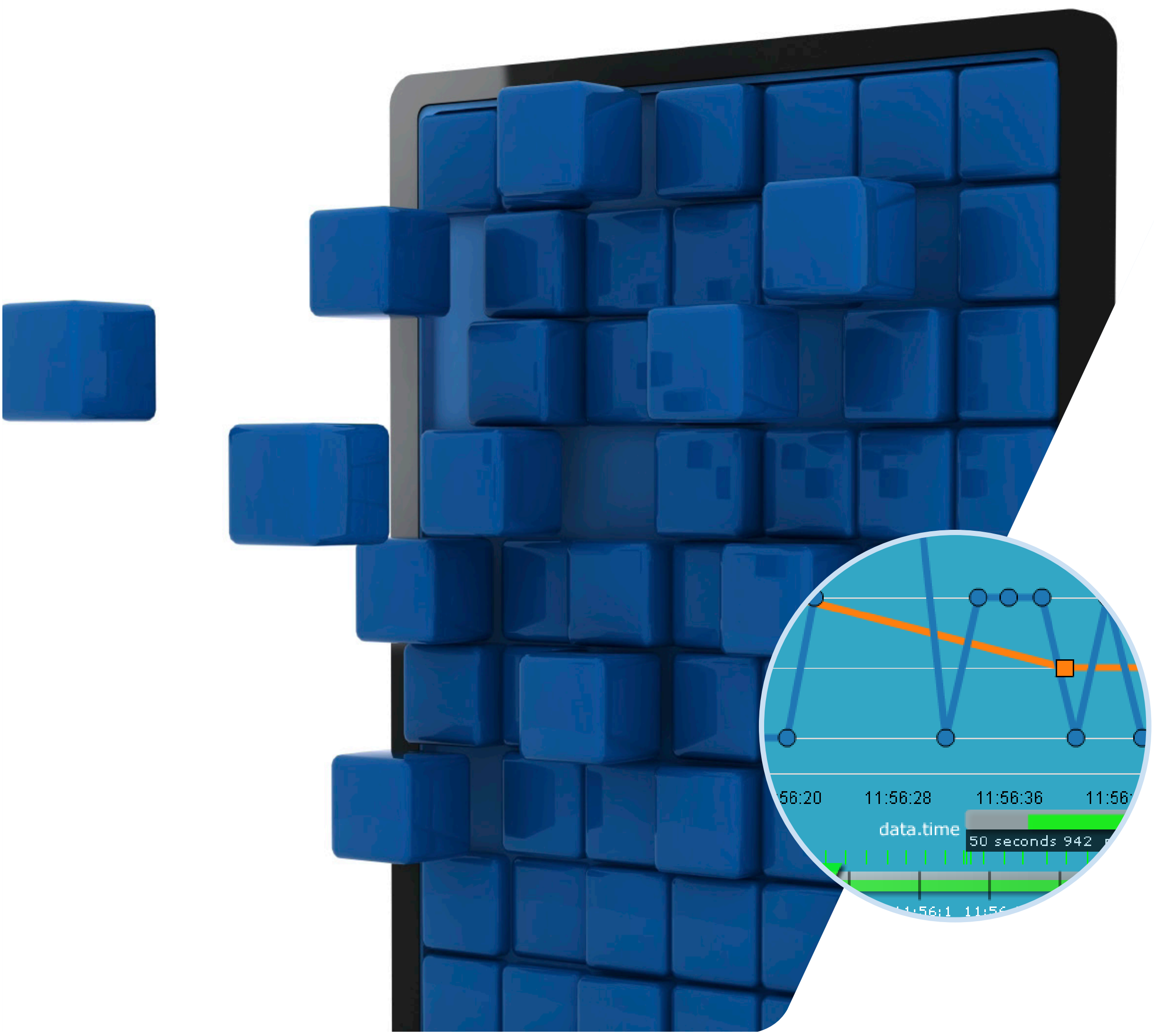
VTT PUBLICATIONS 746



Jarkko Kuusijärvi

Interactive visualization of quality variability at run-time

VTT PUBLICATIONS 746

Interactive visualization of quality variability at run-time

Jarkko Kuusijärvi

ISBN 978-951-38-7412-4 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1455-0849 (URL: http://www.vtt.fi/publications/index.jsp)

Copyright © VTT 2010

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 1000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 4374

VTT, Bergsmansvägen 5, PB 1000, 02044 VTT
tel. växel 020 722 111, fax 020 722 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O. Box 1000, FI-02044 VTT, Finland
phone internat. +358 20 722 111, fax + 358 20 722 4374

Technical editing Mirjami Pullinen

Jarkko Kuusijärvi. Interactive visualization of quality variability at run-time [Laadun varioituvuuden interaktiivinen visualisointi ajoaikana]. Espoo 2010. VTT Publications 746. 111 p.

Keywords software visualization, quality attributes, security, smart environments, adaptive software

Abstract

Smart environments are dynamic in nature, and the software running in these environments requires quality adaptations in order to function efficiently. The result of these adaptations, i.e., quality variability, must be verified in some way, and visualization can be used to aid this verification process. The research problem in this work was to find suitable visualization techniques to visualize quality variability and implement a visualization tool that encompasses these techniques and provides an interactive visualization of quality variability for the user.

As a solution to the research problem, this work presents an interactive quality visualization tool. The requirements specification for the implemented tool was derived from the literature review and the intended usage context of the tool, i.e., smart environments. The literature review explores a set of applicable visualization techniques and compares existing visualization tools with regard to the features required to represent quality variability visually at run-time.

The visualization techniques selected for the tool include interactive timelines, charts and meters that enable analysis of the quality attributes and their variability in different time ranges or points in time. Some additional visualization techniques were also included such as treemaps and graphs to visualize the structure of the smart environment.

The visualization techniques include open source visualization techniques and self-made techniques designed and implemented from the start to cover the specific requirements set for the tool. The main contribution of this work is the visualization tool that can be used to visualize different quality attributes and their variability. Moreover, the tool can easily be deployed in different environments due to its architecture and the selected implementation technologies that make the solution extensible and portable.

The implemented visualization tool was evaluated in the context of a smart environment in which security was adapted at run-time. The case study demonstrated that the implemented tool can be used in the analysis of the variability of

different quality attributes. The trend of a single quality attribute can be stud-ied for different time ranges or points in time according to need. The relation-ships between different quality attributes can also be studied with the help of appro-priate visualization techniques. In addition, the visualization tool was suc-cessfully tested on mobile devices.

Jarkko Kuusijärvi. Interactive visualization of quality variability at run-time [Laadun varioituvuuden interaktiivinen visualisointi ajoaikana]. Espoo 2010. VTT Publications 746. 111 s.

Avainsanat software visualization, quality attributes, security, smart environments, adaptive software

Tiivistelmä

Älykkäät ympäristöt ovat luonteeltaan dynaamisia ja vaativat niissä ajettavilta ohjelmistoilta kykyä muuntautua vastaamaan ympäristön tilannetta; adaptoitua, jotta ohjelmistot pystyisivät toimimaan suorituskykyisesti. Laadun variointi eli adaptointi pitää pystyä todentamaan, ja todentamisessa voidaan käyttää hyväksi visualisointia. Tässä työssä tutkimusongelmana on ollut löytää laadun varioituvuuden visualisointiin soveltuvat visualisointitekniikat ja toteuttaa visualisointityökalu, joka toteuttaa nämä visualisointitekniikat ja tarjoaa käyttäjälle vuorovaikutteisen visualisoinnin laadun varioituvuudesta.

Ratkaisuna tutkimusongelmaan tässä työssä esitellään laadun varioituvuuden interaktiivinen visualisointityökalu. Visualisointityökalun vaatimusmäärittely johdettiin taustatutkimuksesta ja työkalun oletetusta käyttökohteesta, älykkäistä ympäristöistä. Taustatutkimuksessa esitellään soveltuvia visualisointitekniikoita ja vertaillaan olemassa olevia visualisointityökaluja ottaen huomioon ominaisuudet, jotka tarvitaan laadun varioituvuuden ajoaikaiseen visualisointiin.

Visualisointityökaluun valittuihin visualisointitekniikoihin kuuluvat muun muassa vuorovaikutteiset viivadiagrammit, kaaviot ja erilaiset mittarit, joiden avulla laatuattribuuttien arvoja ja varioituvuutta voidaan analysoida eri aikaväleillä. Yleisten visualisointitekniikoiden lisäksi työkaluun sisällytettiin myös muita soveltuvia visualisointitekniikoita, kuten puukarttoja ja graafeja, joiden avulla älykkään ympäristön rakennetta visualisoidaan.

Työkalun visualisointitekniikat sisältävät valmiiksi toteutettuja avoimen lähdekoodin visualisointitekniikoita, joiden lisäksi suunniteltiin ja toteutettiin visualisointitekniikoita kattamaan työkalulle asetetut vaatimukset. Työn pääsaavutuksena kehitetään työkalu, jota voidaan käyttää eri laatuattribuuttien ja niiden varioituvuuden visualisoimisessa. Sen lisäksi työkalun arkkitehtuuri ja valitut toteutusteknologiat mahdollistavat työkalun käytön eri ympäristöissä sen laajennettavuusominaisuuden ja siirrettävyyden ansiosta.

Toteutettu visualisointityökalu testattiin käyttäen kontekstina älykästä ympäristöä ja siinä tapahtuvaa tietoturvan ajoaikaista adaptointia. Malliesimerkki osoitti, että työkalun avulla eri laatuattribuuttien varioituvuutta pystytään tutkimaan usealla tavalla. Yksittäisen laatuattribuutin kehityssuuntaa pystytään arvioimaan määrätyllä ajanjaksolla ja useiden laatuattribuuttien välisiä yhteyksiä pystytään myös tutkimaan usealla eri visualisointitekniikalla. Lisäksi visualisointityökalua testattiin mobiililaitteissa onnistuneesti.

Preface

This thesis was written at VTT Technical Research Centre of Finland at the Software Architectures and Platforms knowledge centre. The work was carried out as part of the EVOLVE (Evolutionary Validation, Verification and Certification)/ITEA2 project. The tool was tested in the context of Smart Spaces in the SOFIA (Smart Objects for Intelligent Applications) project.

I would like to thank Research Professor Eila Ovaska and Research Scientist Antti Evesti from VTT for their reviews and support during this work and Professor Juha Röning from the University of Oulu for his instructions and for reviewing this work. I would also like to thank Professor Jukka Riekki, the second reviewer of this work.

