by the two relations:

$$Environment \leq Hostile$$

$$Topology == Centralized \text{ OR } Topology == Hierarchical$$

In case the component should be used in the selection process, both constraints must be true. Otherwise the component will be deselected. This realization perfectly represents the constraint logic of the original authors (see Equation 6.1): a component will only be ruled out if it has a property or constraint that is violated by the application space.

The PRG is less complex than the one needed for the KMS approach, because the component constraints are handled in the component descriptions. This is also true for the energy consumption. In the current implementation the allowed energy per packet is computed as part of the Requirement Expansion. The result can be directly used in the constraints checking of the components.

With the setup we could exactly reproduce the results of the example described above.

6.1.3 Conclusions

In this section we presented the implementation of the state-of-the-art security configuration tools for WSNs in configKIT. This in first place demonstrates the expressiveness of configKIT. With the setups for KMS and Cionca implemented in configKIT, the two security models can be applied for all compositions executed within the configKIT toolchain.

Both approaches are feature based while the implementations have been different. KMS relies on static database-like components and a more powerful matching logic. In Cionca the components inherently deliver the knowledge whether they work in the given environment or not. Transferring the constraints by checking the protocol compatibility into the components also helps understanding the more complex relations of the system. By this, the Cionca approach follows the notion of a smart component meta-description, rather than the passive property-providing methodology in KMS.

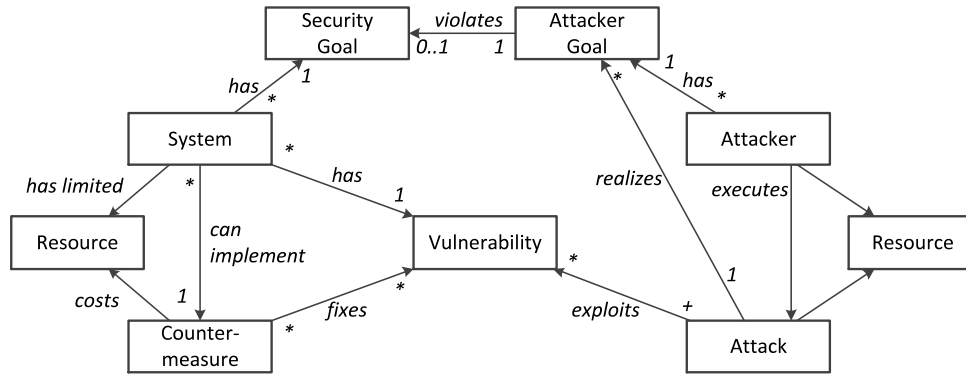


Figure 6.4: Full Ontology as introduced in Section 4.1.2

One weakness of KMS we could also reproduce in configKIT is the need for the user to define deeply technical requirements. Here an automatic mapping from the application scenario to the requirements as it is shown in the Cionca approach is beneficial. Even though Cionca needs a more deliberate definition of properties and scales instead of the binary properties in KMS the total description overhead and the complexity of the implemented model is on the same level.

Both approaches can be considered as specific instances of the feature-based security model, which is discussed in more detail in section 6.3.1 which is introduced as a more general security framework in the following sections.

6.2 View-based Model Development

In this section a methodology for the development of security models is presented and applied. It is the objective that the models are expressive, practically usable and capable of being integrated in the assessment logic of the configKIT framework.

The starting point of the methodology is a rather complex model or ontology describing the entire system, objects, and participants. For the aspect security we presented such an ontology in Section 4.1.2. It is shown again as Figure 6.4. In this model we consider an attacker with a set of attacker goals. The attacker goals can be realized by a set of attacks. The attacks need a set of vulnerabilities the system has and which are not fixed by countermeasures.

Even though the complexity of the ontology is already condensed, it still does not pose an immediately usable model to assess security properties of systems. The ontology just shows the relations between the objects. However, based on such an ontology, concrete models can be derived by focusing on specific views inside this larger ontology.

In our case we focus on specific objects that suit as virtual front line between system and attacker. The objects are Countermeasures, Vulnerabilities, and Attacks. It is the notion that concentrating on specific objects leads to focused views, hiding complexity of the global system. Such focused view allows to strip peripheral details and thus reduce the total model complexity. With such reduced models it becomes possible to express the needed information needed from system side and requirements side to decide whether the system under development satisfies the requirements within this model. The process

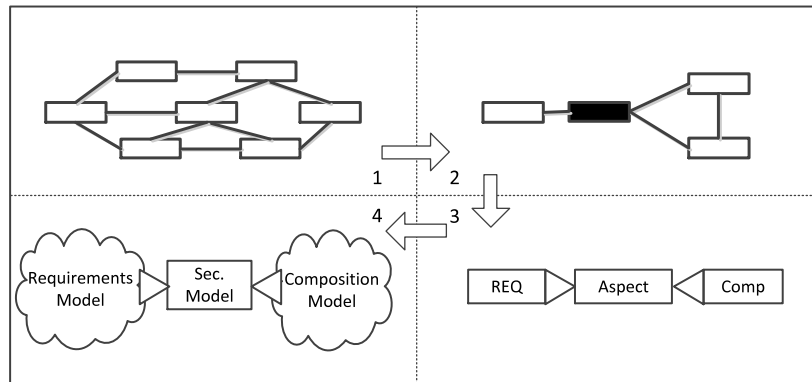


Figure 6.5: Illustration of the security model development methodology: Based on a complex ontology (1 - top left), a view focuses on one aspect (2 - top right). From this a condensed model between requirements and composition will be inferred (3 - bottom right), which finally can be extended to full technical models (4 - bottom left).

steps of the methodology are illustrated in Figure 6.5. It corresponds to the security model assessment architecture as introduced earlier in this chapter (see Figure 6.1).

In the following subsections each view is discussed regarding its ability to decide whether a system is secure against a modeled attacker and which information are needed for such a decision. Therefore, the corresponding central object (countermeasures, attacks, vulnerabilities) is put in the focus of the discussion and surrounding objects are combined. This facilitates the condensed description of

- a model description,
- the definition of security in the context of this view,
- the required information from the composed system, and
- the required information from the requirement definition process,

as it is provided in the following for each of the three views.

6.2.1 Countermeasure-centric View

Countermeasures are the functional link between the system and the attacker world. Since countermeasures are already connected with the system in the ontology this side of the view is already defined and can be reused without modifications. The view on the attacker world however needs adaptations. First, attacker, attacks, and vulnerabilities, will be united to a new general attack object. Such an Attack can be stopped (fixed) by a specific set of countermeasures. The resulting reduced security ontology can be seen as Figure 6.6.

Thus we can define the following model description

Model: Definition of the countermeasures that are required for a secure system.

A system is secure if all required countermeasures are part of the system in sufficient quality.

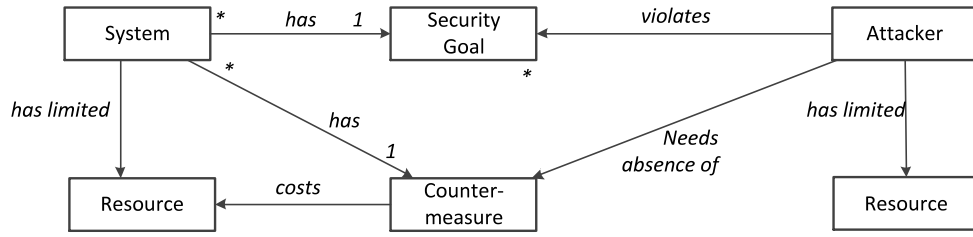


Figure 6.6: Countermeasure Centric Ontology.

Required information from the composed system: list of countermeasures.

Required information from the requirement definition process: list of countermeasures that have to be implemented to prevent successful attacks.

6.2.2 Vulnerability-centric View

According to the security ontology, vulnerabilities are the link between system and attacks. Contrary to the initial model, in the vulnerability view the countermeasures are combined into the system. Since the system will be assessed at development time it can be assumed the countermeasures are part of the system and no add-ons. The relation to the attacker side does not change from the initial model, as the vulnerabilities are still exploited by attacks.

The resulting reduced security ontology is shown as Figure 6.7. It allows to describe the model description as follows.

Model: Evaluation of vulnerabilities a system has and the vulnerabilities that are required for a successful attack

A system is secure if it does not allow a combination of vulnerabilities which could lead to a successful attack.

Required information from the composed system: list of vulnerabilities of the system.

Required information from the requirement definition process: information of needed vulnerabilities to execute an attack

It is the basis for a vulnerability centric security model which is discussed in section 6.3.5.

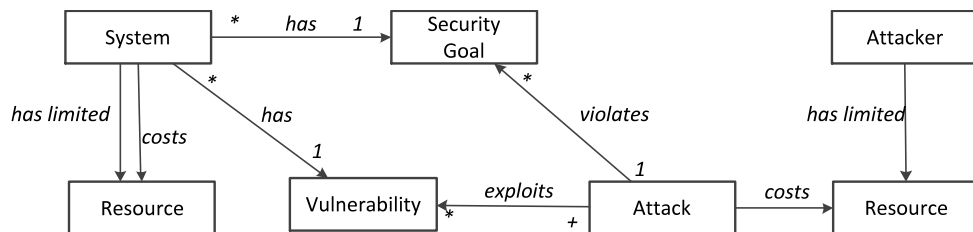


Figure 6.7: Vulnerability Centric Ontology.

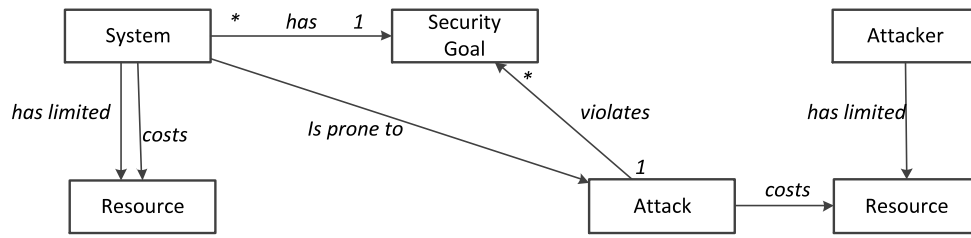


Figure 6.8: Attack Centric Ontology.

6.2.3 Attack-centric View

Similar to the vulnerabilities, attacks are the link between the system and the attacker. As view from the attack, the vulnerabilities are part of the system. While the *exploits* relation from the attack to the system is not wrong in the attack-centric view, formulating the relation in opposite direction appears to be more natural to express the fact that a system can be exploited (or attacked) by an attack as property of the system. Similarly, to illustrate the data flow towards the attack the relation from the system goals are formulated in a way that the system goal is violated by the attack.

The result is an attack-centric security model depicted as Figure 6.8. In this attack-centric view the attacks are the front-line between the attacker and the system.

Model: Evaluation of Attacks a system is prone to.

A system is secure if it cannot be exploited by an attack.

Required information from the composed system: list of attacks the system is prone to or list of attacks the system resists.

Required information from the requirement definition process: list of attacks that threaten the system.

It is the basis for the attack centric security model which is discussed in Section 6.3.3.

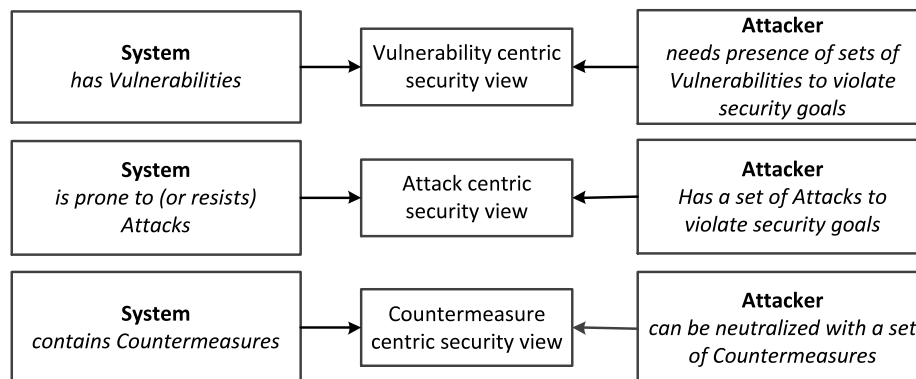


Figure 6.9: The three Security model views and their notion of System and Attacker

6.2.4 Conclusions

Figure 6.9 illustrates the three model views discussed in this section. Each model has its individual notion of attacker and system and the information that has to be gathered. This section avoided details on the question how the required information can be obtained. It was rather important to derive generic security models that are basis for clear unambiguous decisions and can be understood. However, the quality and usability of the models depend on the concrete implementation of attacker and system sub-models. This will be discussed in the following section by means of concrete implementations.

6.3 Practical Security Models and their Integration in configKIT

For each of the general security models proposed in the previous section the major challenge is to derive the required information to decide whether a system is secure. Basically, we have two sources for the information: First, the system description entered by the user, and second, the information of the composed system under development. How the information can be derived varies with the notion of the models

In this section five concrete instances of the models presented in the previous section are investigated:

- A Quality-based Countermeasure-centric Security Model (QCSM)
- A Feature-based Countermeasure-centric Security Model (FCSM)
- A generic Attack-centric Security Model (GASM)
- An Attack Tree based Attack-centric Security Model (ATASM)
- A generic Vulnerability-centric Security Model (GVSM)

For each model it will be described how it works, how it assess the state of security, how the mapping works from both requirements and composition. Finally it will be shown how each model can be implemented in configKIT. This integration is important for three reasons:

- It demonstrates that the security model is in fact implementable rather than a theoretical notion. The implementation also pinpoints practical aspects in the model for further refinements.
- It allows to estimate the overhead for modeling the components and the system. The overhead can partly be measured as lines of model descriptions, but can also be organizational overhead for designer or user.
- It is the basis for actual test cases.

The models are all optional and do not interfere the general selection of configKIT. All other assessments of code size, functional properties, hardware compatibility, radio channels are not affected by the applied security model.

A thorough evaluation of the implemented models is then given in Chapter 7.

6.3.1 Feature-based Countermeasure-centric Security Model (FCSM)

The basic notion of this approach is that countermeasures can be considered as features. Similar to features the countermeasures can be compared on presence in the eventual

system. This notion is common practice in manual and automatic application design processes. In fact, both security composition schemes for WSNs (KMS and Cionca) that were implemented for configKIT in Section 6.1.1 and Section 6.1.2 respectively, are feature based. The model described in this section is a generalization of the Cionca approach.

The model approach is to infer the features that are required for a secure application, and to compare that set with the features provided by the composed application. Features in configKIT are represented by Properties.

A composition is secure if all needed features are present in the composed application.

$$\forall F_R \in P_{SecReq} : \exists F_{Comp} \in P_{Comp} : sem(p_{Comp}) = sem(p_{Req})$$

that is, for each feature (F_R) of the set of security requirements (P_{SecReq}) exists at least one feature (F_{Comp}) in the set of application properties (P_{Comp}) with the same semantics.

Mapping from the requirements: The list of required features is can be either entered directly from the user, or derived from an application profile. The latter, the direct projection from security requirements to required features is a common model to engineer security as we could already see in the KMS and Cionca approach as well as in many examples described for instance in [And08].

The result of the application mapping is the set of properties P_{SecReq} needed in the composition.

Mapping from the composition: Here we follow the notion that a feature that exists in one of the composed components also exists in the composed application.

Considered a component C uses interfaces I_1, I_2, \dots, I_n . Each interface propagates the set of properties of the implementing component $P_{I1}, P_{I2}, \dots, P_{In}$, respectively. Then the properties P_{CC} of the composed component C is the union of

$$P_{CC} = P_C \cup P_{I1} \cup P_{I2} \cup \dots \cup P_{In} = P_C \cup \bigcup_{i=1}^n P_{I_i},$$

so that the set of the composed component C is the aggregation of all features of component C and the features provided by the used components.

Influence of the environment: The environment can influence both the security requirement mapping and the component selection.

As already demonstrated for the Cionca approach, components can parametrize their properties and define the environments they work in. For instance several key management systems work for centralized networks but not in mesh networks. This environment-based filtering clearly affects the components selection.

Implementation in configKIT

In configKIT features (or countermeasures) naturally are represented by properties.

The **component description** contains the properties, while the properties can be set directly or alternatively based on environmental properties. If a property P_C of the component C is defined as function of m environmental properties $P_{E_1}, P_{E_2}, \dots, P_{E_m}$, we add the relation

$$R : \langle \{P_{E_1}, P_{E_2}, \dots, P_{E_m}\}, P_C, f_c(P_{E_1}, P_{E_2}, \dots, P_{E_m}) \rangle$$

The component description also contains constraints to respect technical or environmental incompatibilities. The constraints are expressed as relations. The concept has also been discussed in the description of the Cionca approach in Section 6.1.2.

The **application requirement definition process** is a standard requirement expansion: based on the given user inputs, semantically equivalent requirements are derived. A specific example is the application type. The type is determined based on the properties given from the user, then it explicitly defines the list of required security features which are common for this type of application.

For this application mapping we need a relation to map a set of application properties P_A^* to the application type $P_{AppType}$:

$$P_A^* \rightarrow P_{AppType}.$$

Then a list of relations is needed to infer the needed features:

$$P_{AppType} \rightarrow P_{Feature1}, P_{AppType} \rightarrow P_{Feature2}, \dots, P_{AppType} \rightarrow P_{Feature_N}$$

Finally for each feature a constraint has to be added to the PRG, enforcing it to be part of the composition.

The **composition process** is finally just the search for a composition that fulfills the list of constraints by providing the corresponding properties in context of the environment. With the data types defined as described above that is a normal run with the configKIT selection algorithm.

Implementation overhead

The implementation overhead for the FCSM approach is rather low for components and comparison logic. However the mapping from the application requirements to the list of needed features needs some description overhead, simply because each feature has to be inferred separately from the intermediate application type.

6.3.2 Quality-based Countermeasure-centric Security Model (QCSM)

Countermeasures cannot only be expressed as features but also as qualities. This is already applicable in form of attributes for actual features. Then for instance an encryption can be attributed with high or low strength. A qualitative scale for security properties was presented in 4.7.4.

In this section we go a step further and present a methodology to describe countermeasures as qualities of security attributes, such as secrecy, integrity and reliability. It is an additional abstraction that ultimately allows to express security requirements of the system as qualities of these security attributes. Formulating security attributes as features is

a natural way to describe security properties for non-security insiders. Then for example an application would need “high integrity, medium reliability and no concealment”.

The disadvantage of this convenient abstraction is the distance to actual technical details. The approach discussed in this section bridges the gap with a process that eventually allows to assess the entire system with these primary security attributes. Key in this assessment process is a hierarchical decomposition of the system under development. Briefly, if a component is aware of the qualities of the components it uses it can assess its own qualities. This approach well-directed exploits the properties of the configKIT framework.

The model is to describe both the required application and the composed application with qualitative security attributes that can be compared. Attributes are secrecy, integrity, or reliability.

A composition is secure if all required attributes for the application are satisfied by the composition.

$$\forall p_R \in P_{SecReq} : \exists p_C \in P_{Comp} : p_{Comp} \geq p_{Req} \cap sem(p_{Comp}) = sem(p_{Req})$$

meaning that for all properties from the set of security requirements exists at least one property in the composed application that provides at least the required strength and fulfills the required semantic.