Oulu, March 19, 2010

Jarkko Kuusijärvi

Contents

Abstract 3

Tiivistelmä 5

Preface 7

Abbreviations 10

1. Introduction 12

2. Related research and technologies 14
   - 2.1 Software quality 14
     - 2.1.1 Software quality attributes 16
     - 2.1.2 Quality variability 17
     - 2.1.3 Software verification and validation 19
   - 2.2 Contexts for quality variability 20
     - 2.2.1 Service-oriented architecture 21
     - 2.2.2 Smart environment 21
     - 2.2.3 Quality attributes in smart environments 22
   - 2.3 Introduction to visualization 23
     - 2.3.1 Reference model for visualization 24
     - 2.3.2 Information visualization 25
     - 2.3.3 Challenges in information visualization 28
   - 2.4 Existing visualization tools 30
     - 2.4.1 EVolve 31
     - 2.4.2 Toolkit: prefuse 32
     - 2.4.3 Streamsight 33
     - 2.4.4 PortVis 35
     - 2.4.5 SnortView 36
     - 2.4.6 A network security visualisation prototype 37
     - 2.4.7 NEXThink REFLEX 38
     - 2.4.8 Summary 39

3. Interactive Quality Visualization tool 43
   - 3.1 Overview 43
     - 3.1.1 Monitoring process 45
     - 3.1.2 Visualization process 46
   - 3.2 Requirements 47
   - 3.3 Architecture 50
     - 3.3.1 Structure 50
     - 3.3.2 Visualization Platform 54
     - 3.3.3 VMonitor 60
     - 3.3.4 Visualization View 61
     - 3.3.5 ActionScheduler 63

3.4 Visual mapping 65
3.4.1 Graphs 65
3.4.2 Treemaps 66
3.4.3 Timelines and scatter plots 68
3.4.4 Meters 69
3.4.5 Time slider 70

4. Implementation and testing 72
4.1 Implementation environment 72
4.1.1 Constraints 73
4.1.2 Graphics libraries 73
4.2 Implemented components 74
4.2.1 Initial version of the tool 74
4.2.2 VMonitor 76
4.2.3 Visualization View 77
4.2.4 Meters 79
4.2.5 Time slider 79
4.2.6 Timeline visualization 81
4.2.7 Treemap visualization 83
4.3 Testing 84
4.4 Case study 86
4.4.1 SOFIA Smart Space 86
4.4.2 Testing environment 87
4.4.3 Validation scenario overview 87
4.4.4 Smart environment structure 89
4.4.5 Quality attribute analysis 90
4.4.6 Quality trade-off analysis 92
4.4.7 Cross-platform testing 93
4.4.8 Summary of case study results 97

5. Discussion 98
5.1 Implementation of the IQVis tool 98
5.2 Results 99
5.3 Comparison with existing tools 101
5.4 Future development 102

6. Conclusion 103

References 105

Abbreviations

API	Application Programming Interface
AVM	ActionScript Virtual Machine
CIA	Confidentiality, Integrity and Availability
CPU	Central Processing Unit
EMonitor	Environment Monitor
EVOLVE	Evolutionary Validation, Verification and Certification, project
FP	Flash Player
FPS	Frames Per Second
GUI	Graphical User Interface
HCI	Human-Computer Interaction
IEEE	Institute of Electrical and Electronics Engineers, an international non-profit professional organization for the advancement of technology related to electricity
IOP	InterOperability Platform
IQVis	Interactive Quality Visualization
ISO/IEC	International Standardization Organization and International Electrotechnical Commission
IV	Information Visualization
JSON	JavaScript Object Notation, lightweight data-interchange format
MVC	Model-View-Controller
NIDS	Network-based Intrusion Detection System
OSGi	Open Service Gateway initiative (now the OSGi Alliance)
PC	Personal Computer
RDF	Resource Description Framework
SaaS	Software as a Service
SDK	Software Development Kit
SIB	Semantic Information Broker

SOA	Service Oriented Architecture
SOFIA	Smart Objects for Intelligent Applications
SS	Smart Space
SSA	Smart Space Application
SSAP	Smart Space Access Protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
TTT	Type by Task Taxonomy
UI	User Interface
V&V	Verification and Validation
VMonitor	Visualization Monitor
XML	eXtensible Mark-up Language

1. Introduction

Modern software products typically run in highly dynamic environments and interact with multiple other software products running in the same environment. At the same time, environments are becoming more and more dynamic with devices continuously entering and leaving the environment. Smart environments consist of several different types of devices interconnected for information exchange [1]. Pervasive or ubiquitous computing has emerged through recent advances in middleware technologies, software agents and smart technologies (e.g., sensors, devices, wireless networks) [1]. Pervasive or ubiquitous computing offers a suitable platform for realizing smart environments that acquire and apply knowledge effectively and link computers to everyday settings and commonplace tasks in our surroundings [1]. When entering and interacting with a smart environment, the user may want to preserve a certain security level (or security performance). The user can, for example, just look up devices in the smart environment where no security is needed, but when the user decides to use a service such as a display or an application, he/she may wish to keep his/her actions secret.

Changes to security solutions at run-time can be considered to constitute quality variability, which is controlled reconfiguration of quality rather than quality variation, which is unwanted change to a quality. Quality attributes or quality requirements, e.g., reliability and security, can be specified using ontologies at design time and be connected to architectural models to provide static solutions [2]. In order to handle quality variability at run-time, however, dynamic solutions are needed based on the context of the reconfiguration [3]. As stated in [4], it is not enough to make all security decisions at design time and it is therefore necessary to manage security at run-time. For these purposes it is necessary to reveal the modifications that need to be made to the security by means of monitoring and adaptations.

In this work, the research problem concerns the way software quality variability can be visualized at run-time to help the user analyse the adaptation results. The visualization concerns only execution qualities, as the purpose is to visualize run-time behaviour. This research problem has several aspects: 1) how to visualize the change in quality attributes so that the user can analyse the variability effectively, 2) which visualization techniques are used to map the quality attribute data into a visual form, and 3) how to make the tool extensible and easily adaptable so that it can be used in different contexts. Run-time visualization of these qualities is necessary for the system administrators to analyse, verify and validate the adaptation results in real time. Visualization is even more helpful in smart environments in which users join, leave and use services in a dynamic way and adaptations are more likely to occur, and administrators can therefore use visualization to help determine realized qualities in different devices and environments.

The purpose of this work is to design and implement a tool that solves these problems. How are quality attributes, e.g., security, visualized so that the changes are noticeable? Although the main contribution of this thesis is visualization of quality attributes, the acquisition of these attributes is also considered in the data source of the tool so that variability can be presented in real time. The tool is demonstrated in the context of visualizing security variability in a smart environment in the Smart Objects for Intelligent Applications (SOFIA) project [5]. In a smart environment, numerous devices and applications can work in the same or a different environment.

The implemented tool is called Interactive Quality Visualization and has the acronym IQVis. The primary requirement of the IQVis tool is that it can visualize quality attributes such as security at run-time to allow administrators of the system to analyse the realized variability and adaptation. This tool is used to visualize quality attributes in a specific environment, i.e., it is not an abstract visualization solution. This tool is a task-specific visualization tool designed to visualize quality variability. The tool is therefore not used to visualize large amounts of data but to visualize certain properties, i.e., the behaviour, of the monitored system to help the user observe the system. The tool should also be user friendly and support extensibility and adaptability. The tool is used at run-time in the context of a smart environment [5], and the monitor component working as the data source should thereby support real-time updates to the qualities. The run-time security monitoring and adaptation system used to provide the quality attributes from the Smart Space (SS) is outside the scope of this work

2. Related research and technologies

This chapter describes research and technologies related to visualization and the subject that is visualized, software quality and its variability. The research on visualization focuses on the field of information visualization. First, software quality, verification and validation, and smart environments are discussed as the context to the visualization needs. Second, visualization as a term is explained, and visualization systems and tools are reviewed and compared to provide an insight into existing visualization techniques. A comparison of the reviewed tools is provided at the end of this chapter.

2.1 Software quality

Software quality is defined as "the degree to which software possesses a desired combination of quality attributes" in the Institute of Electrical and Electronics Engineers (IEEE) 1061 standard for a Software Quality Metrics Methodology [6]. The standard also defines a quality attribute as a characteristic of software or a generic term applying to quality factors, quality subfactors or metric values [6].

The International Standardization Organization and International Electrotechnical Commission (ISO/IEC) standard 9126-1 [7] define a software quality model for external and internal quality that categorizes software quality attributes into six characteristics: functionality, reliability, usability, efficiency, maintainability and portability. These characteristics are defined as follows [7]:

- *Functionality* is the capability of the software product to provide functions that meet stated and implied needs when the software is used under specified conditions.
- *Reliability* is the capability of the software product to maintain a specified level of performance when used under specified conditions.

- *Usability* is the capability of the software product to be understood, learned, used and attractive to the user when used under specified conditions.
- *Efficiency* is the capability of the software product to provide appropriate performance relative to the amount of resources used, under stated conditions.
- *Maintainability* is the capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in the environment and requirements and functional specifications.
- *Portability* is the capability of the software product to be transferred from one environment to another.

These characteristics are further divided into subcharacteristics that influence the quality characteristics, for instance, functionality comprises suitability, accuracy, interoperability and security. The capability of the software is determined by a set of internal attributes that can be measured with the use of metrics [7]. A metric is a function that outputs a single numerical value (which can be interpreted as the degree to which the software possesses a given attribute that affects its quality) from software data input [7]. Examples of metrics for external characteristics are given in ISO/IEC 9126-2 [8] and for internal characteristics in ISO/IEC 9126-3 [9].

Figure 1 presents the hierarchy of quality terms, from quantitative quality metrics to high-level software quality [6], [7]. This thesis focuses on the visual representation of the measured quality attributes.

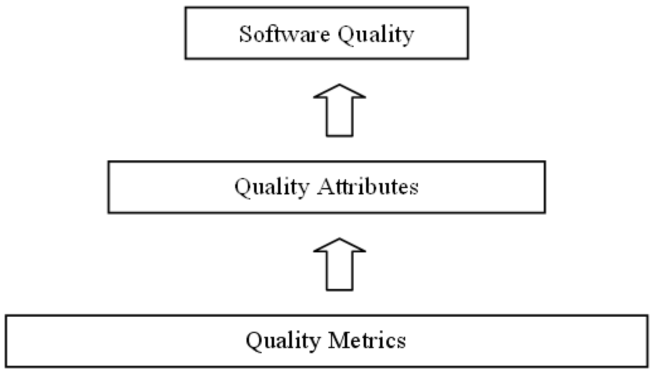


Figure 1. Quality hierarchy.

2.1.1 Software quality attributes

Quality attributes are non-functional features of a system. Quality attributes are categorized into execution and evolution quality attributes (or simply ‘qualities’) [10]. Execution qualities (functional qualities) are observable at run-time and thus express the behaviour of the program. The selected execution qualities for visualization are listed in Table 1 (modified from [10]).

Table 1. Execution qualities.

Attribute	**Description**
Performance	Responsiveness of the system, which means the time required to respond to stimuli (events) or the number of events processed in some interval of time.
Security	The system’s ability to resist unauthorized attempts at usage and denial of service while still providing its service to legitimate users.
Availability	Availability measures the proportion of time the system is up and running.
Scalability	The ease with which a system or component can be modified to fit the problem area.
Reliability	The ability of the system or component to keep operating over the time or to perform its required functions under stated conditions for a specified period of time.
Adaptability	The ability of software to adapt its functionality according to the current environment or user.

Of these qualities, security is of particular interest as a required attribute for visualization at run-time. The reason for this is that in the example, smart environment [5] security is adapted at run-time according to the situation in the smart environment at any given time. Nevertheless, the visualization of the execution qualities in general is taken into account.

Evolution qualities (non-functional qualities) are not observable at run-time, however, because the solutions for these qualities lie in static structures of the software system. For example, evolution qualities should be considered in the development and maintenance of a software system. The evolution qualities include maintainability, flexibility, modifiability, extensibility, portability, reusability, integrability and testability. [10]

It is said [11] that implementations of non-functional requirements typically look similar across different systems, while functional requirements and their implementations are closely tied to individual systems. A tool for the automated addition of architectural quality attributes for Java software is presented in [11]. The tool allows capturing the non-functional requirements and placing them in a context with the functional specification and finally generating code that implements the required functional and non-functional requirements. Object visualization and logging service, for example, do not interfere with the normal interaction between the object and its clients. The tool also supports services that may modify the interaction, fault masking that helps to achieve availability of the software and atomic transaction that allows deferred executions. [11]

An example of visualizing quality attributes in the design phase is presented in [12], in which trust and performance are visualized with a goal model showing nodes and edges. The tool presented in [12] does not visualize the quality attributes of software at run-time. This thesis concentrates on the execution qualities at run-time, i.e., visualizing the variability of these qualities in an appropriate and effective way. The aim is for the variability of these qualities to be easily noticeable with the help of visualization. The techniques to monitor the variability of these qualities are outside the scope of this thesis. Visualization is easier for some qualities than for others. Performance, for instance, can be monitored through measurements, and easily visualized with numbers corresponding to, for example, response times. Other qualities such as adaptability require some monitoring and the development of a visualization technique suitable for demonstrating its value and variability.

2.1.2 Quality variability

Variability is defined as “the ability of a software artefact to vary its behaviour at some point in its lifecycle” [13]. The software artefact can be, for example, the software architecture design, components, classes, source code and executable binaries. There are a number of steps to introduce variability into software. The first step is the identification of variability, i.e., where variability is needed. The second step is to constrain variability, i.e., limit the features of the identified variation to provide sufficient flexibility. The third step is to implement variability, i.e., select a realization technique that provides the best possible balance between the constraints identified in the previous step. The last step is to manage the variability, i.e., maintain and manage the variability introduced into the software when, for example, requirements change over the time, new products are added or old products are removed. [13]

A quality attribute variability model that uses ontologies to model variability is introduced in [3]. The paper in [3] focuses on quality variability management of execution qualities. Three types of quality variability in software family architectures are defined [3]:

- *Variability among different quality attributes,* for example: reliability is an important quality for one family member, but for other family members there are no reliability requirements.
- *Different priority levels of quality attributes,* for example: in high-end products reliability is an extremely important property, while in other products only medium or low-level reliability is needed.
- *Indirect variation*: functional or quality variability can cause variation, for example, improvements in the reliability of one component may require other components to also be at the same level of reliability.

Variation points are defined as points at which choices are made with regard to which variation should be used in a particular place in a software system. One technique to implement variation points is the variability realization technique. The taxonomy of these techniques is presented in [13]. Quality variability with variation points is a static solution implemented at design time. Static solutions are not enough in service-oriented systems in which quality changes over time according to the context. The context can be, for example, the environment and users (external state of the system) or capabilities, resources and regulations (internal state of the system) [3]. The reasons (i.e., sources) for quality variability can be subjective, business related or technological, as defined in [3]:

- *Subjective reasons*: the user of a software service prefers different qualities in different contexts.
- *Business reasons*: the type of application may set different quality criteria, for instance, differing measurement accuracy related to time, place and ratio for services intended for professional use as opposed to those for non-professional use.
- *Technological reasons*: the implementation technology or amount of available resources may lead to quality variation, especially when externally developed software or a service is used without adaptation.

The handling of quality variability at run-time requires that the following assumptions can be realized [3]:

- quality attributes are defined in an unambiguous way
- quality attributes have quantitative metrics
- quality characteristics are defined explicitly in the architecture models
- quality characteristics can be measured at run-time
- the dynamic system has decision-making mechanisms concerning reconfiguration validity and correctness.

The assumptions presented above are essentially elaborated requirements, with the emphasis on run-time requirements, for the steps presented in [13] for introducing variability into software. There are many tools and techniques to model quality attribute variability at design-time [14]. An example of a tool that supports quality requirements specifications at design-time is the Quality-Oriented Architecting Environment that enables the definition of the quality requirements in architecture models using a specific ontology [2]. Quality can also be modelled in Model-Driven Development using Quality-Driven Domain-Specific Modelling to map requirements to models [15]. While there are tools for design-time quality variability management, there is a need for techniques and tools to analyse variability at run-time [3].

2.1.3 Software verification and validation

Software verification and validation (V&V) is a subject that is closely related to software quality. The IEEE 1012-2004 standard [16] defines verification and validation as follows:

- *Verification* is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. Verification is the process of providing evidence that the software and its associated products satisfy system requirements allocated to software at the end of each life cycle activity and solve the right problem.
- *Validation* is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies the specified requirements. Validation is the process of providing objective evidence that the software and its associated products conform to requirements for all life-cycle activities during each life-cycle process.

Informally, verification may be defined by the question “Are we building the product right?” and validation by the question “Are we building the right product?” as stated in [17]. Software integrity levels such as complexity, criticality, risk, safety level, security level, desired performance, reliability and other characteristics are used to determine the V&V task to be performed [16]. V&V techniques are usually used at design time and during the life-cycle of a software system. Tests include, for example, acceptance, component, integration and system testing [16]. Due to the complex software systems of today, V&V is also performed at run-time to observe situations that traditional V&V techniques cannot cover.

Run-time verification[1] is used to check dynamically that the actual executions of the system conform to the given requirements of the system. Run-time verification techniques are intended as the lightweight counterpart of traditional (off-line) verification techniques to check the properties that cannot be fully verified offline. Run-time verification techniques analyse the observed behaviour in the target system to check the correctness of single executions by, for example, monitoring events and outputs. The amount of resources available at run-time is limited when compared with off-line methods, and properties that can be checked at run-time therefore tend to be quite simple. [18, p. 526]

Run-time verification can therefore be used, for instance, to verify the security of the software system at run-time or check information that is only available at run-time. The comprehensiveness of checks is often a trade-off against efficiency, and the level of abstraction in verification is therefore an important choice. In many cases it is necessary to monitor multiple components and multiple abstraction levels of different operations. A combination of multiple monitors and levels of abstraction is presented to improve run-time verification, especially in the case when the system is intended to maintain security properties. [19]

2.2 Contexts for quality variability

As stated in [3], quality shall be changed according to context over time in service-oriented systems. Service-oriented systems have become an essential aim of today’s software. The following describes service-oriented architecture, software as a service, and smart environment concepts. The quality properties, i.e., reli-

[1] http://www.runtime-verification.org.

ability, availability, performance and security, of smart environments are also discussed.

2.2.1 Service-oriented architecture

Service-oriented architecture (SOA) is essentially a collection of services that communicate with each other. There are many formal definitions of SOA. The Open Group, for example, defines SOA as “an architectural style that supports service orientation” [20], while the OASIS group defines SOA as “a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains” [21]. The idea of SOA is to provide flexible and loosely coupled services and thereby improve interoperability of services. SOA basically consists of a service consumer, a service provider and a service directory. SOA provides a flexible infrastructure that can easily adapt to user requirements. [22]

Software as a Service (SaaS) is a delivery paradigm of software in which the software is offered to the user as a service through the Internet. The software is hosted on, for example, a Web server and delivered to the customer on demand. Payment for the software follows a subscription model in which the user can order the service for a period of time. [23] SaaS provides users with the opportunity to use software on-demand for a period of time and can therefore be used as a delivery paradigm in smart environments. SaaS can be used with SOA to provide different services more flexibly by offering the architecture for it. Smart environments consist of different services that are offered to the user. SaaS and SOA provide the means for realizing these services.

2.2.2 Smart environment

Cook and Das [1] refer to a smart environment as “one that is able to acquire and apply knowledge about the environment and its inhabitants in order to improve their experience in that environment”. This definition implies that smart environments (i.e., the software) must be able to adapt functionality and quality to provide users with the best experience. Some of the most recent topics in computer science, i.e., ambient intelligence, and ubiquitous and pervasive computing, aim to create smart environments by making applications and their functionalities independent of PC hardware and available in any place [24]. Hermann et al. studied research laboratory demonstrations of smart environments and listed the potential and the key aspects of the approaches as follows [25]:

- highly integrated and seamlessly available data, services and resources in public and private environments
- exchange of information, access rights of objects, ambient resources and devices
- exchange of personal information between users and environment
- location-based availability of nearby entities, location-based User Interfaces (UIs) for services, data and applications
- system "intelligence": adaptivity and to some degree autonomous system decisions, e.g., on the use of ambient systems or data exchange.

Of the abovementioned list, the system "intelligence" aspect is a key factor in this thesis, as quality variability can be thought of as part of this. Smart environments are environments that vary dynamically, as devices can join and leave continuously. Performance can be affected when users join and leave, and consume different services dynamically (data latency and transaction throughput [14]). Reliability is another quality that is essential in a smart environment. Performance and reliability, for example, must be provided to emergency services. Typical devices found from smart environments include sensors, actuators and smart devices [1].

2.2.3 Quality attributes in smart environments

Smart environments require the qualities explained in Table 1, which include reliability, availability, performance and security, to mention a few. As smart environments are dynamic in nature, variability of these qualities is also necessary: different software needs different levels of, for example, security. An application that is used to handle the user's personal information, e.g., bank information, must guarantee a sufficient level of security. Performance is directly affected by the number of devices using a service at a given time. Availability and reliability are key qualities for providing a seamlessly working smart environment for the users.

ISO/IEC's 9126 standard defines security as the "capability of the software product to protect information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or systems are not denied access to them" [7]. The core principals of information security can be divided into three categories: confidentiality, integrity and availability (the CIA triad). Confidentiality is about protecting information from unauthorized access, integrity concerns protecting information from unauthorized modification, and availability concerns guaranteeing legitimate access to the information at all times. [26, pp. 4–5]

Confidentiality and integrity are usually provided by encryption and decryption. The characteristics of smart environments, i.e., a decentralized and dynamic nature, complicate encryption, decryption and authentication. [27]

2.3 Introduction to visualization

Historically, visualization was divided into scientific visualization and information visualization (IV). Scientific visualization typically involves scientific application and physical data (e.g., medical images, meteorology or even mathematics) whereas information visualization involves abstract data (e.g., financial data) [28]. Visualization is often used to analyse very large amounts of data (e.g., [29]). These two major areas of visualization often contradict each other, as it is often not clear to which category some data set belongs [30]. Sub-areas of research are drawn from information visualization, for instance, security visualization and software visualization.