If we consider the primary security attributes secrecy, integrity and reliability we can write directly:

$$\begin{aligned} & (sec_{Req} \notin P_{SecReq} \cup (sec_{Comp} \in P_{Comp} \cap sec_{Comp} \geq sec_{Req})) \bigcap \\ & (int_{Req} \notin P_{SecReq} \cup (int_{Comp} \in P_{Comp} \cap int_{Comp} \geq int_{Req})) \bigcap \\ & (rel_{Req} \notin P_{SecReq} \cup (rel_{Comp} \in P_{Comp} \cap rel_{Comp} \geq rel_{Req})), \end{aligned}$$

meaning that an application is secure if each of the security requirements is either not part of the application requirements or the requirement is defined in the composition in a quality which satisfies the requirement.

Mapping from the requirements can be done directly by the user because the requirements are primary security attributes that can be understood by non-security-experts. The set of properties is rather small and understandable so the application designer can define them explicitly.

Alternatively a mapping from the application type, similar to the approaches discussed in Section 4.7.2, can define the required attributes.

Mapping from the composition: This is the challenging operation in this model.

The mapping can be done manually. Then a composed application will be assessed by an expert and the result is stored with the application. The selection process in such a scenario is reduced to selecting an application that provides the required properties.

An automatic deduction of the security properties of the composed application needs a specific composition model with according meta information. The decomposition strategy is described in the following subsection.

Influence of the environment The effect of environmental properties on the quality aspects of the components has to be defined inside the meta-information of the components. It means that the composed application is able to assess its security properties in context of the given environment.

The effect of the environment to the application decomposition is from minor significance, due to the presumption that security requirements for an application do not change with the environment.

Decomposition Strategy

The fundamental idea is that an application can be decomposed to smaller components to a level that allows to assess security properties of the components directly. For the assessment it is further assumed that a component which uses other components can assess its own security characteristics if it is aware of the security properties of the used components. It means if a component C uses i interfaces I_1, I_2, \dots, I_i , the security properties of C sec_C are determined by a function f_{sec_C} over the security properties $sec_{I_1}, \dots, sec_{I_n}$ of the used components

$$sec_C = f_{sec_C}(sec_{I_1}, sec_{I_2}, \dots, sec_{I_n}). \quad (6.2)$$

while the security properties sec_c are vectors containing the supported security attributes $sec_c = [sec_{Attr_1}, sec_{Attr_2}, \dots, sec_{Attr_n}]$

This formula is interesting because, first, it uses properties of the interfaces instead of modules. This corresponds to the composition methodology of configKIT. Accessing properties of interfaces instead of components implicates the additional advantage that one module can provide interfaces with different security and performance properties. For instance a network module can offer to send packets encrypted with corresponding data and computation overhead, or alternatively to send them unencrypted.

The second mentionable aspect on Formula 6.2 is that f_{sec_C} is not a generic function but characteristic for the component C and its security property. The function also is not necessarily a direct mapping of the parameters. Instead, the characteristic and behavior of the component C can influence the outcome of the function. It is assumed that the knowledge about this influence is present for the component developer.

Formula 6.2 is also the major discriminating aspect in comparison to similar approaches in literature: As introduced in Section 4.7.3, [OMKD09] describes a similar approach to assess security assurance for systems of aggregated components. However, their approach does not allow to evaluate security properties on a modular basis and ignored the inherent properties of the parent components.

Since the aggregation in the configKIT QCSM model are expressed as flexible relations, it can process all three types of functions flexible and individually for each component in the system.

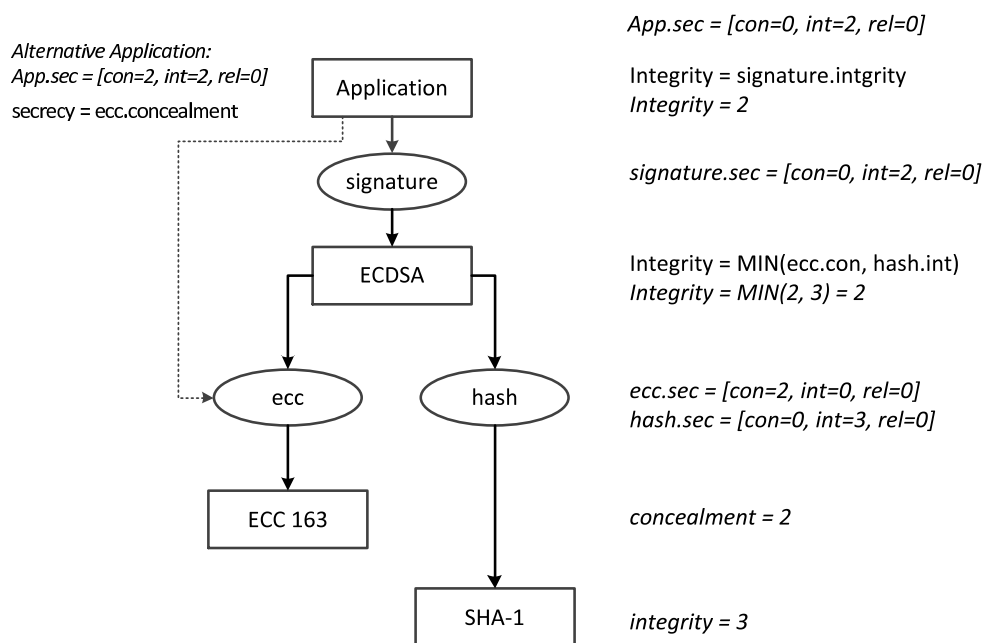


Figure 6.10: Example of the qualitative attribute aggregation: The application uses a signature which is provided by the ECDSA algorithm, which needs an ECC algorithm and a hash. On the right are listed the security properties of the bottom-up security aggregation. On the top right is the evaluation of an alternative configuration in which the application directly accesses the ECC component. It illustrates that QCSM respects the binding of the components.

Example

Figure 6.10 shows the composition graph for an example of a composition of an application that uses a signature. The security related properties are notated right of the graph. While the composition direction is top down, the assessment of security properties has to be evaluated from bottom up.

In the given example the application component uses the signature interface. That is realized with the ECDSA component which uses an ECC interface and a Hash interface. The interfaces are provided by an ECC 163 bit implementation and a SHA-1 implementation, respectively. The dashed line from the application to the ECC interface will be explained later in this example and can be ignored for now.

The security assessment process starts with the leaves of the graph. These components have no structural dependencies and therefore can be assessed statically. In the example the ECC 163 bit module has a medium security strength (concealment = 2)¹. The SHA-1 provides a good integrity strength (integrity = 3). All other security properties are implicitly set to 0. The security vectors are set accordingly and forwarded to the interfaces of the components. There the vectors are used by the ECDSA component. It describes its security strength as integrity attribute, which is computed by the formula $integrity = MIN(ecc.concealment, hash.integrity)$, that is the minimum strength of the concealment provided by the ECC unit and the integrity from the hash. Here it is

¹following the QSC, discussed in Section 4.7.4

the notion that the strength of the signature is weak if the ECC does not provide good cryptographic strength or the used hash has no good integrity. The signature depends on both of them, and the weakness of one of them will compromise the quality of the signature component.

Thus, the integrity of the ECDSA signature in the example is the minimum of 2 and 3, that is $MIN(2, 3) = 2$. Since the other scrutiny attributes are not defined they are set to 0 in the security vector which is forwarded to the signature interface. The security vector (`signature.sec=[con=0,int=2,rel=0]`) finally is forwarded to the application component which directly use the attributes to describe its security properties.

One interesting aspect of this example is that even though the application has a component which provides good concealment, the top application component does not indicate the good concealment attribute anymore. This is caused by the lack of a path connecting the top application with the concealment-providing component. If the application needs concealment it would either look for a signature component that sustains (or tunnels) the concealment attribute, or the application directly uses the ECC interface. The latter alternative is shown in the component graph in Figure 6.10 as dashed component connection between the application and the ECC interface. In such a configuration the application can directly access the ECC security vector with its good concealment property. If the application component is designed accordingly, the concealment property of the ECC component finally describes the concealment of the application, while the integrity property is still described by the signature. However, this is no automatism but the application has to be designed appropriately. That is why the formulas describing the security attributes have to be defined for each component individually, typically by the designer of the component. If the application component only connects the ECC component without actually using it, the ECC module will not improve the concealment properties of the application at all.

Implementation in configKIT

Following the descriptions of the QCSM the integration of the method in the configKIT framework is straightforward.

The component description contains the security properties which are expressed as component properties. The value for the properties can be either set directly (typically for components not using other components), or computed as a relation that works on the security properties of the used interfaces. Thus, we can add the relation

$R : \langle \{I1.Sec, I2.Sec, \dots, Ii.Sec\}, Comp.Sec, f_{sec_c} \rangle$, where $Comp.Sec$ is the internal property of the component, $I1.Sec \dots Ii.Sec$ are the security properties of the used interfaces and f_{sec_c} is the functional description of the mapping. The relation of the ECDSA component in the example of the previous section is

$R : \langle \{ecc.con, hash.int\}, int, "int = MIN(ecc.con, hash.int)" \rangle$

The application requirement process is a direct description of the required quality of the security attributes. It can be a mapping from an application type, as also illustrated in the requirements section of Figure 6.11 (left). This results in the required security attributes as system properties and the constraint relations that require the application

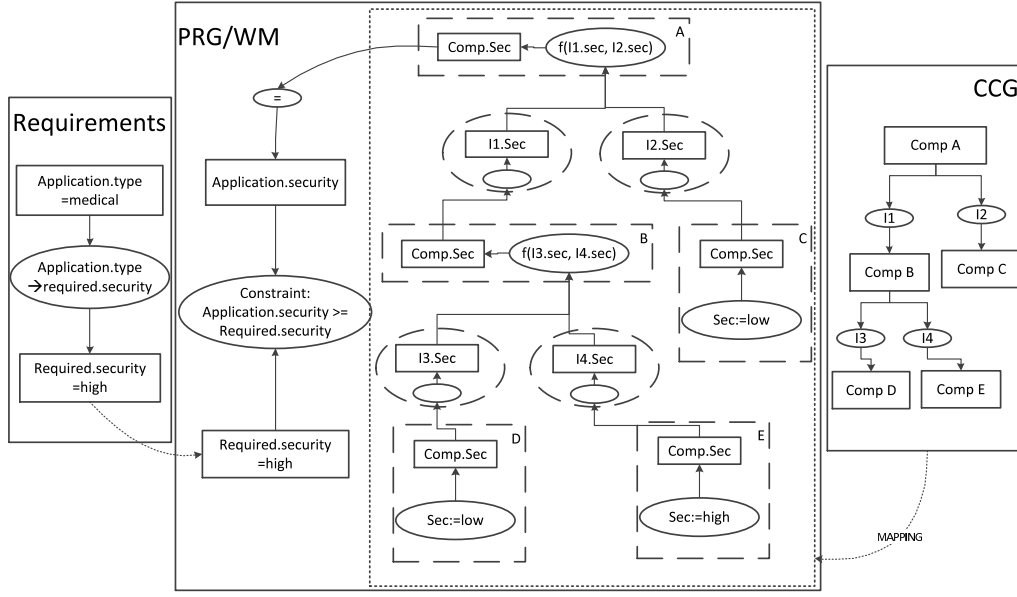


Figure 6.11: Integration of the QCSM security model in configKIT: On the right is the composition consisting of component A (comp A) to E, bound by interfaces. The properties of this component structure are transferred to the PRG. Aggregated from bottom-up the representation of the top component contains the security assessment of the composition (comp.sec). It is finally compared with a constraint that is a derived user requirement.

attributes

$$\begin{aligned}
 R = R \bigcup & \{ \{ req.con, app.con \}, \emptyset, "(req.con > app.con) = false" \} \\
 & \bigcup \{ \{ req.int, app.int \}, \emptyset, "(req.int > app.int) = false" \} \\
 & \bigcup \{ \{ req.int, app.int \}, \emptyset, "(req.rel > app.rel) = false" \}.
 \end{aligned}$$

The added relations are illustrated in Figure 6.11 by the large ellipse in the left part of the PRG.

The composition process consists of the integration of the properties and relations associated with components of the current application selection to the PRG. This follows the standard configKIT composition algorithm. The top component is associated with the application domain so that comp.sec of the component is translated to *App.sec*. These application security properties are compared with the required ones.

Figure 6.11 shows an exemplary flow from the user-given requirement (application.scenario = medical) over the requirement expansion (required.security=high) to the PRG which contains the requirement that the required security level must not be higher than the security level provided by the application.

Since security is multi-dimensional, the security property can be represented as a vector or as a set of values, such as reliability, concealment, integrity. In such a case, each value is implemented with an individual property and with individual relations.

Implementation overhead

The overhead for implementing and modeling the QCSM approach is rather low. For the signature example each used component needed one additional rule to compute the security properties. The description of the application requirements is also slim since it is a direct description of the required attribute qualities. The forwarding and comparison logic inside the PRG only needs one comparison for each attribute, as also illustrated in Figure 6.11. The apparent complexity in the figure results from the composition and property forwarding. It is an automatic process and does not need explicit modeling. Thus it can be concluded that QCSM has a low modeling overhead while it promises to express the security properties with high quality.

6.3.3 Generic Attack-centric Security Model (GASM)

The generic attack centric security model (GASM) presented in this section is based on the attack centric view on the security ontology. An ASM infers a set of possible attacks that can compromise security goals of the system under development. The set is compared with the set of attacks the system is prone to - or alternatively to which attacks the system has resistance. The corresponding model alternatives –white list and black lists– are discussed in this section below. In the model description in this section we only focus on the white list methodology. This means the system has to express the resistance against attack types explicitly in order to be considered as secure.

The model approach is to infer a set of attacks that can violate security goals of the system, and to compare that set with the attacks against which the composed application has resistance.

A composition is secure if for all attacks that can compromise security goals, there exists a property in the composed system that disables that attack. Expressed over the set of system properties this is

$$\forall p_R \in P_{SecReq} : \exists p_C \in P_{Comp} : p_{Comp} \geq p_{Req} \cap sem(p_C) = sem(p_R),$$

meaning that for all members of the set of required attack resistances (P_{SecReq}) it has to exist at least one property in the composition (P_{Comp}) with the same semantics. We can say p_C is the degree of resistance against attack p_R .

Mapping from the requirements: For the mapping process from the application requirements to the list of possible attacks we consider three possible approaches:

Direct selection by the user: For a relatively small amount of supported attacks it is an option that the application designer chooses the attacks the application should resist against. This is done for example for web applications where attacks like SQL-injections, phishing [DTH06], worms and viruses are well-known and understood attacks. There resistance against phishing is a user goal while in fact it is the requirement to resist a specific attack. However, we do not favor the notion of users picking attack patterns. First because users tend to pick the most prominent attack which is not always the most important one. And second because we do

not believe that end users will be able to understand or even weight WSN attack patterns appropriately.

Direct mapping of attacks from the application type: This approach is similar to the application requirements expansion process proposed for the FCSM. Based on user inputs such as the application type a set of attacks is selected by a direct mapping.

Attack trees: In section 4.7.2 attack trees and attack dependency graphs were discussed as means to map attacker goals to actual attacks and to evaluate attributes of the attacks. Attack trees cannot only be used to decompose an attacker goal but the structure also allows to evaluate the attack alternatives.

The attack tree approach is certainly the most interesting of the three choices. However, due to its complexity it will be discussed in detail in the following section. The generic methodology discussed in this section focused on the direct mapping of the application.

Mapping from the composition: Similarly to the features of the FCSM, the attacks a component provides resistance to, are properties of the components. The properties are aggregated so that the application has a set of attack resistances.

Considered a component C uses interfaces I_1, I_2, \dots, I_n . Each interfaces propagates the set of properties of the implementing component $P_{I_1}, P_{I_2}, \dots, P_{I_n}$, respectively. Then the properties P_{CC} of the composed component C is the union of

$$P_{CC} = P_C \cup P_{I_1} \cup P_{I_2} \cup \dots \cup P_{I_n} = P_C \cup \bigcup_{i=1}^n P_{I_i}$$

It means that the set of the attack resistances of the composed component C is the aggregation of all attack resistances the component C already provides without composition plus the resistances provided by the used components.

Influence of the environment: The effect of environmental properties on the robustness of components against a specific attack is part of the component description. Similar to the methodology in the FCSM approach a component must be able to determine whether it provides attack resistance in context of the system environment. Similar to the FCSM these conditions are described in the meta-information of the components.

The Attack Checklist View

The approach GASM can be considered as a checklist model. Such a checklist approach can be implemented either based on white lists or on black lists.

In the white list methodology, sometimes referred as positive security model [OWA11], all applications are assumed to be prone to attacks unless they explicitly state that they have a resistance. In a black list approach an applications is assumed to resist all sorts of attack unless a vulnerability is stated.

White lists typically are more restrictive than black lists since a selection has to be justified. In contrast, when applying black list de-selecting a component has to be justified. This means the result of the white list approach are compositions which are secure with some significance, while the results following the black list methodology still

contain insecure compositions. That is why basically white lists are favored to foster security assurance.

While on the component view the decision for the positive list is reasonable, the composition rules are uncertain. Considered an Application uses two components A and B. Component A is prone to an attack, while component B has resistance. What is the resistance of the application? The following table shows the results following the composition methodology described above:

	comp. A	comp B	App
White List		+	+
Black List	-		-
complete descr.	-	+	?

Thus, if we use the white list approach, the application is secure if component B has resistance. The black list approach assesses the application as insecure because component A is insecure. Considered A is a radio and B is a cryptographic module, and the investigated attack is an eavesdropping attack. Then the positive list approach is the right choice, since we assume that the positive properties of the cipher are used to harden the radio. If the attack is a buffer overflow attack and A is vulnerable and B is safe, certainly the whole application is vulnerable.

Obviously the most reliable option is to integrate the decision in the application module similar to the QCSM methodology. However in our view it is not reasonable to assume a component developer is aware of all potential attacks that could apply to the modules used by the component. Ultimately that would mean a designer of a middleware component has to be aware of potential attacks on the nodes' hardware. As solution we require attack types to define whether they follow the positive list or the negative list approach. It affects the way properties are aggregated in the component composition.

Implementation in configKIT

The configKIT framework does not favor black lists or white lists. Each property p in the composition process can be constrained to be enabled ($p=\text{true}$), disabled ($p=\text{false}$), not enabled ($((p=\text{true})=\text{false})$), or not disabled ($((p=\text{false})=\text{false})$), while typically white list require the requested property to be enabled, and the black list approach requires the property not to be disabled.

Beside the optional black list approach, the integration of the attacks in the GASM in configKIT corresponds to the integration of features in the FCSM: Attack resistances are properties of components that depend on the environment.

The application requirement definition process is a normal requirement expansion, while for attacks the pull methodology appears to be suitable. For the FCSM the push approach was favored in order to avoid inconsistent feature combinations. There the application pointed on the features (or countermeasures) it needed to be secure. Countermeasures that inherently will be activated for specific systems bear the risk of inconsistencies due to the possibility of two or more very similar features that will be activated in parallel.

With the attack-centric view such inconsistencies do not occur because two very similar attacks still need to be tackled independently. Therefore attack properties can be activated based on their inherent information. The modeled attack 'knows' for which environments or applications it is a threat.

Thus we can define an attack as a property P_A as function of m environmental properties $P_{E_1}, P_{E_2}, \dots, P_{E_m}$

$$R : \langle \{P_{E_1}, P_{E_2}, \dots, P_{E_m}\}, P_A, f_{P_A}(P_{E_1}, P_{E_2}, \dots, P_{E_m}) \rangle$$

The composition process is a normal configKIT composition process. It is the goal to satisfy all constraints, i.e. the constraints injected by the application requirement definition process.