Security visualization uses visualization techniques to help, for example, analyse the huge security-related logs or network traffic faster (e.g., [31]). Software visualization, however, visualizes the artefacts related to software and its development. It visualizes the structure, behaviour and evolution of software [32, p. 3]. The focus of this thesis is on information visualization, specifically security and software visualization, as the idea is to visualize software quality attribute variability at run-time. Security visualization is also reviewed, because security variability is the context for visualization in the example environment.

Ware [33, p. 2] defines the term visualization as “a graphical representation of data or concepts”. This definition states that visualization can basically be anything, for example, an image, map, presentation slide or anything that changes data into a visual form. Vision is the most valuable sense for providing information from computers to humans because humans acquire more information through vision than though all the other senses combined [33]. For this reason, visualizations have an expanding role in cognitive systems. Card et al. [28, p. 6] define visualization as “the use of computer-supported, interactive, visual representation of data to amplify cognition,” where cognition is the acquisition of knowledge. This narrows down the definition into computer-supported visualization in which people can also interact with the result and use it to assist in the understanding of the data. The main purpose of visualization is therefore to give an insight into the data, i.e., help in decision-making and discovery. Card et al. propose six major ways in which visualization can amplify cognition by perception [28, p. 16]:

1. Increasing memory and processing resources by allowing storage of massive amounts of information in a quickly accessible form (e.g., maps).
2. Reducing searching by grouping information together.
3. Enhancing recognition of patterns by enhancing patterns.
4. Perceptual inference by making some problems obvious.
5. Perceptual monitoring by allowing monitoring of a large number of potential events.
6. Manipulable medium by allowing exploration of a space of parameters unlike static diagrams.

The benefits of visualization are stated above. Visualization can help us interpret abstract and/or complex data and form a mental image of them. It is easier to find, for example, the lowest and highest values of a price from a line graph than from a table of prices if there are hundreds or thousands of different values.

2.3.1 Reference model for visualization

The high-level architecture of a visualization system can be described as a pipeline in which raw information is converted into a visual form by means of mapping. Figure 2 [28] depicts the main faces in the visualization pipeline. The first step is to transform raw information into data tables, i.e., relational descriptions of data extended to include metadata. Data tables are then converted into visual structures (graphical properties) by means of visual mapping. Finally, views are constructed with view transformations (specifying, for example, position, scale or other graphical parameters). The user can interact with any of the steps and change the parameters. [28]

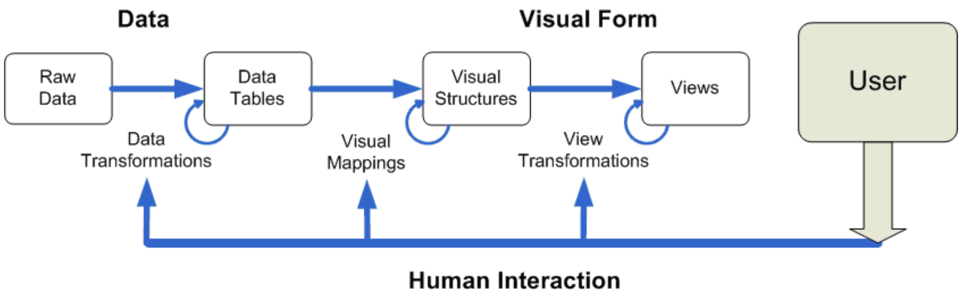


Figure 2. Reference model for visualization.

2.3.2 Information visualization

Information visualization (IV) is a subcategory of visualization that visualizes abstract information (with no natural mapping between the data and visual element), such as financial data. Information visualization usually includes human interaction to help with the visualization usage.

The Visual Information-Seeking Mantra "Overview first, zoom and filter, then details on demand" described by Shneiderman [34] gives instructions on how data should be presented so that they are most effective for users. This guideline is commonly used in information visualization. The aim of interacting with visualization is to find the information needed from the data by, for example, filtering unwanted results.

Types of data

The first step of visualization is the raw data to be converted into a visual representation. The raw data in information visualization can basically be anything. One classification of data is made by Ware [33]. It divides the data into entities, relationships and the attributes of these. Entities are the objects of interest, for instance, people, fish or devices. Relationships are conceptual associations between entities, such as a device belongs to a smart environment. Attributes, on the other hand, are properties attached to entities or relationships, e.g., the device is a tablet. Ware defines the quality of attributes into category data, integer data or real-number data [33]. Category data means that the data have a nominal value, such as a label. Integer data are used for data that can be numbered or ordered, for instance, the position in a list, and real-number data are used in other cases. [33]

Taxonomies

Shneiderman [34] proposed taxonomy based on data types and low-level tasks. The type by task taxonomy (TTT) includes seven data types. The TTT is explained below [34].

- *1-dimensional*: simple, linear data, e.g., sets or sequences such as program source code or text documents.
- *2-dimensional*: planar or map data, such as floor plans of a building or other layouts, geographical maps.
- *3-dimensional*: physical objects, such as the human body, a molecule or a car.
- *Temporal*: timelines, such as project management or historical presentations in which the start and the stop time of items are separated.
- *Multi-dimensional*: data with more than three attributes, such as relational and statistical databases.
- *Tree*: tree structures of hierarchies of data or node-link diagrams in which each item has only one parent item (except the root), such as file system directories.
- *Network*: node-link structures (graphs) in which nodes can be connected to an arbitrary number of other nodes, such as the World Wide Web network.

This taxonomy is only a high-level presentation and these data types are often mixed in order to provide a more precise visualization of the subject of study, for example, when visualizing stock prices, time is just another dimension with one-dimensional data. The seven tasks related to these data types are presented below [34]:

- *Overview*: provides a view of the whole data collection.
- *Zoom*: provides a detailed view of a single item.
- *Filter*: filters out uninteresting items of data from the view.
- *Details-on-demand*: provides details of a selected item group or item.
- *Relate*: views relationships between selected items or item groups.
- *History*: stores a history of user actions (undo, replay, step-by-step progress).

- *Extract*: allows extraction of information from selected items (or saving the current settings/view of the program).

The tasks presented above provide high-level interaction techniques for the visualization that comply with the Visual Information-Seeking Mantra. Other similar interaction techniques are presented in [35] as follows:

- *Select*: These techniques provide the user with the ability to select item(s) of interest. This helps the user follow the item of interest when the representation technique is changed or a large number of items are visible at the same time. An example of this is highlighting the selected item in a different colour or making the item noticeable in other ways.
- *Explore*: These techniques allow the user to examine the visualization in a different way, for example, when the user views a subset of data, he/she can change the subset to a different one or remove/add items from/to it. The most common explore technique according to [35] is panning, when the user can move the scene or the camera, for example, by dragging the camera with the mouse.
- *Reconfigure*: These techniques allow the user to view the data from different perspectives, i.e., to help the user recognise hidden characteristics of data and the relationships between them. An example technique is support for sorting and rearranging table columns.
- *Encode*: These techniques enable the user to modify the current visual representation, for instance, by changing the colour, shape or size of an item or changing the whole view to a different one. The user can also change, for example, the representation of data from a pie chart to a histogram.
- *Abstract/Elaborate*: These techniques provide the user with the possibility of adjusting the level of abstraction of the view. The user can, for example, first look at the overview and then elaborate individual items. This can be done by, for example, showing a tool tip for an item or by zooming in on the view so more details can be seen. These techniques can actually be any of the techniques from the details-on-demand task described in the type by task taxonomy of Shneiderman [34].
- *Filter*: These techniques allow the user to delimit the data items being visualized. The user can, for example, specify a range or condition so that only items meeting the criteria are shown. An example of this is

the user selecting ranges by moving sliders or clicking check boxes. The view is updated accordingly. These techniques are called dynamic query controls [35].

- *Connect*: These techniques allow the user to highlight relationships and associations between the data items viewed and to show possible hidden items relevant to the specified item. This can be done by, for instance, highlighting or pointing out relevant items when the user clicks on an item or moves the mouse cursor over an item.
- *Other interaction techniques*: These techniques include common operations found in all interactive applications (not just information visualization), such as undo/redo (allow the user to undo or redo a command or view the history of commands or data sets and allow resetting of the commands or views).

While the TTT provides a high-level taxonomy for information visualization, it does not provide a detailed classification for visualization techniques. Chi analysed 36 different visualization techniques in [36] by breaking them down into four data stages (from raw data to the final view), three types of data transformations (required when transforming data from one stage into another) and four types within stage operators (e.g., for rearranging data). This pipeline resembles the high-level visualization pipeline presented by Card et al. in [28].

2.3.3 Challenges in information visualization

It is argued that humans can only observe detailed changes in one object at a time, and yet we still look at our surroundings with the impression that we see all the objects and their details simultaneously [37]. This lack of perception capability has to be taken into account when designing visualization systems so that excessive amounts of information are not shown at once. This is a difficult task when visualizing large sets of data or very detailed information. For this reason, interaction techniques such as redo and undo are necessary so that the user can look at the data from different perspectives and possibly several times. If the visualization technique uses animation to show the events that are occurring, there may be a wish to change the delay of the animation for different situations.

Different techniques that help visualize change in time series analysis are discussed in [38]. Meaningful characteristics of change through time in time series are magnitude of change, shape of change, velocity of change and direction of change [38].

- *Magnitude of change* identifies the difference between measures of something at two points in time.
- *Shape of change* identifies whether some value moves up, down, left or right, or a combination of these as it varies through time.
- *Velocity of change* refers to the speed or rate of change. If, for example, a line chart contains two measurements which both increase by 10 per cent at a time but one is 100 times bigger than the other then the velocity of change in the smaller measurement is not noticeable on the chart.
- *Direction of change,* or trend, identifies the overall or general direction of change in time series values, for example, the variable of time can be introduced into a scatter plot by adding animation. Another technique is to add visual trails that capture the place and state of an object through time, for example, every time the object's properties change, a new object is placed in the new position with new properties, and the old object is left in the old position and its visibility is reduced by some percentage so that it is identified as a previous object. [38]

User interface (UI) design and interaction are the major factors in designing visualization systems that help the user to explore the data. Chen [39] states that usability is recognized as the first problem of visualizations. Other problems stated in [39] include, for example, prior knowledge (level of prior knowledge needed to understand the visualized information), scalability, (e.g., the challenge of visualizing data streams) and aesthetics (the purpose is to provide an insight into the data, not just pretty pictures).

Yi et al. [35] divide information visualization systems in general into two main components: representation and interaction.

- The representation component concerns the mapping of data into visual representations and comes from the field of computer graphics.
- The interaction component comes from the field of human-computer interaction (HCI) and involves the dialogue between the user and the system.

Yi et al. argue that the representation component has received most of the attention by far in the IV research while the interaction component has often been relegated to a secondary role. They state that only a few papers had focused on the interactive aspects of IV (at that time, in 2007). Interaction is stated as an essential part of IV systems because without it the visualization techniques just become static images or autonomously animated images. They state that the further study of interaction is undisputed. [35]

Interaction techniques include sorting, zooming, mouse-overs, selections, animations, etc. Interactive navigation solutions can be roughly divided into three techniques [40]: focus+context, zooming+filtering and incremental exploration. Focus+context is a viewing approach that shows the user a global view of the data with a detailed view of a portion of the data. An example of a focus+context visualization made for graphs is presented in [41], which shows a detailed view of one node in the centre of the circle node graph and other nodes in less detail on the radius of the circle. Zooming+filtering reduces the amount of data visible by means of filtering, i.e., by selecting a subset of the data (e.g., a treemap in [29]). Incremental exploration techniques show only a portion of the data at a time and are thus able to handle huge amounts of data. [40]

Nowadays, distributed visualizations on the Internet provide lots of visualization techniques and interactivity from which the user can construct his/her own visualizations. Google Visualization[2], for example, provides many visualization techniques, such as maps, timelines and charts for free. Users can also create visualizations (or gadgets) from existing visualization techniques and add them to their web pages, or they can create new visualization gadgets to share with the world. Another similar example is Many Eyes[3], a data visualization tool that allows users to create new visualizations from data sets they upload themselves by using the visualization techniques provided.

2.4 Existing visualization tools

Much research has been conducted in the area of information visualization, and several tools and frameworks have been developed to visualize different topics. This section reviews some of the existing systems and tools developed in this research area. As the context for quality variability visualization in this thesis is security in Smart Spaces, the survey of existing tools included many security visualization tools in addition to abstract and other specific tools. The run-time

[2] http://code.google.com/apis/visualization/
[3] http://manyeyes.alphaworks.ibm.com/manyeyes/

visualization of security variability, for example, has not been a subject of research, although security visualization as a whole has been studied comprehensively. Most of the security-related systems concentrate on visualizing network security attacks such as the ones reviewed in this chapter. As the study concentrates on the run-time behaviour of the program, some tools such as the visualizing non-functional requirements [12] are left out.

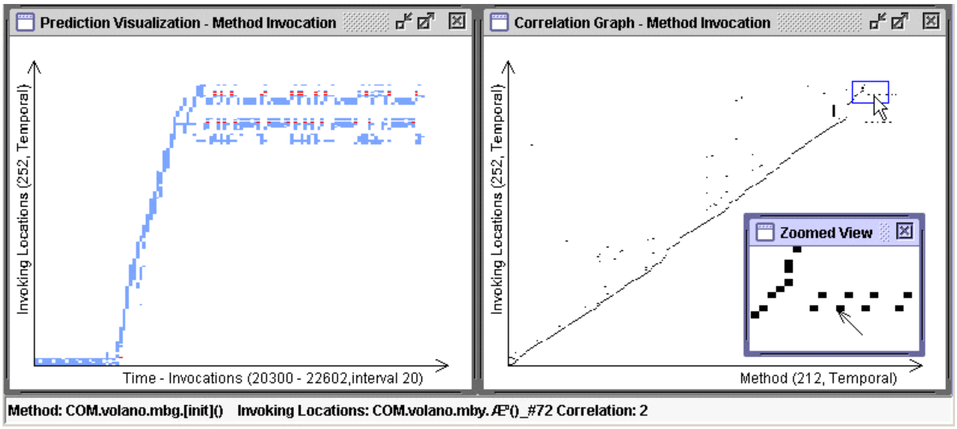
2.4.1 EVolve

EVolve (not to be confused with the EVOLVE project in which this thesis was written) is a software visualization framework that is designed to be both open and extensible. It was originally developed to visualize the run-time behaviour of Java programs to help develop compiler optimizations. New data sources or visualization techniques can easily be integrated into it as all the interfaces are clearly defined via the Java Application programming interfaces (APIs), and the framework is also publicly available. It can be used both as a standalone tool using predefined data sources and visualizations and as a toolkit for developing new visualizations. [42]

The EVolve platform consists of three components: data source, visualization library and fixed core. The core takes care of communication and is static, while the data source and the visualization library can be modified to fit the user requirements. The data in the example are collected offline. Extensibility is guaranteed with an abstract, internal representation of the source data. Data elements are divided into entities and events. Elements consist of an entity reference and a value, which also has properties such as amount and time. Custom properties can also be defined to support new visualizations. Different visualizations are able to display data from a variety of data sources as they provide an abstract presentation of their visualization capabilities. The core makes sure that only data elements with appropriate properties are sent to the visualization. [42]

The framework provides eight predefined visualizations that use either a tabular format to present data or x- and y-axes which show, for example, events as they occur in time. The user interface supports sorting, zooming, mouse-overs and selections to amplify the ability to interpret and draw conclusions. The user can also compare the data by viewing different visualizations at the same time or using colouring to track some data element.

Figure 3 [42] shows method invocations in two views. The left-hand picture shows the predictability of events and the right-hand picture shows the correlation between the invocation location and the invoked methods. [42]

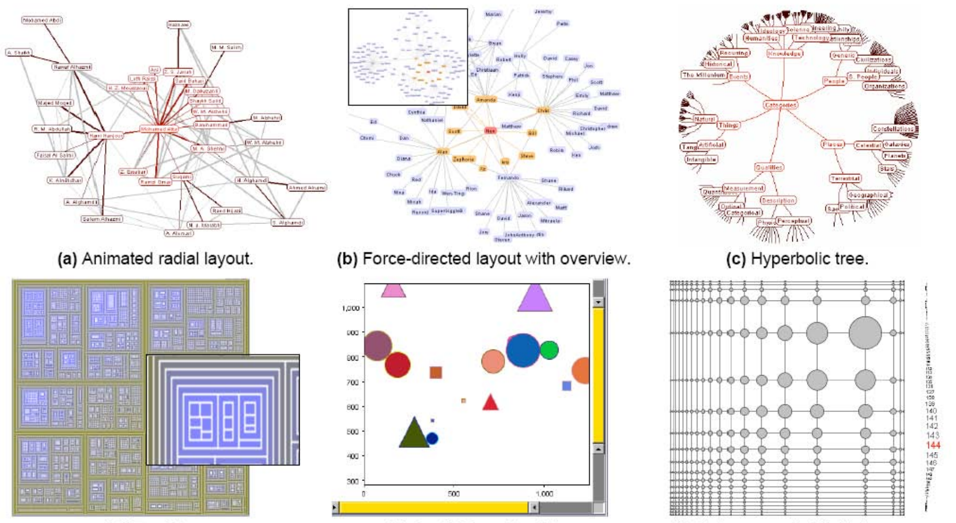

Prediction Visualization - Method Invocation
Invoking Locations (252, Temporal)
Time - Invocations (20300 - 22602,interval 20)
Correlation Graph - Method Invocation
Invoking Locations (252, Temporal)
Method (212, Temporal)
Zoomed View

Method: COM.volano.mbg.[init]()   Invoking Locations: COM.volano.mby.Æ()_#72 Correlation: 2

Figure 3. Method invocation views in EVolve.

2.4.2 Toolkit: prefuse

Prefuse [43] is a toolkit for interactive information visualization. It combines visualization building blocks such as node-link diagrams, containment diagrams and visualizations of unstructured (edge-free) data such as scatter plots and timelines. Some sample visualizations from the toolkit are shown in Figure 4 [43].



(a) Animated radial layout. **(b)** Force-directed layout with overview. **(c)** Hyperbolic tree.

(d) TreeMap. **(e)** SpotPlot scatterplot. **(f)** Fisheye graph. **(g)** Fisheye menu.

Figure 4. Sample prefuse applications.

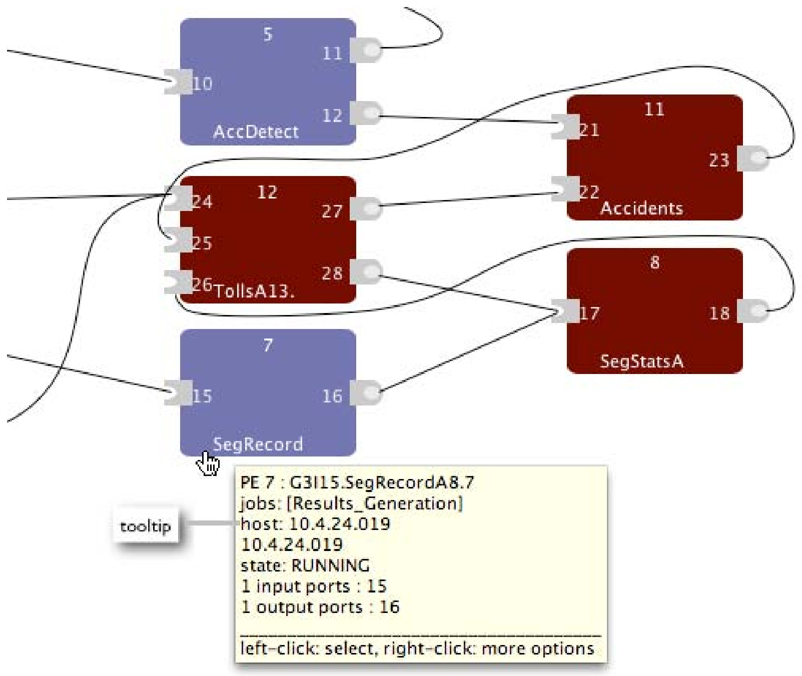
The prefuse toolkit follows the model-view-controller design pattern. This toolkit is designed to incorporate as much as possible of the existing visualization techniques to provide a comprehensive visualization system for users. Although it does have many usable visualization techniques, it is not a ready visualization tool, but rather a toolkit for the developer to use to create visualizations intended for data visualization. [43]

2.4.3 Streamsight

Streamsight is a visualization tool for large-scale streaming applications [44]. It is developed for visualizing and debugging stream processing systems composed of operators interconnected by streams. Streamsight is implemented as an Eclipse plug-in and is composed of three main subsystems: a communication component, data model managers and the visualization component. It was designed to use visualization techniques that support the dynamic and adaptive nature of streaming applications.

Streamsight supports monitoring, understanding and debugging for large-scale streaming applications. Monitored information includes performance information such as the number of packets received or Central Processing Unit (CPU) utilization. The tool can visualize the information both in real time and from saved snapshots of earlier visualizations. [44]

Figure 5 [44] shows part of the Streamsight basic view, a graph showing connections between processing elements and details of one individual processing element in a tooltip. Nodes in the graph are coloured according to different criteria, for instance, the run-time state or performance counter chosen by the user. Many interaction techniques are incorporated into Streamsight. The user can, for example, zoom and pan complex topologies, view tooltips of nodes by moving the mouse cursor over them or highlight the upstream or downstream path of an element. The user can also view the graph from different perspectives, for example, from a software or hardware perspective. The software perspective divides the system according to jobs and applications. The hardware perspective, however, divides the system into applications running on specific hosts, and from a large-scale point of view the hosts can be further divided into clusters of computational hosts. [44]



PE 7 : G3I15.SegRecordA8.7
jobs: [Results_Generation]
host: 10.4.24.019
10.4.24.019
state: RUNNING
1 input ports : 15
1 output ports : 16

left-click: select, right-click: more options

Figure 5. Part of the Streamsight view.