Implementation Overhead

The implementation overhead for the GASM approach is rather low for components and comparison logic. Due to the possibility to describe attacks directly with pull relations, omitting a pushing intermediate application as required for the FCSM, the structural overhead of the requirement definition and requirement expansion could be improved.

6.3.4 Attack-Tree-based Attack-centric Security Model (ATASM)

The attack-centric security model described in this section is a refinement of the generic attack-centric model presented in the previous section. There we realized that most challenging aspect of the GASM is the deduction of possible attacks. In this section an approach is described that manages possible attacks bases on the attack tree approach.

Attack trees describe the alternatives an attacker has to fulfill an attacker goal. The root of the tree structure is the attacker goal. Children of a node are refinements of the attacker goal. Leaves of the tree are actual attacks that cannot be further refined. Siblings of a node can be either optional or conjunctive. The latter requires all siblings to be fulfilled to achieve the goal. For example to extract data from the memory of a sensor node an attacker has to be able to access the node physically and be able to read the memory

Attack trees can be applied as tool to map attacker goals. Then the leaves of the tree define the set of attacks the application should resist against. However, a system does not need to be resistant against all attacks of the set to be secure. In the example of the data extraction it is sufficient if the system does not permit an attacker to physically access the nodes OR if it is technically not feasible for an attacker to read the memory of the nodes.

Attack Tree Integration in configKIT

Attack trees in configKIT are represented in the PRG as relations over properties, while the attacks are represented by properties. The relations are formulas, according to the following grammar:

$$\begin{aligned} G &\rightarrow \text{Name} \text{ ':' } \text{opt} \mid \text{Name} \text{ ':' } \text{nopt} \mid \text{Attack} \\ \text{opt} &\rightarrow \text{'or'} \text{ ' ('params')} \\ \text{nopt} &\rightarrow \text{'and'} \text{ ' ('params')} \\ \text{params} &\rightarrow G \mid G \text{ ' ; ' } \text{params} \end{aligned}$$

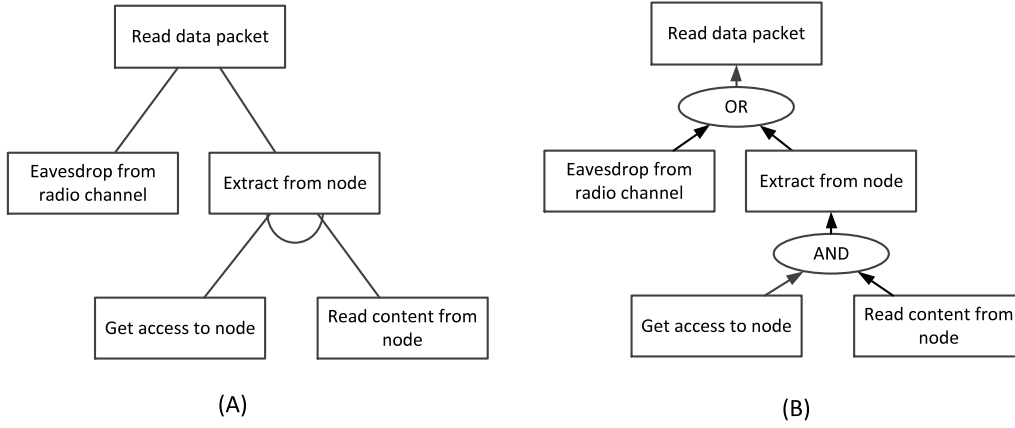


Figure 6.12: Small Attack Tree example for the attacker goal 'read data packet'. an Attacker could either eavesdrop from the radio or extract the data from the node, which requires to get access to the node and be able to read from the node. (a) is a standard attack tree representation, (b) is the representation as PRG.

G is an attacker goal and the terminals of the tree are the attacks (*Attack*). Each G is a node in the tree, while *opt* and *nopt* define the type of the children (optional or non-optional), and *params* is the list of children leading to the nodes G . Each node has a name (*Name*) to support manual engineering.

The small attack tree depicted as Figure 6.12 can be described by the following term:

$$Read_pk : OR(Eavesdrop, Extract : AND(Access, Read_content))$$

The structure as trees or graphs already complies with the properties of the attribute relation graph. To map the attack tree to the configKIT Property-Relation structure we define attacks and attack goals as properties that are true if they are achievable for an attacker.

We define:

- G is a property with name $Name$
- *opt* and *nopt* are relations
- $R_{opt} : \langle params, Name, 'OR' \rangle$ and
- $R_{nopt} : \langle params, Name, 'AND' \rangle$
- *params* is a list of type $Name$ indicating the children of G

For the example the following two relations have to be added:

$$R_1 : \langle \{Eavesdrop, Extract\}, Read_Pk, 'OR' \rangle$$

$$R_2 : \langle \{Access, Read_Content\}, Extract, 'AND' \rangle$$

The resulting PRG structure is shown in Figure 6.12 (B). The arrows in the graph show that the inference direction is from the attacks to the attacker goal. Thus the attacks the system is prone to, define the feasibility of the attacker goals.

This allows to define the key properties of the model:

The **model approach** is to determine the attacks a system is prone to. Attack trees then are applied to evaluate whether the set of possible attack leads to the realization of attacker goals that violate security goals of the system.

A **composition is secure** if the set of attacks the system is prone to does not enable (by propagation through the attack tree) an attacker goal that violates a system goal.

Consider the composed system has the subset A_C of its properties P_C containing the attacks the system is prone to. Further, it is assumed the relations of the attacker tree are part of the PRG structure, so that $A_X \times PRG.R$ results in a set of all properties, i.e all reachable attacker goals. For the given set of security goals P_{SecReq} the system is secure if

$$A_X \times PRG.R \cap P_{SecReq} = \emptyset,$$

meaning that the set of achievable attacker goals must not overlap with the set of security requirements.

Mapping from the requirements: The user defines the security goals which activate the attack tree structures for the corresponding goal.

Additionally the system description provides properties which parametrize the attack trees. These properties can be a description of the attacker and her capabilities.

Mapping from the composition corresponds to the one presented for the GASM approach. Components provide a list of attacks they resist to.

Influence of the environment: The effect of environmental properties on the robustness of components against a specific attack is part of the component description. It corresponds to the GASM approach.

The major difference is the effect of environment and application description on the evaluation in the attack tree. Many attacks require on specific environmental properties. This is expressed in the attack tree. For instance in the example of Figure 6.12 the possible access to the nodes is clearly affected by the environment.

Implementation in configKIT

The actual attack tree structure is part of the PRG of the working model. 6.13 illustrates the data structure. The approach maintains the same attack tree structure independent of requirements and composition. Requirements and composition only parametrize the leaves of the tree. The values are aggregated following the predicate logic upwards in the tree.

The **component description** is identical to the white list GASM approach. That means, components with a resistance against a specific attack define the corresponding property in the PRG.

The **application requirement definition process** is a deduction of system properties that can parametrize the attack trees. It needs additional relations in the requirements expansion process.

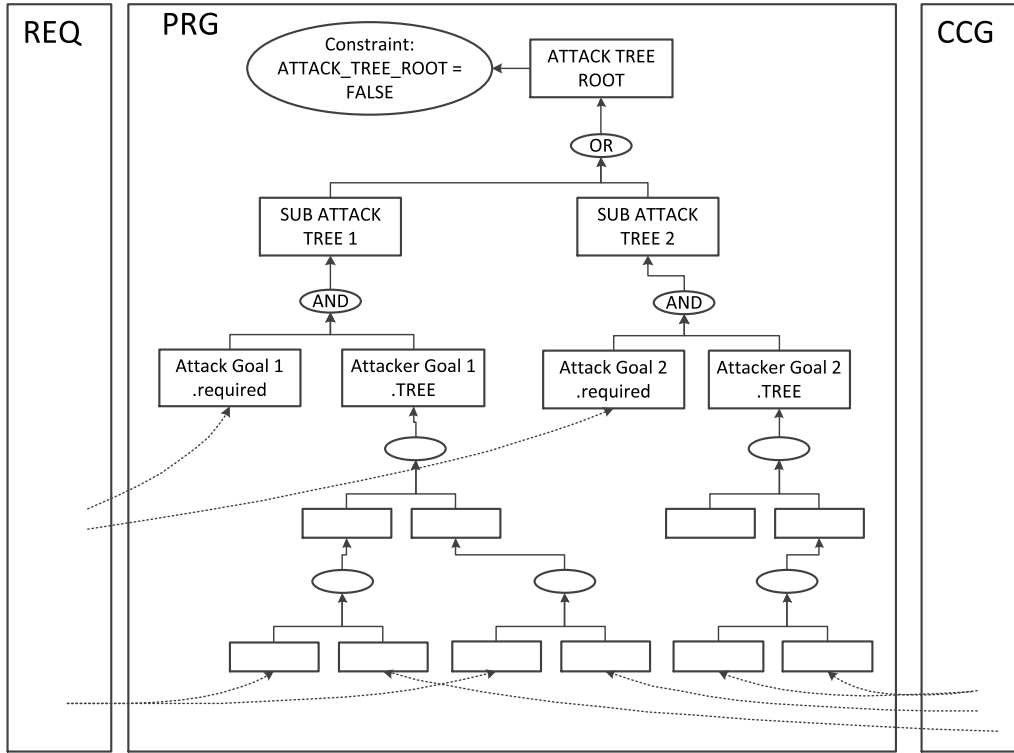


Figure 6.13: Integration of the Attack Tree Approach in configKIT. On the top it combines the security of the system in the single condition ($ATTACK_TREE_ROOT = FALSE$). The leaves of the attack tree are parametrized either from the requirements (defining that a specific attacker goal is a real threat in the scenario) and the CCG (properties from the composed system).

The integration of the attack trees is already addressed earlier in this section. In practice all attack trees are compound in one big attack tree structure, while each sub attack tree is connected with the property that determines whether the goal described by the attack tree is a goal of the system. Translated in the attack-tree notation the rule has the structure:

$$ATTACK_TREE_ROOT : OR(AT1 : AND(Security_Goal_required, Security_Goal_AT), \\ AT2 : AND(Security_Goal2_required, Security_Goal2_AT), \dots)$$

This combines all security properties in one property ($ATTACK_TREE_ROOT$) which is $FALSE$ if no security goal of the system is violated. The attack tree structure is static for all applications. It is just parametrized by the requirements.

Figure 6.13 shows the data structure of the flow of information

The composition process is a normal configKIT composition process once the attack tree data types are defined. During the composition process the only security related constraint is ($ATTACK_TREE_ROOT = FALSE$)

Implementation overhead:

The implementation overhead is dominated by the attack tree structure. Developing and implementing a reliable set of attack trees is a significant work that has to be done by security experts. However, since the structure is static for all applications it only has to be done once.

The overheads for the component descriptions and application description correspond to the required efforts in the GASM approach.

6.3.5 A generic Vulnerability-centric Security Model (GVSM)

Following the notion of the security ontology, attacks need vulnerabilities to be realized. Many attacks even need several vulnerabilities in order to violate the systems' security goals. The vulnerability-centric security model described in this section closes the gap between system and attacks. It extends the ATASM approach introduced in the previous section in a way that the leaves of the attack trees are linked with the vulnerabilities needed to execute the attack.

Vulnerabilities as described in Section 4.1.2 are properties of the system that can be exploited.

The model is to determine the vulnerabilities the composed system has and evaluate whether these vulnerabilities can be exploited in attacks that can violate the systems' security goals.

A composition is secure if for all attacks which can compromise a protection goal at least one required vulnerability is not achievable for the assumed attacker.

Formally it can be defined by the 5-tuple $O: \langle G, A, V, M_G, M_A \rangle$: For the attacker goals G , the attacks A , and vulnerabilities V we can define a mapping $M_G : G \times A$ that is true for $\forall g \in G, \forall a \in A : a$ is an attack that realizes the goal g . Further we define a mapping $M_A : A \times V$ that is true for $\forall a \in A, \forall v \in V : v$ is a vulnerability required to realize a . Then a system S with its set of vulnerabilities V_S is secure if

$$\forall g \in G : \forall a \in A, (g, a) \in M_G : \exists v \in V, (a, v) \in M_A : v \notin V_S$$

This means a system does not need to be free of all sorts of vulnerabilities but only free of vulnerabilities that can be combined to attacks that compromise a security goal of the system.

Mapping from the application requirements: The mapping from the application is an extension of the attack trees described for the ATASM approach. It is the notion that the attacks, i.e. the leaves in the attack tree, are aware which vulnerabilities have to be exploited. It is assumed that each attack can only be realized by a unique set of vulnerabilities, while the same set of vulnerabilities can lead to more than one attack.

Mapping from the composition: The composition mapping is similar to the GASM approach: Vulnerabilities are properties of the components, which may depend on environmental properties.

Similar to the attack-centric models we have the design option between positive lists and negative lists.

positive list: a component or of the system is assumed to be free of vulnerabilities unless they are listed explicitly.

negative list: a component or of the system is assumed to be prone to vulnerabilities unless a resistance is listed explicitly.

The issue has been discussed in detail for the GASM approach. The solution there is also applicable to the vulnerabilities: for each of the vulnerabilities it can be defined which methodology the property aggregation follows.

Influence of the environment: The environment influences the vulnerabilities of the components. The effect of the environment on the application requirements corresponds to the effects described in the ATASM approach: the environment affects the properties of the global attack tree.

Implementation in configKIT

The GVSM approach is similar to the ATASM in many aspects. A major modification is the addition of the vulnerabilities to the attack tree. Both vulnerabilities and attacks are properties. Since attacks know the vulnerabilities they need attack objects pull in the corresponding vulnerabilities by adding the relation:

$$R : \langle \{P_{V_1}, P_{V_2}, \dots, P_{V_m}\}, P_A, f_{P_A}(P_{V_1}, P_{V_2}, \dots, P_{V_m}) \rangle$$

Such pull relations also ensure that each attack is solely realized by one set of vulnerabilities.

The actual composition process is a normal configKIT composition looking for compositions that satisfy the constraint of the ATTACK_TREE_ROOT.

Implementation overhead

The implementation overhead corresponds to the overheads determined for the ATASM approach. The only extension are the added vulnerabilities to the attack trees. The overhead for these relations is rather low. The application definition process and component mapping is identical to ATASM.

The efforts for a component designer for modeling the components may be lower since vulnerabilities are somewhat more tangible and reusable than attacks. However the general efforts are not influenced by this slight modification.

6.4 Conclusions

The five security models presented in this chapter are the result of a new methodology to deduce practical models from a general security ontology.

The notion of this approach is –as first step– to reduce the complexity of the model by focusing on specific views on the ontology. The views on the ontology provide a good abstraction and allow to explain security properties. As second step, the specific views were further refined with focus on deduction of requirements and composability in the component model which finally can be integrated as security models in configKIT.

Applying this process and using the security ontology for WSNs we could identify three views:

The Countermeasure-centric View, which assesses the security of a system based on the countermeasures the system possesses.

The Attack-centric View, which assesses the security of a system based on the resistance against attacks.

The Vulnerability-centric View, which evaluates the system on the existence of vulnerabilities needed for the attacks.

Based on these views, five specific practical security models were derived:

- A Quality-based Countermeasure-centric Security Model (QCSM)
- A Feature-based Countermeasure-centric Security Model (FCSM)
- A generic Attack-centric Security Model (GASM)
- An Attack Tree based Attack-centric Security Model (ATASM)
- A generic Vulnerability-centric Security Model (GVSM)

The two models for each the attack and countermeasure view exemplify that the same view can be implemented differently. For countermeasures one model uses explicit features, while QCSM applies qualities. The difference is that users typically can express the required security qualities (as in QCSM) but do not know which security features are needed as it is needed for FCSM. On the other side the assessment logic needs to be significantly more sophisticated if precise qualities should be derived, instead of rather tangible features as for FCSM.

The technically most advanced approach is the vulnerability-centric security model (GVSM). It internally decomposes full attack trees and the vulnerabilities needed for the attacks. Then the feasibility for the exploitation of the vulnerabilities is evaluated based on the composed system. Finally a system is secure if no combination of vulnerabilities can be found that can be exploited for a successful attack in the given scenario.

Even though the five models at the end differ significantly, they still represent the initial security ontology.

In order to evaluate the practical value a thorough evaluation of the implemented models is given in the next chapter.

Chapter 7

Secure In-Network Aggregation as Case Study for configKIT

In-network aggregation (INA), introduced in Section 4.5, is a valuable mechanism to reduce energy consumption in WSNs. However, INA has several security concerns as discussed in that section. The basic problem is that nodes in WSNs cannot necessarily trust their neighbors or the aggregating nodes. That is a particular issue if – as it is the case for INA – the neighboring nodes are assigned with the task to aggregate the sensitive data. Concealed Data Aggregation [GWS05] is a means facilitating data aggregation on nodes that do not necessarily know the content of the data they are computing. The mathematical concepts as well as the benefits and potential drawbacks are complex and not easy to understand.

In this chapter the feasibility of configuring secure INA by means of configKIT and the proposed security models is investigated. Therefore, first, reference data for the six design options of INA are gathered by implementing the approaches in a modular way and simulating the resulting applications. While this integration process is executed manually, the modular implementations with the well-defined interfaces are basis for an automatic integration process later in the design process.

The major focus of this chapter is on the integration and evaluation of secure INA in configKIT. As first step it requires a general integration of the WSN application and the INA components in the configKIT framework. Then the security models presented in the previous chapter are applied to model the security attributes of the INA components. Configuration test runs provide test results that eventually allow to evaluate the results. The practical tests demonstrated that the configKIT framework is able to control the configuration the secure INA. The tests further validated that all investigated security models deliver reasonable results.

7.1 Experiment Setup

In this section the general test setup and the considered INA algorithms and their parameters are defined. The test setup is basically a multi-hop star network topology with one sink. We will use this topology in the simulation later in this chapter to measure the foot-

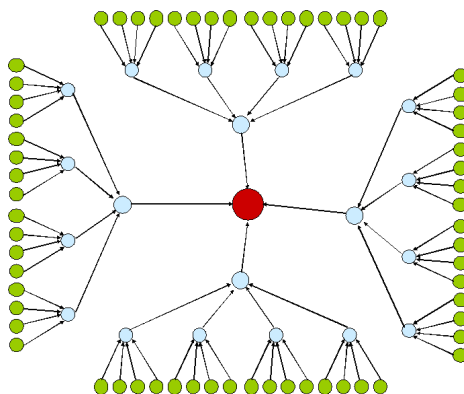


Figure 7.1: Topology of the simulated test network. Red is the base node, blue are aggregation nodes and green are data-providing sensor nodes.

prints of the INA algorithms in practice. The topology is also the network background for the exemplary user selection process discussed at the end of this chapter.

The test environment simulates 85 nodes:

- 1 sink (base station)
- 4 first level aggregation nodes
- 16 second level aggregation nodes (4 for each first level aggregator)
- 64 actual sensors/data provider (4 for each second level aggregator)

The topology of the network is shown as Figure 7.1. For this network an INA application should be deployed. The design options are the INA algorithms discussed in Section 4.5. The following description lists the investigated INA approaches and briefly introduces key aspects and chosen parameters, including the origin of the source codes.

No INA an Implementation without In-Network Aggregation for comparison. It does not provide any encryption. The used source code was written by us.