Streamsight supports both monitoring and visualization of the information and incorporates many visualization techniques that are helpful for interactive visualization (explained in Section 2.3). These interaction techniques include, for example, overview, filter, zoom and tooltips.

2.4.4 PortVis

PortVis [45] is a security visualization tool that can be used to visualize security-related network data, especially very coarsely detailed data about ports. The main aim of PortVis is to enable the analysis of large-scale and small-scale security events occurring in the network. PortVis uses very high-level data and is a very high-level tool. It can uncover high-level security events and count the activities but not show the activities themselves.

Figure 6 presents the PortVis application showing all the different visualization tools simultaneously. The different visualization tools (views) shown in Figure 6 are explained below. [45]

1. Timeline of the visualization
2. Main (hour) visualization, including a magnification circle
3. View of the magnification circle showing ports
4. Activity view of the selected port
5. Appearance control of the main and port displays
6. Options panel, e.g., selection of the data source to display.

PortVis aids the user by showing multiple different views of the data so that it is easy to correlate the data. It is useful for high-level analysis and for detecting patterns occurring on certain ports, but it does not provide detailed information about the security events.

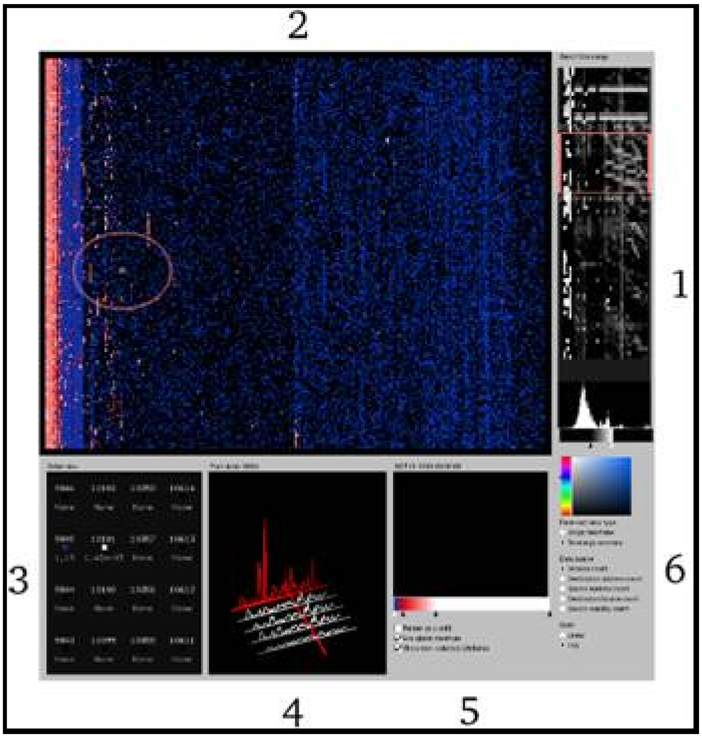
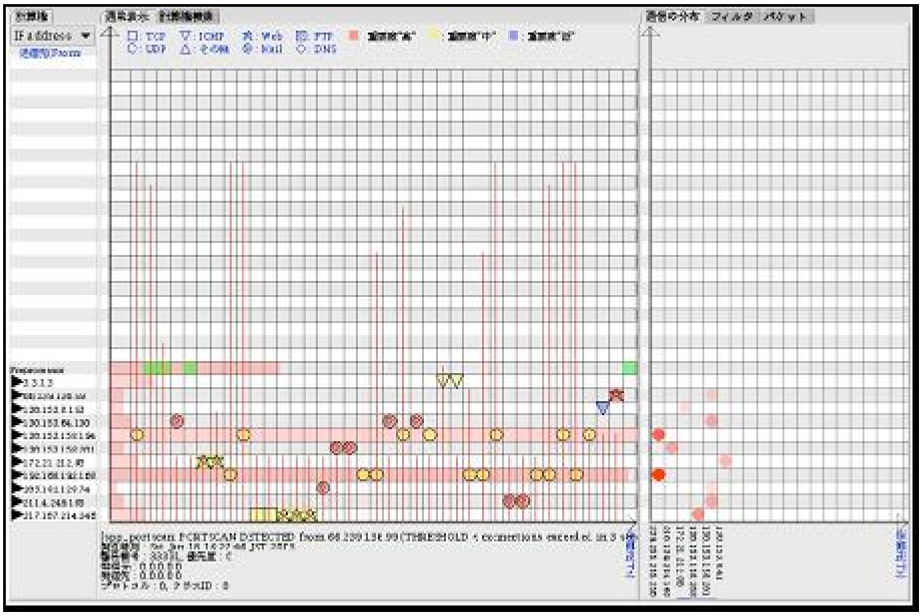


Figure 6. PortVis application showing multiple views.

2.4.5 SnortView

SnortView [46] uses Network-based Intrusion Detection Systems (NIDS) logs to present security alert information. It helps the system administrator to filter out false detections, i.e., situations in which NIDS give an alert that is either false positive or false negative. False positive regards normal traffic as an attack, while false negative does not give an alert for a real attack. SnortView uses visualization to effectively recognize the false positives in the logs instead of using traditional approaches that, for instance, customize the signature database used in the alert detection and thus not eliminating possible false negative alerts. [46]

Figure 7 shows the main frames of SnortView: the source address frame on the left, the alert frame in the centre and the source-destination matrix frame on the right. The source address frame shows the source Internet Protocol addresses acquired from the NIDS logs. The alert frame shows the time on the horizontal axis and displays alerts as coloured icons. Icons are coloured with a priority from 1 to 3 (red, yellow, blue) and their shapes visualize different kinds of attacks. The source-destination matrix frame shows the source and destination with red circles. When the user clicks on the circle the communication path between the source and the destination is highlighted with blue lines so that the user can quickly comprehend the situation. [46]



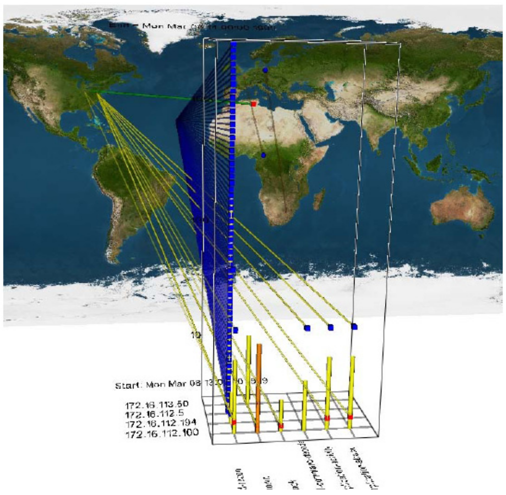
(spp_portscan) PORTSCAN DETECTED from 68.239.138.99 (THRESHOLD 4 connections exceeded in 3 [illegible]
[illegible]
プロトコル : 0, クラスID : 8

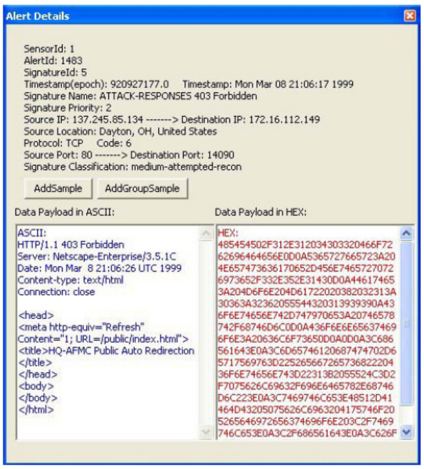
Figure 7. SnortView frames.

2.4.6 A network security visualisation prototype

Musa and Parish described a network security visualization prototype in [31] that visualizes the security alerts in three-dimensional displays. The tool also supports filtering, drill-down and playback of alerts to support the analysis of the data. A geographical view showing an attack is presented in Figure 8. This tool supports multiple visualization techniques such as a parallel coordinates plot, scatter plot, three-dimensional view and timeline animation. It also provides detailed information about the security alerts, as seen in Figure 8. [31]

The timeline animation of this tool allows the user to replay the attacks with the exact sequence in which they occurred. The tool also supports real-time monitoring of alerts. This is done by monitoring security logs and refreshing the display every second. [31]





```
SensorId: 1
AlertId: 1483
SignatureId: 5
Timestamp(epoch): 920927177.0    Timestamp: Mon Mar 08 21:06:17 1999
Signature Name: ATTACK-RESPONSES 403 Forbidden
Signature Priority: 2
Source IP: 137.245.85.134 -------> Destination IP: 172.16.112.149
Source Location: Dayton, OH, United States
Protocol: TCP    Code: 6
Source Port: 80 -------> Destination Port: 14090
Signature Classification: medium-attempted-recon
```

Data Payload in ASCII:

```
ASCII:
HTTP/1.1 403 Forbidden
Server: Netscape-Enterprise/3.5.1C
Date: Mon Mar  8 21:06:26 UTC 1999
Content-type: text/html
Connection: close

<head>
<meta http-equiv="Refresh"
Content="1; URL=/public/index.html">
<title>HQ-AFMC Public Auto Redirection
</title>
</head>
<body>
</body>
</html>
```

Data Payload in HEX:

```
HEX:
485454502F312E312034303320466F72
62696464656E0D0A5365727665723A20
4E657473636170652D456E7465727072
6973652F332E352E31430D0A44617465
3A204D6F6E204D61722020382032313A
30363A3236205554432031393939 0A43
6F6E74656E742D747970653A20746578
742F68746D6C0D0A436F6E6E65637469
6F6E3A20636C6F73650D0A0D0A3C686
561643E0A3C6D657461206874747027D6
5717569763D2252656672657368222204
36F6E74656E743D22313B2055524C3D2
F7075626C69632F696E6465782E68746
D6C223E0A3C7469746C653E48512D41
464D43205075626C6963204175746F20
5265646972656374696F6E203C2F7469
746C653E0A3C2F686561643E0A3C626F
```

a) b)

Figure 8. A Network Security Visualisation Prototype showing a) a geographical view of an attack on a local network and b) security alert details.

2.4.7 NEXThink REFLEX

One commercialized solution of security visualization is REFLEX by NEX-Think described in [47]. This system only collects a small set of parameters from the connections on the network to help the security administrator interpret the data. It is designed to help the system administrator react to security incidents in a corporate network quickly enough to prevent major incidents. [47]

Visualization is based on a grouping technique that groups the network items (i.e., applications, users, ports, hosts) or groups of network items into a tree structure. Each item has a state attribute indicating whether the item is expanded or collapsed. When it is collapsed it shows the children of the node as a group and when it is expanded it shows the children of the node individually. The user can click on an item to collapse or expand it. The system shows connections between network items with arcs connecting the nodes, e.g., an application and the ports it uses. The connections of an application can be highlighted by moving the mouse cursor over the group, thus providing more interaction. Figure 9 presents the grouping view showing grouped network items with parallel-coordinates visualization on the left-hand side. [47]

Security alarms are handled by showing the application that caused the alarm and visualizing the network items involved in it and the usual behaviour of the application such as the ports used (see Figure 9, right-hand picture). [47]

Overall, this security visualization system provides the greatest amount of user interaction and helps the administrator quickly analyse the status of the system.