INA without encryption is an In-Network Aggregation without cryptography. The used source code was written by us.

Hop-by-Hop (HbH) is an INA that transmits encrypted data over the radio but decrypts the data on each aggregation node to perform the aggregation. The used source code was written by us, the applied stream cipher was taken from [KSW04].

CMT is the CMT privacy homomorphism (PH) implementation with (4 bytes) 32 bit data value. The used source code is from the original authors of the algorithm [CMT05].

Domingo-Ferrer (DF) is the Domingo-Ferrer PH implementation with the parameters $D=2$ and $M=32768$. The used source code is from the authors of [GWS05].

Hybrid CDA (hCDA) is the hybrid implementation with DF ($D=2$, $M=32768$) and CMT (16 bit). The combining source code was written by us.

The initial situation can be stated as follows: We have a network and a task and we are looking for the best algorithm that forwards the aggregated values from the sensor nodes to the sink. We are not interested in the precise sensor readings but in an average value, i.e. the sum of the single readings and the number of values.

With respect to the evaluation of the configKIT approach we are interested in the following results:

- Is automatic evaluation possible at all?
- Is security assessment reproducible and reasonable?
- Are the technical parameters reasonable and correct?
- How much overhead is required to describe the model?

7.2 Practical Implementation of CDA

This section describes the steps required to implement the INA schemes applying standard manual methods. As target platform we chose micaZ nodes and the tinyOS operating system. The implementation of the six INA approaches, their footprints and simulation results will deliver the input required for setting up the configKIT component repository.

We wrote an application component that sends values from the sensor nodes to a sink. The values are statically stored on the nodes, so that in the experiment no additional operations for gathering sensor values are required. This means further that the simulation environment ensures that the sensor nodes only have the keys and the function set they need for fulfilling their actual tasks. So for example aggregator nodes do not have keys for de- or encryption, unless required as for the HbH approach. All keys are considered to be distributed before the start of the simulation.

The data collection process is synchronous. That is, in intervals the base node broadcasts a data request that is forwarded by the aggregator nodes to the sensor nodes. The sensor nodes answer in each epoch to their corresponding aggregator node. The second level aggregator nodes forward the result to the first level aggregator nodes which eventually forwards the result to the base node. The sink performs the final aggregation and decrypts the result.

The structure of the implemented application component can be seen in Figure 7.2. Additionally to the CDA algorithm the node program uses the main-component, components for sending and receiving packets, a Timer and LEDs. The CDA algorithm is connected via the *ph* (Privacy Homomorphism) interface. It will be introduced in more detail below. All implemented INA approaches provide the same *ph* interface. Also all three classes of nodes (base station, aggregator, sensor) use the same set of components, while the actual program behavior differs due to the different tasks.

Interface

The shared *ph*-interface consists of the following four functions:

`error_t add(uint32_t *a, uint32_t *b)` is the homomorphic addition operation and will add the two given ciphertexts *a* and *b*.

`error_t configure(uint32_t *conf vector)` initializes the parameters.

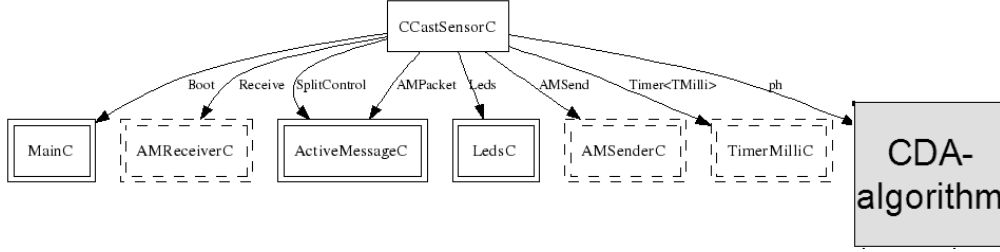


Figure 7.2: General configuration of the test program in TinyOS. The application component CCastSenseC uses interfaces to send, receive, control LEDs and timers, and the CDA algorithm that can be changed in the configuration. The diagram was automatically generated by the nesc compiler. It denotes the name of the used interfaces on the edges.

`command error t decrypt(uint32 t *c)` implements the decryption and returns the plaintext for the ciphertext `c`.

`command error t encrypt(uint32 t a)` implements the encryption operation and computes the ciphertext `Encrypt(a)`.

To harmonize the interfaces of the different modules a very general interface was selected. Configuration parameters and ciphertexts are pointer-based and allow encoding any sort of information.

However, since the source codes are from different authors we already had the first problem harmonizing the interfaces. For all but CMT it was sufficient to slightly adapt the interfaces. For CMT we had to write a wrapper.

The Three Applications

While we have one general architecture in practice we get three different program images covering the specific tasks of the individual roles in the network:

Sensor: it waits for a request from the sink. If such a request arrives, the sensor uses the `encrypt` function of the `ph` interface and sends the result to the radio.

Aggregator: When it receives a request from the sink the aggregator sends it to the next aggregator nodes or the sensors. Then it waits for the answers which will be aggregated using the `add` function of the `ph`-interface. The aggregated value is sent towards the sink.

Sink: After sending an aggregation request it waits for the incoming answers. They will be aggregated (`ph.add`) and finally decrypted (`ph.decrypt`).

All nodes are aware of the topology and of their neighbors. Each image contains exactly the same component configuration, while not every image uses all functions. The sensor does not need addition and decryption. The aggregator does not need decryption and encryption, and the base station does not use encryption and addition. It is however task of the compiler to remove not used functions, because all application images use the same source code files. Thus the component structure of all applications is exactly as shown in Figure 7.2. Beside the application code the only difference is a configuration file (a C-header file) that contains parameters and keys for the operations.

Table 7.1: Overview of memory consumption, traffic figures and needed duty cycles of the implemented aggregation approaches in the 85-nodes-network.

Alg.	Node	Size		Traffic		Duty Cycles per epoch per Node
		ROM [byte]	RAM [byte]	Packets [number]	Total Size [byte]	
No INA No Crypt	Sensor	12468	271	214	4922	109980
	Aggregator	11586	321			230584
	Base	11632	273			319460
INA H2H Encr.	Sensor	15120	468	106	2438	95980
	Aggregator	15694	548			78325
	Base	14548	500			62778
INA without encryption	Sensor	12490	271	106	2438	94914
	Aggregator	11834	321			74016
	Base	11632	273			57993
CMT	Sensor	15460	347	106	2438	115717
	Aggregator	14608	373			71934
	Base	14794	1311			1124207
DF	Sensor	13749	332	106	2862	95443
	Aggregator	12942	374			84287
	Base	12972	354			71108
Hybrid CMT and DF	Sensor	17070	464	106	2862	115906
	Aggregator	15994	478			84270
	Base	16274	1428			1135265

7.3 Simulation Results

The code was compiled for micaz nodes and simulated with Avrora version 1.7.105 [TLP05]. Avrora is also the source of the estimated packet count and sizes and the CPU duty cycles.

Table 7.1 shows the results for each role and algorithm in the 85-nodes network. Column three and four show the memory consumption for ROM and RAM, respectively, column five shows the number of packets tracked in the scenario, followed by total accumulated size of the packets. The right column lists the number of clock cycles for one epoch, i.e. one aggregation process of the entire network. Even though it was not the main objective of this exercise the measurements revealed several interesting results:

It is not surprising that the no in-network-aggregation is the smallest and the hybrid has the largest code size. It was also expected that the base-node of CMT and hCDA (that contains CMT) requires the most dynamic memory, because it has to store each sensor's private key. It is a bit surprising that CMT is larger than DF, despite the algorithm should not be significantly more complicated. This is apparently caused by the more flexible CMT implementation and the required wrapper. However, the actual reason is that CMT needs a StreamCipher that requires significant memory and energy.

The number of packets in our simulation is as expected. These results could have been computed without actual implementation. Even though the network is not severely large, the packet count for the no-INA is already the double of all INA approaches. Packets of DF and hybrid are slightly larger than packets for the other three CDA algorithms because DF (that is also part of hCDA) has a larger ciphertext. This table (as well as the current implementation) does not consider potential packet loss or not-responding

nodes. It can be expected that it would increase the total packet size for the CMT based simulations.

The probably most interesting figures are the computation cycles per epoch. The micaz has roughly 8 million cycles per second. Most approaches have a utilization of about 1 percent. The duty time includes everything the CPU has to perform, i.e. beside the cryptographic operations, also receiving and sending of packets. The actual energy consumption required for radio, board, memory are not considered here. The No-INA approach needs much more computation efforts than expected. The CPU is very busy treating packets in the network stack. Though for the sensor nodes the most packets do not reach the application layer receiving and discarding them needs a lot of additional energy. While the additional energy for aggregator and base node was expected, because they have to handle the additional packets, the actual extend has been surprising. The base nodes handling the CMT algorithm have the highest load. It is caused by the required forward key computation for each connected sensor node. This symmetric key computation is also the reason why the sensors handling CMT need more cycles than the other algorithms.

The results show that all implementations have reasonable memory- and energy consumption. The simulation has also shown that the implementations are correctly working.

7.4 Setting up configKIT for INA

In this section we explain the steps required for setting up configKIT so that at the end it is aware of the properties of the different approaches. First we have to set up the interfaces, then add the modules, then determine and set the properties of the modules so that finally we can add the model of the application component which uses the available interfaces.

Adding the interfaces

The actual process of adding interfaces to the database is quite straightforward. We select the name of the new interface and set whether the interface is an actually implemented interface or a virtual interface. Virtual interfaces can be considered as an abstract class as kind of parent class for a set of specific interfaces. In case of INA, we define INA as the abstract interface. Later this abstract interface can be addressed by an application or another module that wants to use INA. It will be the task of the selection algorithm to find and to assign suitable implementations.

Then we add the actual interfaces: Hop-by-Hop, CDA and hCDA (hybrid CDA). The dedicated specialized interfaces are needed if an application explicitly uses one interface. For example if the application developer wants the application to use Hop-by-Hop then he/she connects the application module with the HbH-interface. Then the other INA-approaches will never be considered as option for this module. The additional classification of interfaces also improves the understandability and maintenance and helps addressing (sub-)sets of interfaces. It implicitly allows a level of fuzziness because applications can pick either direct modules (e.g. HbH), small sets (e.g. CDA), or general classes (e.g. INA).

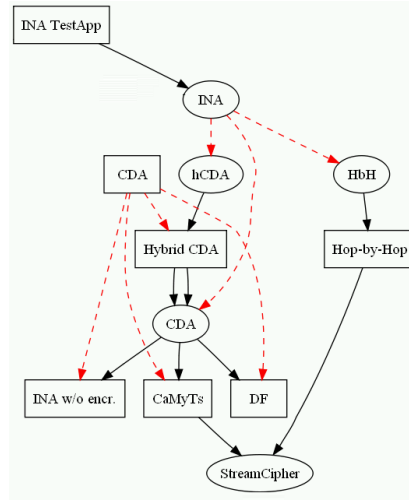


Figure 7.3: Detail of the configKIT component repository focusing on INA components: boxes are modules, ellipses are interfaces, black arrows are uses/provides, red arrows are 'is-relations'.

Adding the modules

In the second step, the modules will be added to the repository. The modules either provide or use available interfaces. In our INA case we add:

- INA without encryption (IWE) provides the CDA interface (we consider no encryption as weakest form of encryption),
- Domingo Ferrer (DF) provides the CDA interface and uses a random interface,
- CMT provides CDA and uses a StreamCipher,
- HbH provides HbH and uses a StreamCipher,
- hybrid CDA (hCDA) provides hCDA and uses two CDAs,

We presume that the other interfaces (Random and StreamCipher) have already been defined.

Similar to the interfaces we can also define abstract classes that can combine a set of modules in a group. In our case we add the abstract module CDA and assign the specific modules (hybrid CDA, INA w/o encr., CMT, DF). With the abstract module CDA we can later address the set of semantically similar CDA-modules. Finally we have to add the application module. An application is just another module that uses other interfaces. Our simple application that does nothing but sending aggregated values, requires a network interface and the INA-interface. All interfaces and modules mentioned so far can be seen in Figure 7.3. It is a detail of the screen shot of the configuration tool zooming in the CDA part.

For each of the components the memory footprint and a qualitative energy assessment is set as component properties. The memory consumption of ROM (program code) and RAM (data) are set in Bytes. As discussed in Section 5.4.5, memory consumption cannot be determined a-priori with certainty. Despite the summation of single components is not perfect, we apply this approach because it is a reasonable estimation that helps to assess the memory consumption.

Table 7.2: Comparison of CDA algorithms (derived from Section 4.5)

	IWE	HBH	DF	CMT	hCDA
Resistance regarding					
ciphertext only	--	++	++	++	++
Chosen plaintext attacks	--	++	-	+	++
Replay attacks	--	++	--	++	++
Malleability	--	++	+	--	+
Malicious Aggregation	--	-	-	+	+
Forged packets	--	++	++	+	++
Captured sensors	<i>o</i>	-	--	+	+

We follow a similar pragmatic approach for the assessment of the energy consumption. As introduced in Section 5.4.5, we assess the energy property in a qualitative scale. The INA approaches certainly can be compared by cycles per epoch, as it is done in Table 7.1. However the data could only be used in direct comparisons and would not have any significance in comparison with routing protocols for example.

With our proposed subjective metric, i.e. '0' for bad and '3' for very good, it is possible to combine different protocols, and we still maintain the option to compare similar protocols directly. Clearly that approach is not perfectly satisfying, but similar to the properties of security and memory it can help to filter suitable or not suitable modules.

7.5 Modeling INA with different Security Models

In the previous section components and interfaces were set up in configKIT and the structural and functional dependencies were modeled. In this section we show how the security models, presented in the previous chapter, can be applied to practically model the INA algorithms and their properties. The gravity center in this chapter is the investigation of the overhead and the practicability of the models. For this purpose we explicitly define what meta-information are needed for the model, the components, and the requirements. These information are the basis for the tests in the next section.

For the rating of the security properties we apply the results of the evaluation of CDA approaches in Section 4.5. The bottom line of that investigation is shown in Table 7.2 which is a condensed version of Table 4.3.

For each of the five security models introduced in the previous chapter we describe:

- The needed meta-information in the components to express the security properties of the INA approaches,
- Required information as part of the requirement expansion, in order to translate the user requirements to technical requirements,
- The needed additional rules inside the Working Model of the composition process.

The exemplary listings and component descriptions are based on the configKIT XML component and model description introduced in Section 5.6.1. Anyway, the syntax shown in the examples is supposed to be readable without knowledge of the full description scheme.

The technical description for each of the models in this section is concluded with a brief discussion of the properties and potential problems.

7.5.1 FCSM

In the Feature-based Countermeasure-centric Security Model (FCSM) countermeasures are modeled as features which are provided by the used components. The basic notion of the approach is that the model compares that list of features with a list of features/-countermeasures the user has defined as part of the requirement definition process.

Component Meta-information: The components contain information about the features and their quality. For the INA case, features are represented by the criteria in Table 7.2.

For example the DF component is shown as Listing 7.1. We only have to define existent features, which reduces the declaration overhead.

Listing 7.1: extract of DF modeled with FCSM

```

1 <relation>ciphertext_attack_resistance := +2</relation>
2 <relation>Malleability_resistance := +1</relation>
3 <relation>Forged_packets_resistance := +2</relation>

```

Requirements description: The FCSM needs an explicit mapping from the user requirements to the features. It is the notion to set a feature to 1 if the user requirements imply it, otherwise to 0

Listing 7.2: extract of FCSM requirement expansion

```

1<relation>ciphertext_attack_requirement=concealment?1:0</relation>
2<relation>plaintext_attack_requirement=concealment?1:0</relation>
3<relation>replay_attack_requirement=integrity?1:0</relation>
4<relation>Malleability_requirement=reliability?1:0</relation>
5<relation>Malicious_Aggregation_requirement=integrity?1:0</relation>
6<relation>Forged_packets_requirement=integrity?1:0</relation>
7<relation>
8   Captured_sensors_requirement=
9   ((environment=hostile) & (integrity|concealment))?1:0
10</relation>

```

The only deviation is the last line. It says resistance against captured sensor nodes is only required if the environment is hostile and either integrity or concealment is required. It is an example how the environment influences the actual system requirements.

Model: The model ultimately adds the constraints for each of the required features, similar to

`ciphertext_attack_requirement<=ciphertext_attack_resistance`

for the attribute describing the resistance against ciphertext attacks. The corresponding rules (one for each feature) are added to the PRG.

Discussion: In case of INA and in presence of the information in Table 7.2 FCSM works satisfying, since the process represents the standard behavior of manual development. The problem is that the features have to be standardized. The developer of a new CDA approach has to describe all existing features. Without the knowledge of the other descriptions an overlap is unlikely. Beside that the implementation of the model for the INA is straightforward and without complications.

Table 7.3: QCSM component security assessment. The \$1 in CMT stands for the Concealment strength of the connected streamcipher. The \$1 and \$2 in the hCDA row represent the strengths of the connected privacy homomorphisms.

Alg.	Integrity	Concealment	Reliability
No INA	1	0	1
H2H Encr.	2	1	1
INA w/o encr	1	0	0
CMT	3	\$1	1
DF	1	2	2
hCDA	MAX(\$1,\$2)	MAX(\$1,\$2)	MAX(\$1,\$2)

7.5.2 QCSM

In the Quality-based Countermeasure-centric Security Model (QCSM), instead of features qualitative security properties are applied. The qualities are assigned to a composition by functional aggregation for each component, bottom-up starting with the components that do not use other components. Inside the model this assessment will be compared with the requirements provided by the user.

Component Meta-information: The security classification can either be set directly, or – in case of components that use other security relevant modules – it can be derived from the used component. Since all INA components but CMT and hCDA are leaves in the component tree, we can assign the attributes directly. As result of the studies in Section 4.5, for hCDA the security attributes can be defined as the maximum of the security attributes of the used components. CMT directly forwards the secrecy parameter of the used StreamCipher (secrecy = \$1.secrecy), so that the secrecy of CMT is determined by the secrecy of the StreamCipher.

Table 7.3 shows the final assessments. \$1 and \$2 represents the security attribute of the first and second used interface respectively. The values are defined as properties of the corresponding components.

Requirements description: The description overhead on requirements and model logic are very low. The requirements are directly set by the user. Similarly, the comparison in the model only needs the comparisons of the (currently three) security attributes. The reasoning for the assessment of the security of the composition needs two lines of code describing the security relations in average.

Discussion: Applying only three simple criteria involves the risk of an overlap in the meaning. In one example (it is described in the results subsection) we saw that reliability can be set by communication protocols as well as by 'actual' security protocols. This can be beneficial as described in the example, but conflicts are possible.

However, this effect must be considered if the security complexity is boiled down to three attributes. The advantages of QCSM comprise the very slim security model and understandable simple requirement inputs. With the described component description strategy the total overhead also for these information is reasonable.

7.5.3 GASM

In the generic attack-centric security model (GASM) feasible attacks are derived from the user requirements and compared with the resistances of the application.

Model: The set of attacks considered in the example is defined by the attacks investigated in Section 4.5 and the criteria in Table 7.2. Thus, we consider the following seven attack types: ciphertext only attack, chosen plaintext attack, replay attack, malleability attack, malicious aggregation attack, forged packet attack, and captured sensor attack.

Component meta-information: The components contain information about the attack resistances as component properties. In the INA example we use the resistances shown in Table 7.2, while at least one '+' means that resistance against the attack is present. In such a case the corresponding attack property is set to FALSE, indicating that there is resistance.