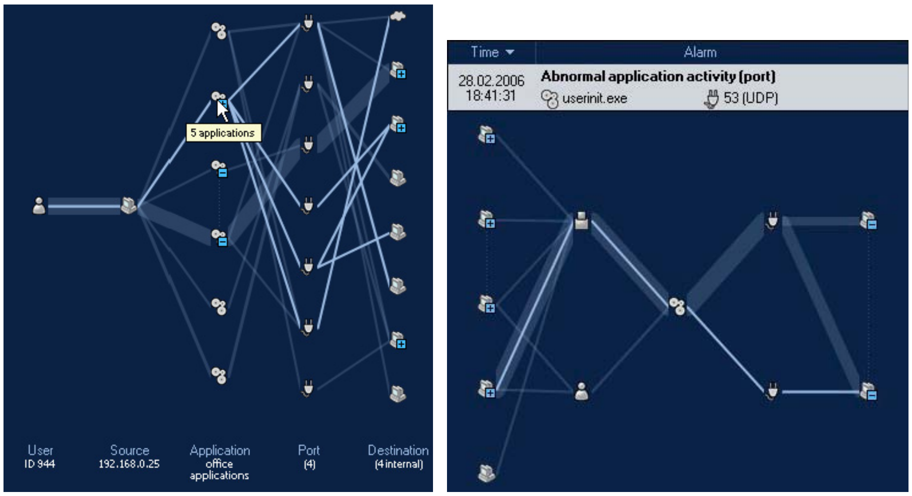


Figure 9. REFLEX solution showing a basic view and alarm.

2.4.8 Summary

Of the reviewed tools, Streamsight has many of the preferred qualities with regard to the research problem of this thesis. It combines monitoring and visualization of the needed data and offers many interaction and visualization techniques. EVolve, however, was designed to be extensible, and new sources of data and visualization techniques can easily be added.

As stated earlier in this chapter, there are many security visualization systems but only a few can be utilized directly for the research problem of this thesis. The reason so many security visualization tools were reviewed was that security and its variability is visualized in the context of this thesis as the primary noticeable quality in the laboratory smart environment [5]. Some systems included a real-time visualization option to allow the administrator to analyse the results more accurately. The usual visualization techniques used for security-related topics are timelines, scatter plots, parallel coordinates and graphs. Most of the systems are based on logs (computer logs or network logs) or on alarms generated by security systems.

Interaction was a major factor in the comparison of the tools because interaction is the base criteria for a good visualization system. Real-time visualization was also considered, as the purpose of this thesis is to provide a solution that can be used to visualize the run-time quality of software (providing real-time visualization). The comparison was performed with the information provided in the papers describing the solutions. The results of the comparison of the visualization systems and tools are stated in Table 2. The columns in Table 2 are explained as follows.

- Based on: how or where the data to be visualized can be collected?
- Interaction techniques: does the tool incorporate interaction with the user?
- Visualization techniques: list of the visualization techniques used.
- Applicability: in what context can the tool be used?
- Extensibility: is the tool designed to be extensible, i.e., can the new visualization techniques or data sources be added easily according to documents?
- Real-time visualization: can the tool visualize the data in real time when the monitored system is running?

The evaluated visualization systems used various visualization and interaction techniques to provide the information to the user as cognitively as possible. It is clear that one visualization technique is not enough to present various types of data. Visualization systems can be divided in two major categories: toolkits and task-specific tools. A visualization toolkit can be built from various techniques to provide as much support as possible to visualize different types of data. Another approach is to provide a task-specific visualization system that incorporates various visualization techniques to provide the best visualization result of the specified topic such as network security. The only commercial tool in this review was the NEXThink REFLEX tool for visualizing network security.

Most of the evaluated visualization tools were developed for a specific purpose. Only the prefuse toolkit and EVolve were designed with the option to visualize abstract information. EVolve supports visualization of somewhat abstract data through a well-defined protocol that is used mainly to make visualizations of the x- and y-axes. The prefuse toolkit provides many visualization techniques for use by the developer, but does not provide many ready visualization solutions.

EVolve provides the best support for extensibility. It supports the addition of new visualizations and data sources with a clear API. The prefuse toolkit provides many visualization techniques that can be reused in the development of a new visualization tool that incorporates some of the visualization techniques. The prefuse toolkit uses the reference model design pattern, which employs the Model-View-Controller (MVC) architectural pattern [48, p. 125]. The reference model is widely used and advocated in visualization papers and frameworks [49]. Streamsight is implemented as an Eclipse plug-in and is divided into communication, data model and visualization components. The architecture of EVolve is also divided into three main components: data source, visualization library and a core component. This kind of separation of the architecture seems to support extensibility very well.

Many of the visualization techniques found in the reviewed tools can be employed in the visualization tool for quality attribute variability. Timelines, for instance, are very useful for providing a means to look at the values of different qualities at different times. This visualization technique can be combined with other visualization techniques to provide history functions. Scatter plots, parallel coordinates and different charts can be used alone to provide information on the way a certain quality has varied over time. The prefuse toolkit has many visualization techniques that support the principles mentioned in [38] to help visualize change in time series, for example, animation. Animation techniques help the user notice changes in different quality attribute values by either changing a property (e.g. size, shape or colour) or by moving the object along the screen to a different location or a combination of these. Devices and software must also be visualized because the purpose is to visualize the quality variability of different software in different devices. Simple graph visualization can be used to visualize devices and their software in a smart environment. This allows the user to look at the quality attributes of a specific device and software as well as viewing, for instance, the security quality of all the devices or the software on a scatter plot.

Table 2. Comparison of visualization approaches.

Tool	Based on	Interaction techniques	Visualization techniques	Applicability	Extensi-bility	Real-time visualization
EVolve	logs or custom data source	yes	bar chart, scatter plot, stack, tooltip or custom visualization	run-time behaviour of Java programs	yes	no
Prefuse toolkit	abstract data	yes	radial layout, hyperbolic tree, treemap, scatter plot, fisheye graph, tooltip, and more	abstract	yes	no
Streamsight	real-time	yes	timeline, graph and tooltip	streaming applica-tions	no	yes
PortVis	information	poor	timeline, scatter plot, histogram	network security	no	no
A Network Security Visu-alisation Pro-totype	logs	yes	timeline, animation, scatter plot, parallel coordinates and geo-graphical view (3D)	network security	no	yes (1 second refresh time)
SnortView	logs	poor	tabular visualization, two-dimensional time diagram and matrix	network security	no	poor (2 minutes refresh time)
NEXThink REFLEX	logs	yes	parallel-coordinates, graphs and tooltips	network security	no	yes

3. Interactive Quality Visualization tool

The Interactive Quality Visualization (IQVis) tool is designed to visualize the variability of software quality attributes at run-time. The development of the tool starts by specifying the software requirements for the task. This chapter covers the functional and non-functional requirements specification and architecture design for the tool. This chapter first gives an overview of the visualization process and then defines the requirements of the IQVis tool. Finally, it depicts the architecture of the whole system and its components at the end of this chapter with a section describing sample visual mappings for quality attribute variability.

3.1 Overview

As stated in Chapter 2, one visualization technique is not enough to visualize different topics. This is also the situation in the case of visualizing quality attribute variability. Different visualization techniques are used to illustrate different aspects of the data. In order to visualize execution qualities such as security and performance, suitable metrics need to be defined that output a single numeric value of the quality so that it can be visualized. The development of suitable metrics for quality variability monitoring is outside the scope of this thesis. The monitoring of qualities is environment specific, and different monitoring solutions can be used in different environments. Extensibility of the tool is therefore highly preferable so that it can be reused in different contexts. This thesis does not cover the actual design of the monitoring system on the environment side, but the design of the monitor component is considered for extensibility.

Figure 10 presents an overview of the visualization process with the IQVis tool (delimited with a dashed line rectangle) in some environment. The process starts with monitoring and transmission of the data to be visualized to the IQVis tool. The visualization process is then conducted. After the Visual Mapping, the visual presentation of the data is shown to the user with the help of some visualization technique, e.g., a bar chart in Figure 10.

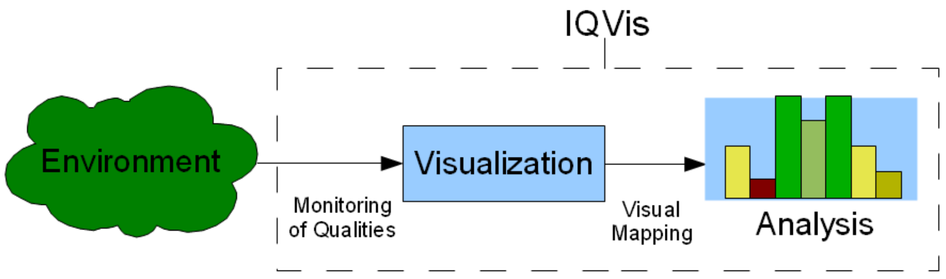


Figure 10. Overview of the visualization process.

The monitoring works as the data source for the IQVis tool. The monitored data can be provided from trace files (not in real time) or from the monitored environment directly (in real time) at run-time. The IQVis tool does not care whether the monitored data are provided in real time or not, but the IQVis tool architecture takes into account that the data can be provided incrementally or at once. The IQVis tool in Figure 10 presents the whole visualization tool. It comprises the Visualization Monitor (VMonitor), Visualization Platform and Visualization Views, which will be defined in more detail later in this chapter.

The data source components, monitors, can be built as a client-server pattern with different components, e.g., one component on the environment side (Environment Monitor, EMonitor) and one component on the visualization side (Visualization Monitor, VMonitor). By implementing the data source as a client-server pattern, the EMonitor (on the environment side) can provide the same data to multiple IQVis tools if necessary. The EMonitor can send the monitored data to the VMonitor by using some specified protocol and representation, e.g., JavaScript Object Notation (JSON, a lightweight data-interchange format) or eXtensible Mark-up Language (XML). The VMonitor parses the information, converts it into a form that the IQVis tool understands and updates the model that holds the internal representation of the data.

3.1.1 Monitoring process

The VMonitor component is the data source for the visualization and therefore a vital part of the system. The task of this component is to convert the data from the environment into an internal representation that the IQVis tool can recognize and store. Although the main problem is to design and implement the VMonitor component so that it can communicate with the IQVis tool, the monitoring process on the environment side should also be taken into account. That said, the process of monitoring information from the target environment is beyond the scope of this work, but designing and implementing a monitor component (VMonitor) to validate the visualization part is within the scope of this work.