Requirement description: The requirement expansion is identical to the FCSM approach. It also includes the environment-related enabling of attack threats. Also the model and description of the resistances as part of the modules correspond to the implementation presented for FCSM.

Discussion: Technically and logically the GASM approach is very similar to the feature-based FCSM. The difference is that resistances against attacks are considered instead of explicit features. Since in FCSM in fact we considered attack resistances as features, both models are practically identical. In this consequence it is not stringent for all cases. However, the similarities are present, so that eventually it is rather a matter of the view (countermeasures or resistance against attacks) which makes the difference.

7.5.4 ATASM

The most prominent new feature of the attack tree security model (ATASM) are the attack trees as part of the model. Basically the attack trees are a rather complex structure. However, as discussed in Section 6.3.3 the trees only have to be set up once. For the INA example we constructed two attack trees for the assumed two most significant primary attacker goals: to read the content of a packet, and to tamper the content of a packet.

Component Meta-information contain the attacks, the component has resistance against, as properties. This corresponds to the description of the GASM approach. Since the granularity of attacks within the attack tree is finer than for the GASM check lists, also the attack resistances within the components have to be described in more detail. As example, HbH has resistance against concealment-breaking attacks over the radio, but not if the nodes are attacked directly.

The Model is the major difference to the GASM approach, since for the ATASM the attack tree is the means to decide whether a composition is secure. The attack tree is defined as described in Section 6.3.3. It contains one comparable security property which is the root of the attack trees named `ATTACK.TREE.ROOT`. It branches to the (in our case two) actual attack trees: `concealment_tree` and `integrity_tree`.

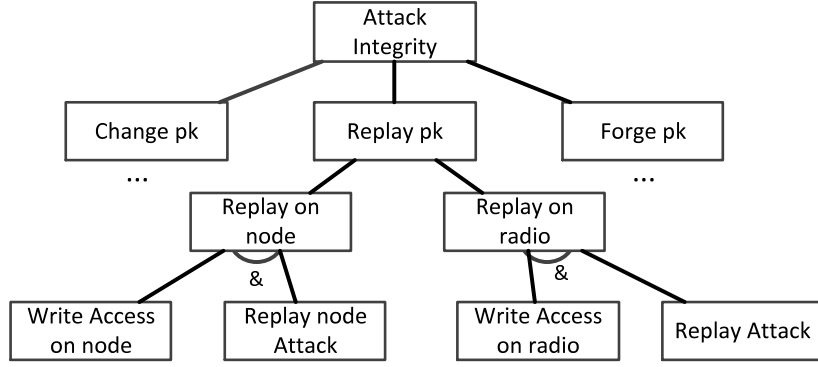


Figure 7.4: Implemented Integrity Attack Tree.

This requirement poses the only security-related constraint of the system: the property `ATTACK_TREE_ROOT` must be *FALSE* (see line 4 in Listing 7.3), meaning there is no security violation. This can be realized by the user, defining concealment and integrity are not required – then the term in line 5 of Listing 7.3 is always *FALSE*. In case a requirement is set, the leaves representing the attacks have to be disabled by properties in the composition in order to fulfill the superior constraint of the attack tree. The `tamper_packet_tree` branches to the three attack goals `replay_packet`, `change_packet_content`, and `forge_packet`. Each of the three attacks can be performed over the radio channel or by accessing the nodes. The resulting attack tree is illustrated as Figure 7.4, the description as part of the property-relation-graph is straightforward.

Requirement description: The task of the requirement description and expansion in the ATASM approach concerns ruling out attacks in the attack tree which either do not violate security requirements or are not feasible for the attacker. In the example we need a mapping from the qualitative requirements to the attacks.

The first mapping rule is already implicit in the attack tree: if a security property is not needed the whole sub tree is disabled (compare line 5 in Listing 7.3). Additionally, specific attacks are disabled based on the environments or the attacker profile. In a friendly (home) environment, the node access attacks are all disabled. We further define that low class attacker will not be able to perform active attacks.

Listing 7.3: Description of the ATTACK TREE ROOT

```

1 <requirement id="s001">
2   <domain>security</domain>
3   <type>ATTACK_TREE_ROOT</type>
4   <relation>=FALSE</relation>
5   <relation>=(concealment & concealment_tree)|(integrity & integrity_tree
6     )</relation>
7   <relation>concealment_tree=read_packet_tree</relation>
7   <relation>integrity_tree=tamper_packet_tree</relation>
8 </requirement>

```

Discussion: ATASM implements the concept of a more complex model, while the needed descriptions for requirements still remain simple. An issue in the mapping from requirements nad component properties to the model is the naming convention. In the case study we had control over requirements, component, and model, so that the consistency of the attack names in these descriptions could be sustained. In practice this issue needs some sort of regulation which still has to be derived.

7.5.5 GVSM

The vulnerability model (GVSM) uses the attack tree model but extends the attacks with vulnerabilities which are needed for an attacker to realize the attack. The challenge for the practical realization of this model is to identify actual vulnerabilities for the given attacks.

Component meta-information The vulnerabilities are described as properties of the components. Based on the security analysis in Section 4.5 we identified the following vulnerability properties:

No INA: readable plaintext = *TRUE*; packets replayable = *TRUE*;

HbH: trusted aggregator = *TRUE*; packets replayable = *FALSE*;
readable plaintext = *FALSE*;

INA w/o Encr: readable plaintext = *TRUE*, trusted aggregator = *TRUE*;
packets replayable = *TRUE*;

CMT: packets malleable = *TRUE*; packets replayable = *FALSE*;
readable plaintext = *FALSE*;

DF: packets replayable = *TRUE*; packets malleable = *FALSE*;
readable plaintext = *FALSE*;

Hybrid CDA: –

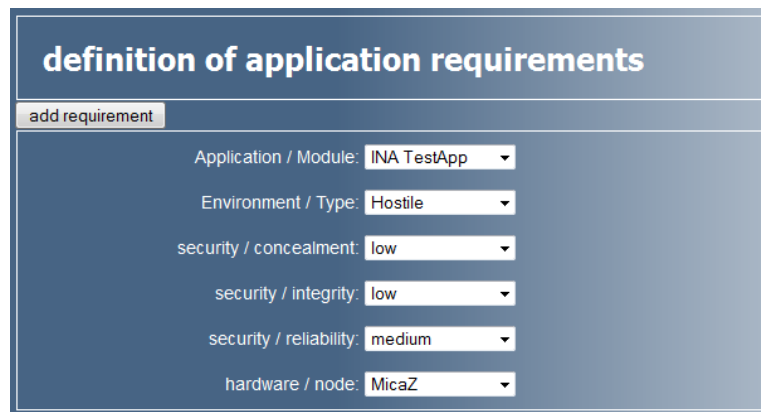
All properties are weak negative, meaning they are *FALSE* unless defined otherwise. Applied on hCDA, which uses CMT and DF, it results in the following initial vulnerability properties: packets malleable = *FALSE*; packets replayable = *FALSE*; readable plaintext = *FALSE*; trusted aggregator = *FALSE*.

Model: The model needs the definition of the vulnerability types and the attachment to the attack tree. In the example we use the attack tree defined in the previous section. The connection of the vulnerabilities to the attacks is simply a relation-based extension of the tree, following the example:

```
node_write_access = attacker_has_access and node_has_debug_port.
```

In this example the `has_access` part of the relation is intended to be disabled by the requirements, the `debug_port` information is provided by the components. The vulnerabilities, directly related to INA are direct extension of existing attacks, while vulnerabilities are exploitable by several different attacks. This is particularly true for the *trusted aggregator* vulnerability which is input to all node-access-related attacks.

Requirements definition is the similar to the ATASM. After loading the attack tree, based on the inputs attacks are disabled that do not fit the attack profile. The vulnerabilities are not explicitly addressed during the requirement expansion.



definition of application requirements

add requirement

Application / Module: INA TestApp ▼

Environment / Type: Hostile ▼

security / concealment: low ▼

security / integrity: low ▼

security / reliability: medium ▼

hardware / node: MicaZ ▼

Figure 7.5: Selection dialog box sets the inputs for the application description

Discussion: The GVSM is the most complex model discussed in this thesis. It extends the already comprehensive attack tree approach by vulnerabilities needed to execute the attacks. Thus it also inherits the problems and the complexity of the ATASM model. Interestingly in the INA example the description overhead in the components is reduced. This is caused by the description of vulnerabilities instead of attacks components. Typically one vulnerability can be exploited by more than one attack, so that the total number of needed description is reduced.

The expressiveness of GVSM is met by no other model. It allows to trace vulnerabilities that are not solved and from this the resulting possible attacks. Thus, for a given configuration it allows to identify the weak spots of the system.

7.6 Configuring INA with configKIT

In the previous section we explained how to set up the repository of configKIT. In this section we show how developers can use configKIT and this way can exploit the stored data to easily configure their WSN-application with INA.

Test Cases

For the test cases we are looking for a setup that allows to compare all security models. In the assumed scenario the user enters requirements of energy, integrity, concealment, reliability as qualities. These inputs can be processed by all models. Additional input is the trust in aggregator nodes.

Figure 7.5 shows an exemplary dialog for the inputs.

Based on the theoretical security assessment and the simulation results we chose a set of test configurations and noted the resulting set of proposed INA approaches. The tests are:

- 1: The user requests an INA application without any additional requirement.

The result, as direct output of configKIT, is shown as Figure 7.6. The diagram is a direct output of the tool and only shows the components. The network function is ActiveMessage and the preferred INA approach is INA without aggregation. Since no security requirements were given, configKIT selected the smallest solution. The

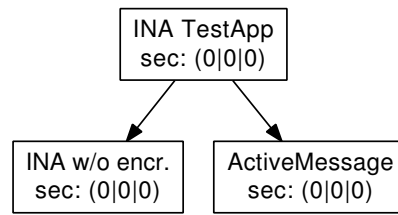


Figure 7.6: Initial configuration of the INA-application: Simple network protocol and simple INA algorithm are small but insecure.

sec: (0|0|0) indicates that no security features are provided. The predicted memory consumption is 11 kBytes and the predicted energy consumption is low.

- 2:** the required concealment is increased, the environment stays trusted
As result HBH encryption is used.
- 3:** additionally the environment is set to hostile, implicating that we cannot trust the aggregator nodes.
Then DF would be the preferred INA approach. CMT and hCDA are alternatives that are also listed, but they are larger and need more energy.
- 4:** additionally integrity requirement is set to medium.
It disqualifies DF as proper solution because its integrity property is too weak. Then CMT with a StreamCipher will be chosen instead.
- 5:** additionally require the reliability feature.
Since CMT does not provide sufficient robustness the only remaining choice is the hybrid CDA which applies both CMT and DF.

Table 7.4 shows the complete assessments and estimated footprints for the different configurations.

Table 7.4: Overview of exemplary inputs, the resulting configurations and their estimated properties. Int, Conc, Rel represent the security attributes Integrity, Concealment, and Reliability.

Test Case	Alg.	Estimated Properties				
		ROM (kB)	Int	Conc	Rel.	Energy
-	No INA	10	0	1	0	0
2	H2H Encr.	13	1	1	0	2
1	INA w/o encr	11	0	0	0	3
4	CMT	15	3	3	0	1
3	DF	13	1	2	0	2
5	Hybrid (secure network)	25	3	3	2	0
5	Hybrid (no secure network)	16	3	3	0	0

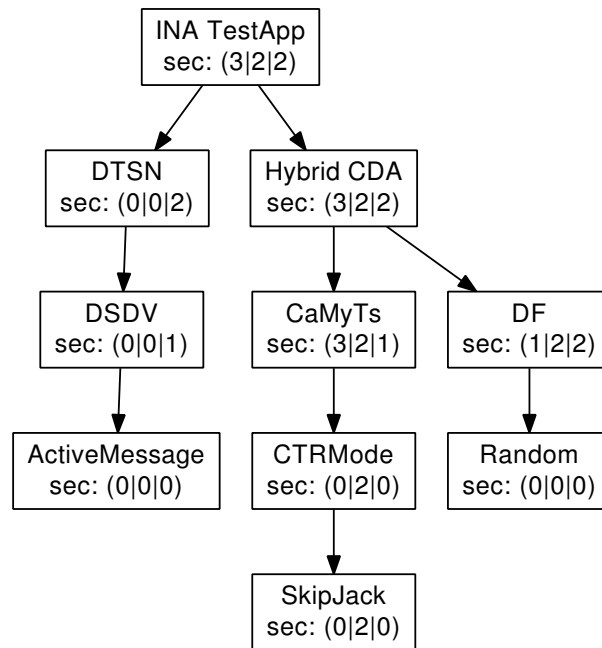


Figure 7.7: Output of the configuration process if all security properties have to be good or better

Results

As example for the output of the selection algorithm Figure 7.7 illustrates the output for test case 5. This case is in particular interesting since the reliability requirement (modeled with QCSM) also triggered the network layers to choose protocols with improved quality. The need for a robust network protocol added the transport and network protocols DTSN/DSDV [MGN07]. This result initially was not anticipated, while it is reasonable for an actual implementation.

In this test case the results indicate that all security requirements are at least 'good' ('2'). It can also be seen that the complexity of the system is already quite high. The total memory prediction is more than 25 kBytes (4kB RAM), so that it is still acceptable for the micaz node.

We executed the tests for each of the five security models. Table 7.5 shows the results. All models led to the same selection of components, which is reasonable since the five models are just different views on the same system including security features and vulnerabilities.

Table 7.5: INA Assessment results for different security models.

	manual	FCSM	QCSM	GASM	ATASM	GVSM
1	INA w/o	INA w/o	INA w/o	INA w/o	INA w/o	INA w/o
2	HBH	HBH	HBH	HBH	HBH	HBH
3	DF	DF	DF	DF	DF	DF
4	CMT	CMT	CMT	CMT	CMT	CMT
5	hCDA	hCDA	hCDA	hCDA	hCDA	hCDA

Finally all models resulted in the expected results. It is no surprise since in context of this thesis we had full knowledge and control over all models and components. So what we could prove is that each model is able to respect all knowledge required to assess INA correctly.

Also concerning the extended properties, the selections and proposed configuration based on the inputs are reasonable, and the predicted memory consumption corresponds to the actual memory consumption determined in 7.1. The security assessments are correct considering our inputs. The energy evaluation is also correct and it allows direct comparisons. Considered that these decisions have been taken based on the fuzzy inputs as shown in Figure 7.5 it is a clear step towards usable configurations of security-related applications and algorithms in the context of WSNs.

7.7 Conclusions

This chapter investigated how to describe the context and interrelations of the INA algorithms in the security models. All necessary was to define the XML files which describe the components, the requirement description, and properties. This could be successfully accomplished for all considered security models, which in first place demonstrates the practicability and flexibility of the configKIT framework. The evidence that eventually all security models resulted in the correct assessment may be a result of the methodology we applied to develop the models. All five models originate in the security ontology introduced in Chapter 3. In this way, with the INA example we could successfully demonstrate all contributions of this thesis.

Indeed, filtering six INA approaches could have been done with significantly less overhead in a smaller database-like approach. However, is has not been the goal of this

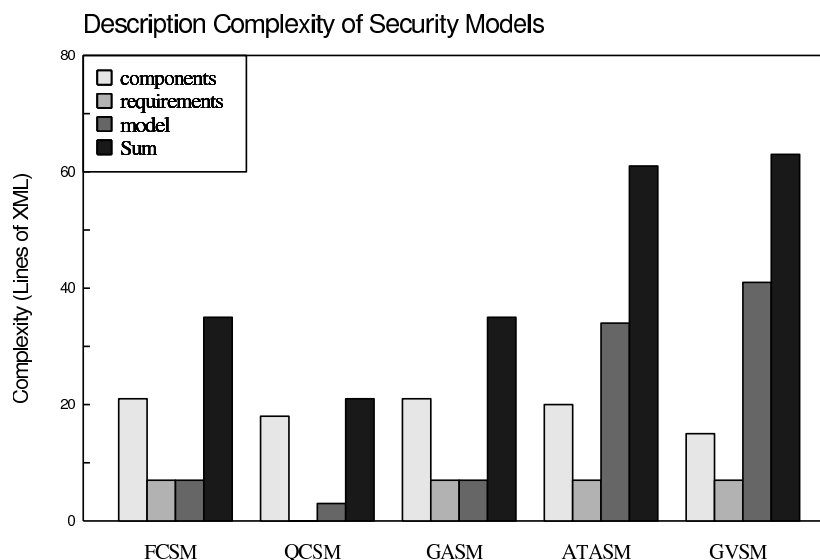


Figure 7.8: Comparison of the complexity to model the security of the INA approaches with the five security models. QCSM has the smallest total overhead due to the minimalistic model. GVSM has the most complex model, but the overhead to model new components is small.

chapter to find an algorithm for a problem, but to demonstrate the effectiveness of the configKIT methodology in general and of the security models in particular. For this purpose the example in this section was chosen to demonstrate how different aspects can be combined to solve a complex design problem. And ultimately we could solve the problem with all proposed security models.

Beside the expressiveness one of our major concerns has been the complexity of the models. Figure 7.8 finally compares the efforts for the five models segmented in the needed lines of XML description for the requirements description, components description, and model description. It is obvious that the total complexity is highest for ATASM and GVSM. However, in particular for GVSM this overweight is caused by the complex model which is fixed. In fact, the overhead for adding new components is the lowest for GVSM, rendering it appealing for developers of components. The model with the lowest total overhead is the quality-based QCSM, due to the straight requirement definition and the slim comparison model consisting of only three rules.

Subjectively, the integration of some models felt more natural than for others. For instance the integration of QCSM just worked and provided good results from the beginning while the more complex models needed several tweaks to result in something sound. In particular the attack tree approaches bear the risk that the knowledge of the properties of the protocols influenced the implementation of the attack trees. In practice and for other protocols it may be necessary to refine the trees.

Chapter 8

Conclusions and Future Work

This thesis presented a tool-supported development flow, named configKIT, that helps users to integrate secured applications in the domain of Wireless Sensor Networks. It consists of a component-based framework that selects and composes configurations of components for WSN applications from user requirements, automatically. During this configuration process the framework employs various models to assess functional and non-functional system attributes, such as security. As result of the design flow, the system can then be assembled and deployed in the network. After a thorough analysis of alternative techniques this approach of compile-time synthesis promises to provide the best possible performance for the design of WSNs.

The first step of the design flow, which was actually implemented, is a novel requirement definition, which bridges the semantic gap between requirements given by users and technical requirements needed for the automatic selection process. In this requirement definition process, first, the user defines the requirements on a domain specific level by selecting and parameterizing attributes provided by a catalog. These high-level requirements are translated to detailed technical requirements utilizing a graph structure of interconnected requirement types. The resulting requirements constitute their direct input for the component-based configuration process.

Core of this component-based configuration process is a selection algorithm that searches the design space of available components for system compositions that are compatible – structurally and semantically – and fulfill the given requirements. The structural properties of the composition are tracked by the component model, while the semantics of the system under development are maintained by the Property-Relation-Graph structure.

The component model provides a rather high-level abstraction of software and hardware modules represented by the components. Interfaces control how the components can be assembled to a complete system. The component model utilizes state of the art concepts such as abstract components and interfaces and a meta-description of properties of the components, which allows the assessment of the behavior of the composed system.

The Property-Relation-Graph structure, which is instantiated as working model of the system, implements a powerful abstraction for the semantics of all aspects of the system. The structure consists of properties, which represent all forms of attributes of the system, and relations which use existing properties, either to define new properties or to implement constraints. Based on this structure, extensive models for system prop-

erties can be designed, as demonstrated with comprehensive models for the estimation of memory and energy consumption.

In this thesis these model capabilities particularly are applied to model and configure security of WSNs. This was first demonstrated by integrating the most advanced security configuration approaches for WSN presented in literature into the approach investigated in this thesis. The results of the original authors could be reproduced, while several limitations of the models regarding expressiveness and flexibility became apparent.

To cope with these issues, new security models were investigated to express the security in WSNs. The models were developed following a novel methodology starting with a holistic security ontology. With specific views on the ontology an additional abstraction could be provided, which allowed explaining security properties with more technical relevance. This finally enabled the possibility to express the needed information from system and requirements in order to decide whether the system under development satisfies the requirements within this model.

As result of the investigation five practical security models could be derived:

- A Quality-based Countermeasure-centric Security Model (QCSM)
- A Feature-based Countermeasure-centric Security Model (FCSM)
- A generic Attack-centric Security Model (GASM)
- An Attack Tree based Attack-centric Security Model (ATASM)
- A generic Vulnerability-centric Security Model (GVSM)

The five models exemplify that even of the same view for the same underlying model, significantly different implementation models can be inferred. For instance, the countermeasures FCSM model uses explicit features, while QCSM applies qualities. The difference is that users typically can express the required security qualities (as in QCSM) but typically cannot name security features as needed for FCSM. On the other hand QCSM's needs a more sophisticated assessment logic to derive precise qualities. However, if this can be achieved –as it is shown with the bottom-up aggregation technique– QCSM is a very promising method to manage security even by non-security experts. The technically most advanced approach is the vulnerability-centric security model (GVSM). It internally decomposes full attack trees and the vulnerabilities needed for the attacks. The feasibility of an exploitation of the vulnerabilities is evaluated based on properties of the composed system. A system is secure if no combination of vulnerabilities can be found that can be exploited for a successful attack in the given scenario. Even though the five discussed models in the end differ significantly, they all could be integrated in the configKIT framework and could be validated to deliver correct results. Embedded in the presented tool chain even complex secure WSN applications can be composed in less than one second.

In the introduction of this thesis we formulated the vision of a design flow that allows end users and domain experts to express their requirements on WSNs – and that results in a deployable configuration of software and hardware. To evaluate to which extend our goal was achieved we applied the techniques developed in this thesis to the non-trivial example of secure in-network aggregation. The evaluation shows that all five practical security models developed in this thesis work correctly and with reasonable model overhead.

Thus, the thesis has significantly contributed to let the vision become reality.

Future Work

While the results are encouraging, many aspects of the work presented in this thesis are subject for further investigations in future work. In this section we present potential future work differentiated between practical application of the presented approaches and further interesting research directions based on the work presented in this thesis.

Future Research Directions include:

Extended security ontology: The security ontology presented in this thesis is generic and intentionally not very complex. An integration of actual business cases, as presented in [EFKW06], for instance from the domain of critical infrastructure could be valuable. The development of actual security models can be executed following the methodology presented in this thesis.

Adding of models to assess new aspects: The Working Model of configKIT was designed for flexible extension with different models to express various aspects of the system. Beside the security models, the current version contains basic models for assessment of memory and energy. These models clearly can be improved. Other models for example to express the quality of service, the consistency of data in the network, or to predict the radio propagation are feasible.

Support of powerful interface descriptions: Currently the component model considers interfaces as static data types. Related work already presents approaches describing interfaces with the complete behavior of the components [LBM⁺02] [CdAH⁺02]. There extensive interface compatibility checking based on a formal methodology and state machines was implemented. An extension of the meta-information in the configKIT component model could be further investigated.

Support of product and feature models: In Section 3.3.3 feature models were introduced as means to provide abstraction from the implementation level. With a model transformation from the implementation-focused component structure in configKIT to the feature-based models, these FMs could replace or at least improve the current dialog-driven requirement definition process. Such a model transformation needs significant research efforts.

Improve the run time of the selection algorithm: The proposed selection algorithm still has non-polynomial complexity. Despite practical optimizations presented in this thesis, it is possible that the composition of large systems can take significant time. To cope with this issue, heuristic search methods such as simulated annealing or genetic optimization are considerable. They do not perform complete search but often finish significantly faster, in particular for large systems. Another improvement of the performance of the algorithm could be achieved with a re-implementation of the algorithm using Logic Truth Maintenance Systems [DK86], constraint solvers or satisfiability (SAT) solvers. These systems are optimized for searching large constraint systems as the PRG. Considered a transformation of the PRG to a SAT can be achieved, it would also provide valuable benchmark results for the implemented selection algorithm.

Support of different data streams: The support of different data streams in the composition architecture, as for instance proposed in [Pfi07], is not explicitly sup-

ported in the current version of configKIT. This feature may be valuable when one application wants to send specific data encrypted and other data unencrypted. Current assessment strategies only assess the application as whole. It can be assumed that maintaining data streams just needs an additional model to describe the new semantics. However, in detail it still has to be studied.

Interface for application programming: Currently configKIT relies on application modules as part of the component repository. In Section 3.1 several programming abstractions were discussed allowing programming the network. A dedicated combination of configKIT and one of the network abstraction approaches could be a valuable contribution. A candidate could be Macrolab [HSH⁺08] whose philosophy seems compatible to configKIT.

Add support for code generation: The current integration process stops after the configuration is generated. It is still manual work to build the eventual system. A further support for the creation of the compilation makefiles with a final compilation step is consequential. On smaller systems with a well-known hardware basis, as for the INA tests, this is not an issue. For larger systems with unknown components and hardware it needs further investigations.

Beside future research directions also future dissemination and practical exploitation is intended:

Public Web service: It is a goal to create a public appearance in the Internet with the configKIT tool chain. It disseminates the approaches developed in this thesis and can foster valuable discussions in the scientific community. It also could be a source for more component models and independent third party case-studies.

Adding components: The value of the toolkit certainly increases with the number of components it maintains. With the easily-understandable component model and the tool-support the barrier for third party developers is rather low to enter new components. Since the component model requires in-depth knowledge about the components, this task cannot entirely be achieved by us.

More test cases: The tests executed with the framework so far were executed by the same person who designed the framework and described the components. Also the evaluated test applications were rather small with a well-known outcome. For a further evaluation of the usability, larger test cases with independent users, developers and framework designers are necessary.

Applying configKIT outside the WSNs world: configKIT with its component meta-model and the PRG model is not restricted to WSNs. What it integrates depends on the components and the applied models. For example in the hardware development for integrated circuits, but also for board layout, the composition of components (modules or blocks) following mostly experience based heuristics is a common development paradigm, which could be supported by configKIT.

Bibliography

- [ABB⁺05] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P.H. Drielsma, P.C. Héam, O. Kouchnarenko, J. Mantovani, et al. The AVISPA tool for the automated validation of internet security protocols and applications. In *Computer Aided Verification*, pages 281–285, 2005. [cited at p. 86]
- [ABV06] G. Acs, L. Buttyán, and I. Vajda. Modelling adversaries and security objectives for routing protocols in wireless sensor networks. In *Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks*, pages 49–58. ACM, 2006. [cited at p. 73, 85, 86]
- [ABV⁺10] D.J. Anthony, W.P. Bennett, M.C. Vuran, M.B. Dwyer, S. Elbaum, and F. Chavez-Ramirez. Simulating and testing mobile wireless sensor networks. In *Proceedings of the 13th ACM international conference on Modeling, analysis, and simulation of wireless and mobile systems*, pages 49–58. ACM, 2010. [cited at p. 16]
- [ADDS91] D. G. Abraham, G. M. Dolan, G. P. Double, and J. V. Stevens. Transaction security system. *IBM Systems Journal*, 30(2):206–229, 1991. [cited at p. 65]
- [Adl00] J. Adler. Computational details of the votehere homomorphic election, 2000. <http://www.votehere.net>. [cited at p. 78]
- [ALRL04] A. Avizienis, J.C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33, 2004. [cited at p. 58, 59]
- [And08] R. Anderson. *Security Engineering - A guide to building dependable distributed systems*. Wiley, 2008. [cited at p. 58, 59, 161]
- [AR06] C. Alcaraz and R. Roman. Applying key infrastructures for sensor networks in CIP/CIIP scenarios. In *Critical Information Infrastructures Security*, volume 4347 of *Lecture Notes in Computer Science*, pages 166–178, 2006. [cited at p. 74, 83, 150, 151, 152, 222]
- [Aßm03] U. Aßmann. *Invasive software composition*. Springer-Verlag New York Inc, 2003. [cited at p. 51]
- [Atm11] Atmel Corp. *Atmel Corp. - Homepage*, 2011. <http://www.atmel.com/>. [cited at p. 25]
- [AY05] K. Akkaya and M. Younis. A survey on routing protocols for wireless sensor networks. *Ad hoc networks*, 3(3):325–349, 2005. [cited at p. 28]
- [Bal01] H. Balzert. *Lehrbuch der Software-Technik*, volume 1. Elsevier-Verlag, 2001. [cited at p. 49]

- [BBD⁺02] R. Barr, J.C. Bicket, D.S. Dantas, B. Du, TW Kim, B. Zhou, and E.G. Sirer. On the need for system-level support for ad hoc and sensor networks. *ACM SIGOPS Operating Systems Review*, 36(2):1–5, 2002. [cited at p. 28, 38]
- [BGH⁺05] P. Buonadonna, D. Gay, J. Hellerstein, W. Hong, and S. Madden. TASK: Sensor network in a box. In *Proc. of the 2nd European Workshop on Sensor Networks (EWSN 2005)*, pages 133–144, Istanbul, Turkey, 2005. [cited at p. 101]
- [BHP06] T. Bures, P. Hnetynka, and F. Plasil. SOFA 2.0: Balancing Advanced Features in a Hierarchical Component Model. In *4th International Conference on Software Engineering Research, Management and Applications*, pages 40–48, 2006. [cited at p. 55]
- [BHUW08] J.M. Bohli, A. Hessler, O. Ugus, and D. Westhoff. A secure and resilient WSN roadside architecture for intelligent transport systems. In *Proceedings of the first ACM conference on Wireless network security*, pages 161–171, 2008. [cited at p. 67]
- [BISV08] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli. The hitchhiker’s guide to successful wireless sensor network deployments. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 43–56. ACM, 2008. [cited at p. 19, 101]
- [BKL⁺07] A. Bogdanov, L. Knudsen, G. Leander, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin, and C. Vikkelsøe. PRESENT: An ultra-lightweight block cipher. *Cryptographic Hardware and Embedded Systems-CHES 2007*, pages 450–466, 2007. [cited at p. 71]
- [BL73] D.E. Bell and L.J. LaPadula. Secure computer systems: Mathematical foundations, 1973. [cited at p. 85]
- [Boe88] B.W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988. [cited at p. 43]
- [Bou07] A. Boulis. Castalia: revealing pitfalls in designing distributed algorithms in WSN. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 407–408, 2007. [cited at p. 34]
- [BRJ05] G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language User Guide, The (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005. [cited at p. 53]
- [BRR08] E.A. Basha, S. Ravela, and D. Rus. Model-based monitoring for early warning flood detection. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 295–308. ACM, 2008. [cited at p. 17]
- [CA05] K. Czarnecki and M. Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In *Generative Programming and Component Engineering*, pages 422–437. Springer, 2005. [cited at p. 48, 49]
- [Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE Symposium on Foundations of Computer Science*, 2001. [cited at p. 91]
- [Cap05] L.F. Capretz. Y: a new component-based software life cycle model. *Journal of Computer Science*, 1(1):76–82, 2005. [cited at p. 44]
- [CCL06] I. Crnkovic, M. Chaudron, and S. Larsson. Component-based development process and component lifecycle. In *Software Engineering Advances, International Conference on*, pages 44–44. IEEE, 2006. [cited at p. 44]

- [CCS06] P. Chandra, B. Chess, and J. Steven. Putting the tools to work: How to succeed with source code analysis. *Security & Privacy, IEEE*, 4(3):80–83, 2006. [cited at p. 93]
- [CdAH⁺02] A. Chakrabarti, L. de Alfaro, T. Henzinger, M. Jurdzinski, and F. Mang. Interface compatibility checking for software modules. In *Proceedings of the 14th International Conference on Computer-Aided Verification*, pages pp. 428–441, 2002. [cited at p. 50, 199]
- [CEKGL08] A. Cuevas, P. El Khoury, L. Gomez, and A. Laube. Security patterns for capturing encryption-based access control to sensor data. In *The Second International Conference on Emerging Security Information, Systems and Technologies*, pages 62–67, 2008. [cited at p. 90]
- [Cha05] A. Chandra. Ontology For MANET Security Threats. Technical report, Jadavpur University, India, 2005. [cited at p. 63]
- [CILN02] R. Crook, D. Ince, L. Lin, and B. Nuseibeh. Security requirements engineering: When anti-requirements hit the fan. In *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*, pages 203–205. IEEE, 2002. [cited at p. 62]
- [CKM00] D.W. Carman, P.S. Kruus, and B.J. Matt. Constraints and approaches for distributed sensor network security (final). *DARPA Project report, (Cryptographic Technologies Group, Trusted Information System, NAI Labs)*, 2000. [cited at p. 74]
- [Cla97] E.M. Clarke. Model checking. In *Foundations of Software Technology and Theoretical Computer Science*, page 54, 1997. [cited at p. 85]
- [CLZ05] E. Cheong, E. A. Lee, and Y. Zhao. Viptos: A graphical development and simulation environment for tinyos-based wireless sensor networks. In *Proceedings of the Third ACM Conference on Embedded Networked Sensory Systems*. ACM, 2005. [cited at p. 34, 39]
- [CM03] W.F. Clocksin and C.S. Mellish. *Programming in PROLOG*. Springer, 2003. [cited at p. 108]
- [CMT05] C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient aggregation of encrypted data in wireless sensor networks. In *MobiQuitous*, pages 109–117. IEEE Computer Society, 2005. [cited at p. 76, 77, 180]
- [CMVH05] S. Cotterell, R. Mannion, F. Vahid, and H. Hsieh. eBlocks: an enabling technology for basic sensor based systems. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, pages 58–es, 2005. [cited at p. 34, 39]
- [CND09] V. Cionca, T. Newe, and V. Dădărlat. A tool for the security configuration of sensor networks. *Journal of Physics: Conference Series*, 178, 2009. [cited at p. 84, 90, 153, 154, 222]
- [DAH01] L. De Alfaro and T.A. Henzinger. Interface automata. *ACM SIGSOFT Software Engineering Notes*, 26(5):120, 2001. [cited at p. 55]
- [DB00] H. De Bruin. A grey-box approach to component composition. *Generative and Component-Based Software Engineering*, pages 195–209, 2000. [cited at p. 50, 51]
- [DF02] J Domingo-Ferrer. A provably secure additive and multiplicative privacy homomorphism. In *ISC '02: Proceedings of the 5th International Conference on Information Security*, 2002. [cited at p. 77]

- [DGV04] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA, November 2004. [cited at p. 27]
- [DK86] J. De Kleer. An assumption-based TMS. *Artificial intelligence*, 28(2):127–162, 1986. [cited at p. 108, 199]
- [DKK07] N. Daswani, C. Kern, and A. Kesavan. *Foundations of Security: What Every Programmer Needs to Know*. Apress, Berkely, CA, USA, 2007. [cited at p. 58]
- [DSVA06] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 29–42, 2006. [cited at p. 27]
- [DTH06] R. Dhamija, J.D. Tygar, and M. Hearst. Why phishing works. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 581–590. ACM, 2006. [cited at p. 168]
- [Dwo01] M. Dworkin. Recommendation for Block Cipher Modes of Operation. Methods and Techniques. Technical report, National Inst of Standards and Technology Gaithersburg MD Computer Security Div, 2001. [cited at p. 71]
- [DY83] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983. [cited at p. 65]
- [EFKW06] A. Ekelhart, S. Fenz, M. Klemen, and E. Weippl. Security ontology: Simulating threats to corporate assets. *Information Systems Security*, pages 249–259, 2006. [cited at p. 63, 199]
- [EJ01] D. Eastlake 3rd and P. Jones. US Secure Hash Algorithm 1 (SHA1), 09 2001. RFC 3174, Informational. [cited at p. 72]
- [EKMM10] M. Eichberg, K. Klose, R. Mitschke, and M. Mezini. Component Composition Using Feature Models. *Component-Based Software Engineering*, pages 200–215, 2010. [cited at p. 48]
- [EMF10] EMF Eclipse Foundation. Eclipse Modeling Framework Project (EMF), 2010. At <http://www.eclipse.org/modeling/emf>. [cited at p. 53]
- [EWKL07] M. Eby, J. Werner, G. Karsai, and A. Ledeczi. Integrating security modeling into embedded system design. In *International Conference and Workshop on the Engineering of Computer Based Systems*, 2007. [cited at p. 86]
- [Fal05] C. Falk. Ethics and hacking: the general and the specific. *Norwich University Journal of Information Assurance*, 1(1), 2005. [cited at p. 64]
- [FC08] A. Francillon and C. Castelluccia. Code injection attacks on harvard-architecture devices. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 15–26, 2008. [cited at p. 69]
- [FF10] S. Faily and I. Fléchaïs. A meta-model for usable secure requirements engineering. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*, pages 29–35. ACM, 2010. [cited at p. 63]
- [FIP99] FIPS. Data encryption standard (DES). Federal Information Processing Standards Publication 46-3, 1999. [cited at p. 70]

- [FIP01] FIPS. Specification for the advanced encryption standard (AES). Federal Information Processing Standards Publication 197, 2001. [cited at p. 70]
- [FIP02] FIPS. Secure Hash Standard, FIPS PUB 180-2, August 1, 2002. [cited at p. 72]
- [Fir05] D.G. Firesmith. A taxonomy of security-related requirements. In *International Workshop on High Assurance Systems (RHAS'05)*. Citeseer, 2005. [cited at p. 58]
- [FRL09] C.L. Fok, G.C. Roman, and C. Lu. Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(3):1–26, 2009. [cited at p. 34]
- [FSLM02] J.P. Fassino, J.B. Stefani, J.L. Lawall, and G. Muller. Think: A software framework for component-based operating system kernels. In *USENIX Annual Technical Conference, General Track*, pages 73–86, 2002. [cited at p. 55]
- [GCN09] A. Grilo, A. Casaca, and M.S. Nunes. The Use of Wireless Sensor Networks for Homeland Security. In *International Conference on Communication, Computer and Power (ICCCP'09)*, 2009. [cited at p. 17]
- [GHH⁺02] R. Govindan, J.M. Hellerstein, W. Hong, S. Madden, M. Franklin, and S. Shenker. The sensor network as a database - technical report 02-771. Technical report, USC Computer Science Department, 2002. [cited at p. 19, 28]
- [GKE04] B. Greenstein, E. Kohler, and D. Estrin. A sensor network application construction kit (SNACK). In *Proc. of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 69–80. ACM, 2004. [cited at p. 35, 100, 139]
- [GKGM05] R. Gummadi, N. Kothari, R. Govindan, and T. Millstein. Kairos: a macro-programming system for wireless sensor networks. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*. ACM, 2005. [cited at p. 32, 38]
- [Gli07] M. Glinz. On non-functional requirements. In *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*, pages 21–26. IEEE, 2007. [cited at p. 46]
- [GLVB⁺03] D. Gay, P. Levis, R. Von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, 2003. [cited at p. 26, 35, 55]
- [Gra04] J. Grand. Practical secure hardware design for embedded systems. In *Proceedings of the 2004 Embedded Systems Conference*. Citeseer, 2004. [cited at p. 65]
- [Gra11] Graphviz. Graphviz – graph visualization software, 2011. website = <http://www.graphviz.org/>. [cited at p. 146]
- [GWMA07] J. Girao, D. Westhoff, E. Mykletun, and T. Araki. Tinypeds: Tiny persistent encrypted data storage in asynchronous wireless sensor networks. *Ad Hoc Networks*, 5(7):1073–1089, 2007. [cited at p. 28]
- [GWS05] J. Girao, D. Westhoff, and M. Schneider. Cda: Concealed data aggregation for reverse multicast traffic in wireless sensor networks. In *IEEE International Conference on Communications*, 2005. [cited at p. 75, 77, 179, 180]
- [GWZ⁺05] V. Gupta, M. Wurm, Y. Zhu, M. Millard, S. Fung, N. Gura, H. Eberle, and S. Chang Shantz. Sizzle: A standards-based end-to-end security architecture for the embedded internet. *Pervasive and Mobile Computing*, 1(4):425–445, 2005. [cited at p. 16]

- [HJM07] M. Hell, T. Johansson, and W. Meier. Grain: a stream cipher for constrained environments. *International Journal of Wireless and Mobile Computing*, 2(1):86–93, 2007. [cited at p. 71]
- [HMCP04] W.B. Heinzelman, A.L. Murphy, H.S. Carvalho, and M.A. Perillo. Middleware to support sensor network applications. *Network, IEEE*, 18(1):6–14, 2004. [cited at p. 37]
- [HP85] D. Harel and A. Pnueli. *On the development of reactive systems*. Weizmann Institute of Science, Dept. of Computer Science, 1985. [cited at p. 41]
- [HPB⁺10] P. Hosek, T. Pop, T. Bures, P. Hnetynka, and M. Malohlava. Comparison of Component Frameworks for Real-Time Embedded Systems. In *Component-Based Software Engineering: 13th International Symposium, CBSE 2010*, 2010. [cited at p. 55]
- [HR06] W. Hasselbring and R. Reussner. Toward trustworthy software systems. *Computer*, 39(4):91–92, 2006. [cited at p. 8, 58, 59]
- [HSH⁺08] T.W. Hnat, T.I. Sookoor, P. Hooimeijer, W. Weimer, and K. Whitehouse. Macrolab: a vector-based macroprogramming framework for cyber-physical systems. In *6th ACM conference on Embedded network sensor systems*, 2008. [cited at p. 32, 38, 200]
- [HSHJ08] T. Heyman, R. Scandariato, C. Huygens, and W. Joosen. Using security patterns to combine security metrics. In *The Third International Conference on Availability, Reliability and Security*, pages 1156–1163. IEEE, 2008. [cited at p. 90]
- [HYN00] R. Housley, P. Yee, and W. Nace. RFC2773: Encryption using KEA and SKIP-JACK. *Internet RFCs*, 2000. [cited at p. 71]
- [IEE90] IEEE. Standard Glossary of Software Engineering Terminology. *IEEE Standard 610.12-1990*, 1990. [cited at p. 45]
- [IHP11] IHP. *IHP - Innovations for High Performance Microelectronics - Homepage*. Innovations for High Performance Microelectronics, 2011. <http://www.ihp-microelectronics.com/>. [cited at p. 23]
- [Jan10] W. Jansen. *Directions in security metrics research*. DIANE Publishing, 2010. [cited at p. 92, 93]
- [Jaq07] A. Jaquith. *Security metrics: replacing fear, uncertainty, and doubt*. Addison-Wesley Professional, 2007. [cited at p. 92]
- [JDG10] G. Jenson, J. Dietrich, and H.W. Guesgen. An Empirical Study of the Component Dependency Resolution Search Space. In *Component-Based Software Engineering: 13th International Symposium, CBSE*, 2010. [cited at p. 127, 139]
- [JHv⁺09] M. Johnson, M. Healy, P. van de Ven, M.J. Hayes, J. Nelson, T. Newe, and E. Lewis. A comparative review of wireless sensor network mote technologies. In *Sensors, 2009 IEEE*, pages 1439–1442. IEEE, 2009. [cited at p. 20]
- [JL97] N. Jorstad and TS Landgrave. Cryptographic algorithm metrics. In *Proceedings of the 20th National Information Systems Security Conference, Baltimore, MD*, 1997. [cited at p. 93]
- [JMR⁺08] C. Jardak, E. Meshkova, J. Riihijarvi, K. Rerkrai, and P. Mahonen. Implementation and performance evaluation of nanoip protocols: Simplified versions of tcp, udp, http and slp for wireless sensor networks. In *Wireless Communications and Networking Conference, WCNC*, pages 2474–2479, 2008. [cited at p. 74]

- [Joh10] Roger G. Johnston. Being vulnerable to the threat of confusing threats with vulnerabilities. *The Journal of Physical Security*, 4 (2):30–34, 2010. [cited at p. 63]
- [JRK⁺09] C. Jarda, K. Rerkrai, A. Kovacevic, J. Riihijarvi, and P. Mahonen. Email from the vineyard. In *5th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities and Workshops, TridentCom*, pages 1–6, 2009. [cited at p. 15]
- [Jür02] J. Jürjens. UMLsec: Extending UML for secure systems development. *UML 2002 The Unified Modeling Language*, pages 1–9, 2002. [cited at p. 86]
- [KAB⁺05] L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis. Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 64–75. ACM, 2005. [cited at p. 17]
- [KGV83] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671, 1983. [cited at p. 141]
- [KHCL07] K. Klues, G. Hackmann, O. Chipara, and C. Lu. A component-based architecture for power-efficient media access control in wireless sensor networks. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 59–72, New York, NY, USA, 2007. ACM. [cited at p. 121]
- [KMP10] J. Krimmling, M. Mahlig, and S. Peter. Test of a Bluetooth Transmission Path for Process Automation - Test einer Bluetooth-Funkstrecke für die Prozessautomatisierung. In *SPS/IPC/Drives 2010, Nuremberg*, 2010. [cited at p. 17]
- [KP04] S. Kumar and C. Paar. Reconfigurable instruction set extension for enabling ECC on an 8-bit processor. *Field Programmable Logic and Application*, pages 586–595, 2004. [cited at p. 25]
- [KPP⁺06] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, A. Rupp, and M. Schimmler. How to Break DES for E 8,980. In *International Workshop on Special-Purpose Hardware for Attacking Cryptographic Systems — SHARCS'06, Cologne, Germany*, <http://www.sharcs.org>, April 2006. [cited at p. 94]
- [Kra97] H. Krawczyk. HMAC: Keyed-hashing for message authentication. *Internet Request for Comments RFC 2104*, 1997. [cited at p. 72]
- [KSW04] C. Karlof, N. Sastry, and D. Wagner. TinySec: a link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 162–175, 2004. [cited at p. 75, 83, 180]
- [KW03] C. Karlof and D. Wagner. Secure routing in sensor networks: Attacks and countermeasures. *Ad Hoc Networks*, 1(1):293–315, 2003. [cited at p. 73]
- [Lau01] K.K. Lau. Component certification and system prediction: Is there a role for formality. In *Proceedings of the 4th ICSE Workshop on Component-based Software Engineering*, 2001. [cited at p. 51]
- [LBD02] T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-based modeling language for model-driven security. *UML 2002 The Unified Modeling Language*, pages 426–441, 2002. [cited at p. 87]
- [LBM⁺02] Á. Lédeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai. Composing domain-specific design environments. *Computer*, 34(11):44–51, 2002. [cited at p. 50, 55, 199]

- [LC02] P. Levis and D. Culler. Maté: a tiny virtual machine for sensor networks. In *ACM Sigplan Notices*, volume 37, pages 85–95, 2002. [cited at p. 28, 32, 37]
- [LDEH02] YW Law, S. Dulman, S. Etalle, and P. Havinga. Assessing security-critical energy-efficient sensor networks. Technical Report TR-CTIT-02-18, Univ. of Twente, TheNetherlands, 2002. [cited at p. 90]
- [LGC05] P. Levis, D. Gay, and D. Culler. Active sensor networks. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 343–356, 2005. [cited at p. 37]
- [LKL07] W. Lee, S. Kang, and D.H. Lee. Product line approach to role-based middleware development for ubiquitous sensor network. *cit*, pages 1032–1037, 2007. [cited at p. 49]
- [LLWC03] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: accurate and scalable simulation of entire tinys applications. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 126–137. ACM, 2003. [cited at p. 34]
- [LM03] T. Liu and M. Martonosi. Impala: a middleware system for managing autonomic, parallel sensor systems. In *Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 107–118, 2003. [cited at p. 32, 37]
- [LMP⁺05] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. Tinyos: An operating system for sensor networks. *Ambient Intelligence*, pages 115–148, 2005. [cited at p. 26]
- [LMPG07] M. Luk, G. Mezzour, A. Perrig, and V. Gligor. MiniSec: a secure sensor network communication architecture. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 479–488, 2007. [cited at p. 83]
- [LPM⁺09] F. Loiret, A. Plsek, P. Merle, L. Seinturier, and M. Malohlava. Constructing Domain-Specific Component Frameworks through Architecture Refinement. In *Software Engineering and Advanced Applications, 2009. SEAA'09. 35th Euromicro Conference on*, pages 375–382. IEEE, 2009. [cited at p. 54]
- [LPZ07] D. Lymberopoulos, N.B. Priyantha, and F. Zhao. mPlatform: a reconfigurable architecture and efficient data sharing mechanism for modular sensor nodes. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 128–137. ACM, 2007. [cited at p. 24]
- [LV01] A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 14(4):255–293, 2001. [cited at p. 71, 93]
- [LWWB05] T. Li, H. Wu, X. Wang, and F. Bao. SenSec: Sensor security framework for TinyOS. In *INSS05: Proceedings of the Second International Workshop on Networked Sensing Systems*, 2005. [cited at p. 83]
- [McL94] J. McLean. Security Models,” in the Encyclopedia of Software Engineering. *J. Marciniak. Ed.*, Wiley. New York, 1994. [cited at p. 85]
- [MCP⁺02] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97. ACM, 2002. [cited at p. 16]

- [MDT10] MDT Eclipse Foundation. Eclipse Model Development Tools (MDT), 2010. At <http://www.eclipse.org/modeling/mdt>. [cited at p. 53]
- [MEM11] MEMSIC Corp. *MEMSIC Corp. - Homepage*, 2011. <http://www.memsic.com>. [cited at p. 21]
- [MFHH05] S.R. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems (TODS)*, 30(1):122–173, 2005. [cited at p. 28, 36]
- [MFMP07] D. Mellado, E. Fernández-Medina, and M. Piattini. A common criteria based security requirements engineering process for the development of secure information systems. *Computer standards & interfaces*, 29(2):244–253, 2007. [cited at p. 87]
- [MGN07] B. Marchi, A. Grilo, and M. S. Nunes. Dtsn: Distributed transport for sensor networks. In *Symposium on Computers and Communications*, 2007. [cited at p. 74, 194]
- [MGW06] E. Mykletun, J. Girao, and D. Westhoff. Public key based cryptoschemes for data concealment in wireless sensor networks. In *IEEE International Conference on Communications*. ICC2006, 2006. [cited at p. 76, 78]
- [MP10] L. Mottola and G.P. Picco. Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Computing Surveys*, 2010. [cited at p. 15, 18, 32]
- [MSW⁺04] A.I. Mørch, G. Stevens, M. Won, M. Klann, Y. Dittrich, and V. Wulf. Component-based technologies for end-user development. *Communications of the ACM*, 47(9):59–62, 2004. [cited at p. 54]
- [NCB10] T. Neue, V. Cionca, and D. Boyle. Security for Wireless Sensor Networks—Configuration Aid. *Advances in Wireless Sensors and Sensor Networks*, pages 1–24, 2010. [cited at p. 84, 152]
- [NHS⁺08] L. Nachman, J. Huang, J. Shahabdeen, R. Adler, and R. Kling. Imote2: Serious computation at the edge. In *Wireless Communications and Mobile Computing Conference, 2008. IWCMC'08. International*, pages 1118–1123. IEEE, 2008. [cited at p. 22]
- [NJOJ05] S. Noel, S. Jajodia, B. O’Berry, and M. Jacobs. Efficient minimum-cost network hardening via exploit dependency graphs. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 86–95. IEEE, 2005. [cited at p. 88]
- [NKA⁺05] L. Nachman, R. Kling, R. Adler, J. Huang, and V. Hummel. The Intel® Mote platform: a Bluetooth-based sensor network for industrial monitoring. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, pages 61–es. IEEE Press, 2005. [cited at p. 22]
- [NS98] D. Naccache and J. Stern. A new public key cryptosystem based on higher residues. In *ACM Conference on Computer and Communications Security*, pages 59–66, 1998. [cited at p. 76]
- [NSSP04] J. Newsome, E. Shi, D. Song, and A. Perrig. The sybil attack in sensor networks: analysis & defenses. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 259–268. ACM, 2004. [cited at p. 69]
- [OKK09] S. Olonbayar, D. Kreiser, and R. Kraemer. Performance and design of IR-UWB transceiver baseband for wireless sensors. In *IEEE International Conference on Ultra-Wideband. ICUWB 2009.*, pages 809–813, 2009. [cited at p. 73]

- [OMG11] OMG. The object management group (omg) <http://www.omg.org/>, 2011. [cited at p. 53]
- [OMKD09] M. Ouedraogo, H. Mouratidis, D. Khadraoui, and E. Dubois. Security assurance metrics and aggregation techniques for it systems. In *Internet Monitoring and Protection, International Conference on*, pages 98–102, Los Alamitos, CA, USA, 2009. IEEE Computer Society. [cited at p. 91, 92, 164]
- [OP10] F.J. Oppermann and S. Peter. Inferring technical constraints of a wireless sensor network application from end-user requirements. In *2010 Sixth International Conference on Mobile Ad-hoc and Sensor Networks*, pages 169–175. IEEE, 2010. [cited at p. 16, 119]
- [OU98] T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. *Lecture Notes in Computer Science*, 1403, 1998. [cited at p. 76]
- [OWA11] OWASP. The open web application security project, 2011. available from <http://www.owasp.org>. [cited at p. 169]
- [PA06] N.R. Prasad and M. Alam. Security framework for wireless sensor networks. *Wireless Personal Communications*, 37(3):455–469, 2006. [cited at p. 90]
- [Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Lecture Notes in Computer Science*, 1592, 1999. [cited at p. 76]
- [Pai00] P. Paillier. Trapdooring discrete logarithms on elliptic curves over rings. *Lecture Notes in Computer Science*, 1976, 2000. [cited at p. 76]
- [Pan10] Panasonic Industrial Europe GmbH. *CR2354 Lithium Battery datasheet*, 2010. Available at: <http://www.panasonic-industrial.com/2464.pdf>. [cited at p. 25]
- [PBVDL05] K. Pohl, G. Böckle, and F. Van Der Linden. *Software product line engineering: foundations, principles, and techniques*. Springer-Verlag New York Inc, 2005. [cited at p. 48]
- [PDDR06] J. Portilla, A. De Castro, E. De La Torre, and T. Riesgo. A modular architecture for nodes in wireless sensor networks. *Journal of Universal Computer Science*, 12(3):328–339, 2006. [cited at p. 23]
- [PER] PERL.ORG. The Perl Programming Language . url=www.perl.org. [cited at p. 146]
- [Pfi07] D. Pfisterer. *Comprehensive Development Support for Wireless Sensor Networks*. PhD thesis, Universität zu Lubeck, 2007. [cited at p. 35, 84, 100, 140, 199]
- [PLP06] K. Piotrowski, P. Langendoerfer, and S. Peter. How public key cryptography influences wireless sensor node lifetime. In *SASN '06: Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks*, pages 169–176, 2006. [cited at p. 25, 74]
- [PLP07] S. Peter, P. Langendoerfer, and K. Piotrowski. On concealed data aggregation for wireless sensor networks. In *CCNC '07: Proceedings of Fourth IEEE Consumer Communications and Networking Conference*, 2007. [cited at p. 77]
- [PLP08] S. Peter, P. Langendoerfer, and K. Piotrowski. Public key cryptography empowered smart dust is affordable. *International Journal of Sensor Networks*, 4(1/2), 2008. [cited at p. 71, 81]
- [PLP09] K. Piotrowski, P. Langendoerfer, and S. Peter. tinyDSM: A highly reliable cooperative data storage for Wireless Sensor Networks. In *Collaborative Technologies and Systems, 2009. CTS'09. International Symposium on*, pages 225–232. IEEE, 2009. [cited at p. 28, 36]

- [PR00] E. Petrank and C. Rackoff. CBC MAC for real-time data sources. *Journal of Cryptology*, 13(3):315–338, 2000. [cited at p. 72]
- [PR07] N. Pham and M. Riguidel. Security assurance aggregation for it infrastructures. In *Second International Conference on Systems and Networks Communications*, 2007. [cited at p. 91]
- [Pre93] B. Preneel. *Analysis and design of cryptographic hash functions*. PhD thesis, Katholieke Universiteit Leuven, 1993. [cited at p. 71]
- [PS08] E. Platon and Y. Sei. Security software engineering in wireless sensor networks. *Progress in Informatics*, 5(1):1–19, 2008. [cited at p. 85]
- [PSC05] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 364–369, 2005. [cited at p. 22]
- [PSL09] S. Peter, O. Stecklina, and P. Langendoerfer. An engineering approach for secure and safe wireless sensor and actuator networks for industrial automation systems. In *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*, pages 1–8. IEEE, 2009. [cited at p. 17]
- [PSL10] K. Piotrowski, A. Sojka, and P. Langendoerfer. Wireless Sensor Networks Can Save Lives-Benefits and Open Issues. *Proceedings-Sensoren und Messsysteme 2010*, 2010. [cited at p. 17]
- [PST⁺02] A. Perrig, R. Szewczyk, JD Tygar, V. Wen, and D.E. Culler. SPINS: Security protocols for sensor networks. *Wireless networks*, 8(5):521–534, 2002. [cited at p. 82]
- [PW01] B. Pfizmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 184–200. IEEE, 2001. [cited at p. 85]
- [PWC10] S. Peter, D. Westhoff, and C. Castelluccia. A survey on the encryption of convergecast traffic with in-network processing. *IEEE Transactions on Dependable and Secure Computing*, pages 20–34, 2010. [cited at p. 75]
- [PZP08] A. Pathak, Q. Zhou, and V. K. Prasanna. Srijan: A graphical toolkit for wsn application development. In *The 4th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS '08)*, 2008. [cited at p. 34, 39]
- [PZV⁺08] S. Peter, M. Zessack, F. Vater, G. Panic, H. Frankenfeldt, and M. Methfessel. An encryption-enabled network protocol accelerator. In *Proceedings of the 6th international conference on Wired/wireless internet communications*, pages 79–91. Springer-Verlag, 2008. [cited at p. 23, 25]
- [RAD78] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. In R. DeMillo, D. Dobkin, A. Jones, and R. Lipton, editors, *Foundations of Secure Computation*, pages 169–180. Academic Press, 1978. [cited at p. 76]
- [RB01] V. Rijmen and P. S. L. M. Barreto. The WHIRLPOOL hash function. World-Wide Web document, 2001. [cited at p. 72]
- [RFMPG06] David G. Rosado, Eduardo Fernandez-Medina, Mario Piattini, and Carlos Gutierrez. A study of security architectural patterns. *Availability, Reliability and Security, International Conference on*, 0, 2006. [cited at p. 90]

- [RG08] V. Rocha and G. Gonçalves. Sensing the world: Challenges on WSNs. In *Proc. of the IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, May 2008. [cited at p. 118]
- [RH07] M. Raya and J.P. Hubaux. Securing vehicular ad hoc networks. *Journal of Computer Security*, 15:39–68, 2007. [cited at p. 64]
- [Riv92] R. Rivest. The MD5 message-digest algorithm. Technical Report Internet RFC-1321, IETF, 1992. <http://www.ietf.org/rfc/rfc1321.txt>. [cited at p. 72]
- [RKJK09] A.M.V. Reddy, A.V.U.P. Kumar, D. Janakiram, and G.A. Kumar. Wireless sensor network operating systems: a survey. *International Journal of Sensor Networks*, 5(4):236–255, 2009. [cited at p. 26]
- [RM95] G. Rugg and P. McGeorge. Laddering. *Expert Systems*, 12(4):339–346, 1995. [cited at p. 47]
- [RM04] K. Römer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, December 2004. [cited at p. 47, 118]
- [Rom85] G.C. Roman. A Taxonomy of Current Issues in Requirements Engineering. *Computer*, 18(4):14–23, 1985. [cited at p. 46]
- [RP09] M. Rounds and N. Pendgraft. Diversity in network attacker motivation: A literature review. In *2009 International Conference on Computational Science and Engineering*, pages 319–323. IEEE, 2009. [cited at p. 64]
- [RPF08] S. Ransom, D. Pfisterer, and S. Fischer. Comprehensible security synthesis for wireless sensor networks. In *Proceedings of the 3rd international workshop on Middleware for sensor networks*, pages 19–24. ACM, 2008. [cited at p. 84, 85]
- [RR07] M. Ringwald and K. Römer. Deployment of sensor networks: Problems and passive inspection. In *Proceedings of the 5th Workshop on Intelligent Solutions in Embedded Systems (WISES '07)*, Madrid, Spain, June 2007. [cited at p. 101]
- [RSA77] R. L. Rivest, A. Shamir, and L. M. Adelman. A method for obtaining digital signatures and public-key cryptosystem. Technical Report MIT/LCS/TM-82, MIT, 1977. [cited at p. 71]
- [RSS06] T. Roosta, S. Shieh, and S. Sastry. Taxonomy of security attacks in sensor networks and countermeasures. In *The First IEEE International Conference on System Integration and Reliability Improvements*, 2006. [cited at p. 64]
- [SA09] R.M. Savola and H. Abie. On-line and off-line security measurement framework for mobile ad hoc networks. *Journal of Networks*, 4(7):565, 2009. [cited at p. 92]
- [Sch96] B. Schneier. *Applied cryptography: protocols, algorithms, and source code in C*. John Wiley & Sons Inc, 1996. [cited at p. 71]
- [Sch99] B. Schneier. Attack trees. *Dr. Dobbs journal*, 24(12):21–29, 1999. [cited at p. 87]
- [Sch03] M. Schumacher. *Security engineering with patterns: origins, theoretical model, and new applications*. Springer-Verlag New York Inc, 2003. [cited at p. 90]
- [Sen11] Sentilla Corp. *Sentilla Corp. - Homepage*, 2011. <http://www.sentilla.com/>. [cited at p. 22]
- [SGM02] C. Szyperski, D. Gruntz, and S. Murer. *Component software: beyond object-oriented programming*. Addison-Wesley Professional, 2002. [cited at p. 49]

- [SHH⁺09] T. Sookoor, T. Hnat, P. Hooimeijer, W. Weimer, and K. Whitehouse. Macrodebugging: global views of distributed program execution. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 141–154. ACM, 2009. [cited at p. 100, 101]
- [SHI10] SHIELDS Project. Shields project - detecting known security vulnerabilities from within design and development tools <http://shields-project.eu/>, 2010. [cited at p. 8]
- [SK97] J. Sztipanovits and G. Karsai. Model-integrated computing. *Computer*, 30(4):110–111, 1997. [cited at p. 52]
- [SLR05] J. Schiller, A. Liers, and H. Ritter. ScatterWeb: A wireless sensor network platform for research and teaching. *Computer Communications*, 28(13):1545–1551, 2005. [cited at p. 22]
- [SMDD10] C. Salinesi, R. Mazo, D. Diaz, and O. Djebbi. Using Integer Constraint Solving in Reuse Based Requirements Engineering. In *2010 18th IEEE International Requirements Engineering Conference*, pages 243–251, 2010. [cited at p. 108]
- [SMK⁺06] E. Sabbah, A. Majeed, K.D. Kang, K. Liu, and N. Abu-Ghazaleh. An application-driven perspective on wireless sensor network security. In *Proceedings of the 2nd ACM international workshop on Quality of service & security for wireless and mobile networks*, pages 1–8. ACM, 2006. [cited at p. 90]
- [Som05] I. Sommerville. Integrated requirements engineering: A tutorial. *Software, IEEE*, 22(1):16–23, 2005. [cited at p. 45]
- [SSD07] H. Soroush, M. Salajegheh, and T. Dimitriou. Providing transparent security services to sensor networks. In *Communications, 2007. ICC'07. IEEE International Conference on*, pages 3431–3436, 2007. [cited at p. 83]
- [Ste46] S.S. Stevens. On the theory of scales of measurement. *Science*, 103(2684):677–680, 1946. [cited at p. 46]
- [SZP⁺03] J. Stankovic, R. Zhu, R. Poornalingam, C. Lu, Z. Yu, M. Humphrey, and B. Ellis. Vest: An aspect-based composition tool for real-time systems. In *Proceedings of the IEEE Real-time Applications Symposium*, 2003. [cited at p. 54]
- [TAD11] TADIRAN Batteries. Datasheet battery type sl-2880, 01 2011. available from <http://www.tadiranbatteries.de/eng/downloads/lithium/pdc06engsl-2880.pdf>. [cited at p. 25]
- [TAGH02] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman. A taxonomy of wireless micro-sensor network models. *SIGMOBILE Mobile Computing and Communication Review*, 6(2):28–36, 2002. [cited at p. 118]
- [TAM11] TAMPRES consortium. TAMPRES - Tamper Resistant Sensor Node, 2011. At <http://www.tampres.eu/>. [cited at p. 73]
- [TCC07a] L. Tobarra, D. Cazorla, and F. Cuartero. Formal Analysis of Sensor Network Encryption Protocol (SNEP). In *Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE International Conference on*, pages 1–6, 2007. [cited at p. 86]
- [TCC⁺07b] L. Tobarra, D. Cazorla, F. Cuartero, G. Diaz, and E. Cambronero. Model checking wireless sensor network security protocols: Tinysec+ LEAP. *Wireless Sensor and Actor Networks*, pages 95–106, 2007. [cited at p. 86]
- [Tex06] Texas Instruments. Cc1100 low-power sub- 1 ghz rf transceiver, 2006. At <http://focus.ti.com/lit/ds/symlink/cc1100.pdf>. [cited at p. 25]

- [Tex07] Texas Instruments Inc. *2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver*, 2007. <http://www.ti.com/lit/gpn/cc2420>. [cited at p. 25]
- [Tex11] Texas Instruments Inc. *MSP430 Family of Ultra-lowpower 16-bit RISC Processors*, 2011. Available at: <http://http://www.ti.com/430wireless>. [cited at p. 25]
- [TLP05] Ben L. Titzer, Daniel K. Lee, and Jens Palsberg. Avrora: scalable sensor network simulation with precise timing. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, 2005. [cited at p. 34, 183]
- [TNMG02] A. Toval, J. Nicolás, B. Moros, and F. Garcia. Requirements reuse for improving information systems security: a practitioners approach. *Requirements Engineering*, 6(4):205–219, 2002. [cited at p. 47]
- [TS98] A. Thomas and P. Seybold. Enterprise JavaBeans Technology. *Patricia Seybold Group, White Paper prepared for Sun Microsystems Inc*, 1998. [cited at p. 51]
- [Ubi07] UbiSec&Sens Consortium. Ubiquitous sensing and security in the european homeland <http://www.ist-ubiseconsens.org/>, 2007. [cited at p. 8]
- [Vit03] P. Vitharana. Risks and challenges of component-based software development. *Communications of the ACM*, 46(8):67–72, 2003. [cited at p. 53]
- [VMD⁺05a] P. Völgyesi, M. Maroti, S. Dora, E. Osses, and A. Ledeczi. Software composition and verification for sensor networks. *Science of Computer Programming*, 56(1-2):191–210, 2005. [cited at p. 34, 55]
- [VMD⁺05b] P. Völgyesi, M. Maróti, S. Dóra, E. Osses, and Á. Lédeczi. Software composition and verification for sensor networks. *Sci. Comput. Program.*, 56(1-2):191–210, 2005. [cited at p. 39]
- [Voa02] J. Voas. Why is it so hard to predict software system trustworthiness from software component trustworthiness? In *Reliable Distributed Systems, 2001. Proceedings. 20th IEEE Symposium on*, page 179. IEEE, 2002. [cited at p. 91]
- [VSL⁺10] P. Völgyesi, J. Sallai, Á. Lédeczi, P. Dutta, and M. Maróti. Software development for a novel WSN platform. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Sensor Network Applications*, pages 20–25. ACM, 2010. [cited at p. 55]
- [VWS⁺06] G. Virone, A. Wood, L. Selavo, Q. Cao, L. Fang, T. Doan, Z. He, and JA Stankovic. An advanced wireless sensor network for health monitoring. In *Transdisciplinary Conference on Distributed Diagnosis and Home Healthcare (D2H2)*, 2006. [cited at p. 16]
- [Wag03] David Wagner. Cryptanalysis of an algebraic privacy homomorphism. In *Information Security, 6th International Conference, ISC 2003, Bristol, UK, Proceedings*, pages 234–239, 2003. [cited at p. 77, 79]
- [WGS06] D. Westhoff, J. Girao, and A. Sarma. Security solutions for wireless sensor networks. *NEC Journal of Advanced Technology*, 59(2), June 2006. [cited at p. 65]
- [Wie99] R. Wieringa. Embedding object-oriented design in system engineering. *Kluwer International Series in Engineering and Computer Science*, pages 287–310, 1999. [cited at p. 41, 46]
- [WN07] K. Walther and J. Nolte. A flexible scheduling framework for deeply embedded systems. In *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops - Volume 01, AINAW '07*, pages 784–791, 2007. [cited at p. 26]

- [Wor04] World Wide Web Consortium. Extensible markup language (XML) 1.0 (third edition), W3C recommendation. Technical report, W3C, 2004. [cited at p. 145]
- [WS04] A.D. Wood and J.A. Stankovic. A taxonomy for denial-of-service attacks in wireless sensor networks. *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, pages 739–763, 2004. [cited at p. 69]
- [WSA10] WSA4CIP Consortium. Wireless sensor networks for critical infrastructure protection <http://www.wsan4cip.eu/>, 2010. [cited at p. 8, 17]
- [WWAD90] S. H. Weingart, S. R. White, W. C. Arnold, and Glen P. Double. An evaluation system for the physical security of computing systems. In *Proc. of the Sixth Annual Computer Security Applications Conference*, December 1990. [cited at p. 65]
- [WY05] X. Wang and H. Yu. How to break md5 and other hash functions. In *EURO-CRYPT*, pages 19–35, 2005. [cited at p. 72]
- [WYY05] X. Wang, Y. L. Yin, and Hongbo Yu. Finding collisions in the full sha-1. In *CRYPTO*, pages 17–36, 2005. [cited at p. 72]
- [XG03] Q. Xue and A. Ganz. Runtime security composition for sensor networks (SecureSense). In *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, volume 5, pages 2976–2980, 2003. [cited at p. 83]
- [YB97] J. Yoder and J. Barcalow. Architectural patterns for enabling application security. In *Fourth Conf. Pattern Languages of Programming (PLoP)*, 1997. [cited at p. 90]
- [YWM05] L. Yu, N. Wang, and X. Meng. Real-time forest fire detection with wireless sensor networks. In *Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on*, volume 2, pages 1214–1217. Ieee, 2005. [cited at p. 17]
- [ZHY⁺06] G. Zhou, C. Huang, T. Yan, T. He, J.A. Stankovic, and T.F. Abdelzaher. MMSN: Multi-frequency media access control for wireless sensor networks. In *IEEE Infocom*, 2006. [cited at p. 73]
- [ZSJ03] S. Zhu, S. Setia, and S. Jajodia. LEAP: efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 62–72, 2003. [cited at p. 82]
- [ZSJ06] S. Zhu, S. Setia, and S. Jajodia. LEAP+: Efficient security mechanisms for large-scale distributed sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 2(4):500–528, 2006. [cited at p. 82]
- [ZSLM04] P. Zhang, C. Sadler, S. Lyon, and M. Martonosi. Hardware design experiences in ZebraNet. In *Proc. of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2004. [cited at p. 16, 119]

Used Abbreviations

ASIC	A pplication S pecific I ntegrated C ircuit
CBD	C omponent B ased D evelopment
CCG	C omponent C omposition G raph
CDA	C oncealed D ata A ggregation
CMT	C astellucia M ykletun T sudik privacy homomorphism
COTS	C ommercial O ff T he S helf
CRG	C omponent R epository G raph
DF	D omingo F errer privacy homomorphism
DL	D iscrete L ogarithm
ECC	E lliptic C urve C ryptography
ECPM	E lliptic C urve P oint M ultiplication
ETE	E nd T o E nd
FPGA	F ield P rogrammable G ate A rray
FM	F eature M odel
FU	F unctional U nit
GUI	G raphical U ser I nterface
HAL	H ardware A bstraction L ayer
HBH	H ob B y H op
hCDA	h ybrid C oncealed D ata A ggregation
INA	I n N etwork A ggregation
KMS	K ey M anagement S cheme
MAC	M edium A ccess C ontrol
MDD	M odel D riven D evelopment

MIC	M essage I ntegrity C ode (Message Integrity Code)
OOP	O bject O riented P rogramming
OS	O perating S ystem
PH	P rivacy H omomorphism
PKC	P ublic K ey C ryptography
PRG	P roperty R elation G raph
QoS	Q uality of S ervice
SAP	S ervice A ccess P oint
VM	V irtual M achine
WM	W orking M odel
WSN	W ireless S ensor N etworks
WSAN	W ireless S ensor and A ctuator N etworks
XML	E xtensible M arkup L anguage

List of Figures

1.1	Scope of the thesis	8
1.2	Structure of the thesis.	12
2.1	Structure of a wireless sensor network	16
2.2	Commercial off-the-shelf sensor nodes	21
2.3	Application specific sensor nodes	22
2.4	Photo of a stacked sensor node platform	23
2.5	Typical hardware architecture of a sensor node	24
2.6	Software architecture of a sensor node	28
3.1	Taxonomy of WSN programming methods	32
3.2	Taxonomy of WSN programming tools	34
3.3	Macrolab system model	38
3.4	“Magic Square” of system engineering	41
3.5	Waterfall Model	42
3.6	Adapted “magic square” with component reusability	43
3.7	Y-process model for component-based development	44
3.8	General component-based Process Model	45
3.9	Feature Model as Abstraction for Product Configuration.	48
3.10	Graphical representation of components and interfaces	52
3.11	Roles in the component-based development	54
4.1	Classification of security-related terms	60
4.2	Ontology of the security terms used in this thesis.	61
4.3	Context of attacker motivation	67
4.4	Principle of in-network aggregation	75
4.5	Different architectures of integrating security toolboxes	82
4.6	Attack Tree example	88
4.7	Combination of attack trees and dependency graphs.	89
5.1	Stages of our envisioned partly automated WSN design process.	98
5.2	Requirement definition process	99
5.3	Code generation process	100
5.4	ConfigKIT Design Approach	101
5.5	Major data structures needed for the configuration process.	102

5.6	Graphical notation of Properties and Relations	106
5.7	Example for a Property-Relations-Graph	107
5.8	Meta-model of the Property-Relations	107
5.9	Symbolic representation of the example 5.7.	111
5.10	Dynamics influencing the PRG in the Working Model.	112
5.11	Example for a circular property realtion.	113
5.12	Implicit handling of multiple property definitions.	114
5.13	Requirement definition process	116
5.14	Requirement design spaces	118
5.15	Static and dynamic data structures on components	120
5.16	Graphical notation of static interface and components	122
5.17	Example graphical notation of single component	123
5.18	Example of a small Component Repository Graph (CRG) with symbolic components and interfaces.	124
5.19	Meta-Model of the static interfaces and components.	125
5.20	Examples for valid and invalid component-interface-connections.	127
5.21	Example for local properties	129
5.22	Property Models as Part of the Working Model	130
5.23	Structure of the component selection process	132
5.24	Top level system composition	133
5.25	Class diagram of the implemented configKIT selection algorithm.	142
5.26	Representation of a relation as a parse tree	143
5.27	Graphical representation of the XML component scheme.	145
5.28	The WSN engineering tool chain	146
5.29	GUI of the components database editor.	147
5.30	User interface for entering the application requirements.	147
5.31	Screen shot of the result page of the configuration process.	148
6.1	Security model assessment architecture	149
6.2	Component Composition Design Space diagram for the KMS example	151
6.3	Component Repository for the Cionca example	155
6.4	Full Ontology as introduced in Section 4.1.2	156
6.5	Illustration of the security model development methodology	157
6.6	Countermeasure Centric Ontology	158
6.7	Vulnerability Centric Ontology	158
6.8	Attack Centric Ontology	159
6.9	Security model views	159
6.10	Example of the qualitative attribute aggregation: Signature	165
6.11	Integration of the QCSM security model in configKIT	167
6.12	Small Attack Tree example	172
6.13	Integration of the ATASM Approach in configKIT	174
7.1	Topology of the simulated test network	180
7.2	Configuration of the test program.	182
7.3	Detail of the configKIT component repository focusing on INA components	185
7.4	Implemented Integrity Attack Tree.	190
7.5	Selection dialog box sets the inputs for the application description	192

7.6	Initial configuration of the INA-application: Simple network protocol and simple INA algorithm are small but insecure.	193
7.7	Output of the configuration process if all security properties have to be good or better	194
7.8	Complexity of Security Models	195

List of Tables

2.1	Data of Off-the-shelf sensor nodes.	21
3.1	Overview on Scales of Measurement.	47
3.2	UML four layer meta-model	53
4.1	Four-class attacker classification.	66
4.2	Examples of solutions for security goals, possible attacks and possible countermeasures.	70
4.3	Comparison of CDA algorithms	81
4.4	Qualitative Security Classification Metric	94
5.1	Operations of the cgraph structure.	108
6.1	KMS CRITIS Guidelines - extract from [AR06]	151
6.2	Security protocol assessment as presented in [CND09]	153
7.1	Overview of memory consumption, traffic figures and needed duty cycles of the implemented aggregation approaches in the 85-nodes-network.	183
7.2	Comparison of CDA algorithms (derived from Section 4.5)	186
7.3	QCSM component security assessment	188
7.4	Overview of exemplary inputs, the resulting configurations and their estimated properties	193
7.5	INA Assessment results for different security models.	194