The monitoring process, for example, can be either distributed or centralized. In distributed monitoring, every device reports the information to be monitored directly to the monitoring service, which then relays this information to the visualization tool. In centralized monitoring, individual devices do not necessarily have to notify the monitored properties; instead, one or more centralized components observe the state of the system or environment from logs or other information. The trade-off between these two choices is clear: a distributed solution demands more from the individual devices in terms of processing power and network resources, and the centralized solution may not be as accurate and reliable but does not influence the execution time of the monitored devices.

Monitoring can also be done by combining these two options. Some of the properties, e.g., critical properties such security, can be monitored directly from the devices/components and other not so important properties from system logs. Either way, the implementation decisions made in the monitoring process of the target environment (EMonitor) do not influence the design of the VMonitor component on the visualization side because the VMonitor communicates with the IQVis tool through a predefined interface. The job of the VMonitor component is to receive data from the environment side, convert it into the IQVis tool's internal representation and communicate with the IQVis tool.

An example implementation of the monitoring process is defined in [50], in which a modified OSGi (formerly known as the Open Services Gateway initiative) platform (Oscar, the predecessor of the Apache Felix) is used as the monitoring source for a resource visualization tool of OSGi-based software components. The OSGi platform can technically be seen as a dynamic Java class loader and a service registry that is globally accessible on a single Java virtual machine. The monitor agent creates an output file for the visualization tool that consists of numerical values representing the behaviour of the OSGi environment and the monitored applications (OSGi bundles, which are Java archives). [50]

3.1.2 Visualization process

The visualization process explained in Section 2.3.1 includes mapping of the data into a visual form, selection of the appropriate view to use and interaction with the user (e.g., view something in more detail or possibly filter the data). This process decides which part of the data is visualized, how the data are visualized and when the data are visualized. The visualization process has to take into account whether or not the visualization is in real time because in real time the visualization changes take place according to the run-time state of the software in the specific environment. The user may wish to stop, i.e., pause the real-time situation and analyse the current situation in more detail. The Visualization View used must be able to react according to the events coming from the model when the data changes, e.g., increment the timeline or notify the user so that he/she knows that new events have occurred.

The mapping of the abstract data into a visual form is an important phase when considering how well the data are understood by the user. Quality attribute variability can be visualized, e.g., using a time series. As said in Section 2.3.3, there are different techniques that help to analyse change in time series, e.g., magnitude of change, shape of change, velocity of change and direction of change [38]. The obvious visualization techniques for visualizing change include timelines, scatter plots and charts (with the help of animation), which can be found from a toolkit like prefuse [43]. Other helpful visualization techniques for quality variability include, for instance, a gauge showing minimum and maximum values and the current value and animation techniques.

As the user may want to use different visualization techniques for different qualities, the tool should support this by allowing the user to choose the visualization technique used for the data, e.g., by opening another Visualization View. As a single visualization technique is rarely enough to visualize all data, different visualization techniques can be grouped into Visualization Views that include at least one visualization technique and provide a view for the data. These Visualization Views are components that the user can choose to use with a specific data source. Multiple views of the same data can be used to analyse the data from different aspects. The user can view, for example, the performance of software in one view window, security in another window and a combination of these in a third window and analyse how changes in both qualities affect each other.

3.2 Requirements

This section describes the requirements for the IQVis tool. The requirements are mostly derived from the context of smart environments, e.g., security quality variability, but they also take into account that the tool could be used in different environments. Requirements are also taken from the literature study of visualization, for example, from Section 2.3.2, which presents interaction techniques. As the purpose of this tool is not to visualize large amounts of data but to visualize changes in quality, scalability is not a serious requirement of the tool. The tool shall, of course, be able to visualize tens or hundreds of different objects such as the device, the programs running on the device and the quality variability of these entities. One of the main requirements for this tool is that it is interactive so that the user can view the data the way he/she wants and thereby gain a greater insight into the data and the relationships between different qualities.

The requirements of the IQVis tool are divided into two categories: functional requirements and quality (non-functional) requirements. Functional requirements cover mainly the run-time properties of the system and quality (non-functional) requirements are mainly considered for the ease of future development of the tool. Table 3 and Table 4 list the main requirements of the tool.

Table 3 lists the functional requirements of the tool in decreasing order, i.e., F1 is the most important requirement. The requirements are numbered and the letter F in front of the number identifies the requirement as a functional requirement. The F1 requirement is the most important requirement because it states that the tool can be used to visualize the variability of different quality attributes. This means that despite the tool being used to visualize security variability (F2 requirement) in the context of smart environments in the test case; the tool is also able to visualize the variability of different qualities.

The usability and extensibility of the visualization tool are considered in the F3 requirement, which allows the user to select used VMonitor and Visualization View(s). This allows the addition of monitors and views to the tool without having to rebuild the entire tool.

Table 3. Functional requirements.

ID	Requirement	Description
F1	Ability to visualize the variability of different quality attributes	The tool is capable of visualizing the variability of different quality attributes.
F2	Security variability	The tool can visualize changes in security quality.
F3	Installation of monitor and view components	The tool allows the user to install VMonitor and Visualization View components and select the ones used.
F4	Time slider	The tool must allow the user to browse the data with a time slider so that quality variability can be seen.
F5	History functions	The tool must be able to save the current situation of the tool, i.e., the currently used monitor and view and other settings.
F6	Data storage	The tool must be able to save the data from the current session with a monitor (data source) so that the session can be replayed if necessary.
F7	Multiple views	The tool can provide multiple views of the same data so that the user can view multiple different aspects of the data at the same time.
F8	Interaction	The tool should be interactive, i.e., it should allow the user to modify the view of the data, e.g., by filtering.
F9	Animation	The tool should use animation to notify changes in the data to amplify cognition.
F10	Ability to handle quality data over a long monitoring session	The tool should be able to visualize many quality attributes in real time over a long monitoring session (performance-wise).

The F4 requirement is also important because time sliders enable the user to select a point in time or a time range in which to view the data. This allows the user to look at the quality attribute values from the whole range monitored in addition to just viewing the latest values. All the values can, of course, be shown on a timeline, but the view may become confusing with a large amount of values, and this would not comply with the guidelines presented in Section 2.3.2: "Overview first, zoom and filter, then details on demand" [34].

For the F5 requirement, history functions are provided to help the user. It is important to be able to save the analysed data so that it can be viewed at later times, and that is why the F6 requirement is presented. The user can save the data from the entire session and afterwards view the data again in playback mode.

The F7 requirement ensures that the user can use the tool as he/she wants and view the data from different aspects as needed.

The F8 and F9 requirements (Interaction) and (Animation) are derived from Sections 2.3.2 and 2.3.3 and help the user to analyse the data and amplify cognition.

The F10 requirement is included in the requirements because it is necessary for the tool to be able to visualize the variability of many software quality attributes in real time and also keep the performance at a satisfactory level in long visualization sessions. The F10 requirement is not considered to be a strict requirement as this is the first prototype version of the tool.

Table 4. Quality (non-functional) requirements.

ID	Requirement	Description
Q1	Extensibility	The system must be easy to extend with, for example, new visualization techniques (views) or data sources (monitors) so that the tool can be used in different environments.
Q2	Interoperability	Monitor and Visualization View components can exchange data with the core through a well-defined API/protocol.
Q3	Usability	The tool should be easy to use even without prior knowledge of the tool.

Table 4 lists the quality (non-functional) requirements of the tool. The requirements are numbered, and the letter Q in front of the number identifies the requirement as a non-functional requirement. Further development of the tool is considered in the Q1 requirement so that the tool can be used in various environments in the future. The Q2 requirement is introduced so that new monitors and views can easily be added to the tool. The Q3 requirement is required so that the tool is easy to use, even without prior experience of the tool. Q3 is added to the requirements because Section 2.3.3 found that usability is recognized as a problem of visualization tools.

When comparing the set requirements (functional and quality) with the qualities found in the surveyed visualization tools in Section 2.4, it is clear that no single tool can fulfil all of these requirements.

3.3 Architecture

This section describes the architecture of the IQVis tool. A structural view of the whole system is provided followed by a presentation of the structures of the major components. Visualization techniques are either designed and implemented or used from the available open source toolkits. The following design of the IQVis tool supposes that the tool is implemented with an arbitrary object-oriented language.

3.3.1 Structure

The MVC software pattern described in [48] is widely used in interactive software solutions. It divides the application into three components: model, view and controller. The model contains the platform functionality for change propagation and data actions. The view(s) display(s) information provided by the model to the user. The controller(s) handle(s) user input. The UI comprises views and controllers that together can also make changes to the model or request data from it. The basic idea is that the model notifies views when its data changes and the views update themselves accordingly. The views can also query the model about its state. The controller receives commands from the user from the view and defines application behaviour accordingly. [48]

Heer and Agrawala present 12 design patterns that have proven themselves based on the existing visualization frameworks that have been reviewed [49]. The Reference Model pattern is widely used in information visualization software, for example, in the prefuse toolkit presented in Chapter 2. The structural view of the Reference Model pattern is presented in Figure 11 [49]. The Reference model pattern provides separation of data models, visual models, views and interactive controls, and supports extensibility and reusability of the software architecture. The DataSource component in Figure 11 loads data sets to be visualized from, for example, a file or database connectivity interface. One or more data sets that can be registered to visualizations and one abstract data set can be used in multiple visualizations. Visual attributes such as shape, size and colour are separated from the abstract data set. A common approach is to create visual items, lightweight components, which represent interactive visual objects with visual attributes.

Visualization, View and Control in Figure 11 employ the standard MVC pattern and provide controls that can affect any level of the system. The Reference Model pattern can be thought of as a tiered version of MVC in which the model is divided into separate abstractions of the data and visual properties such as location, size, shape and colour. [49]

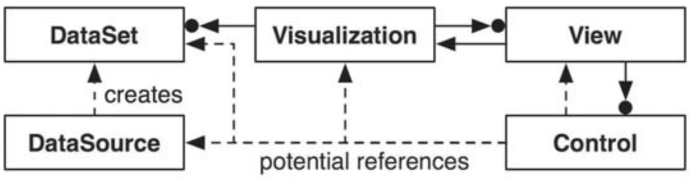


Figure 11. The Reference Model pattern.

The architecture structure of the IQVis tool is presented in Figure 12. The IQVis tool comprises three major components: VMonitor, Visualization Platform and Visualization View. The data source component is divided into two separate components: EMonitor (Environment Monitor) and VMonitor (Visualization Monitor). The EMonitor monitors the environment and sends the monitored data to the VMonitor, which then updates the data to the Visualization Platform. The Visualization Platform takes care of data storage and functionality, and encompasses the Model-View-Controller (MVC) pattern. The monitor component is selected by the platform at run-time to provide data to the tool and can thereby be replaced by another monitor component. This is realized by defining a monitor interface in the IQVis tool. The visualizations part is also separated so that new visualizations can easily be added to the tool in the form of separated components. This architecture is designed to meet the requirement Q1 of extensibility described in Table 4.

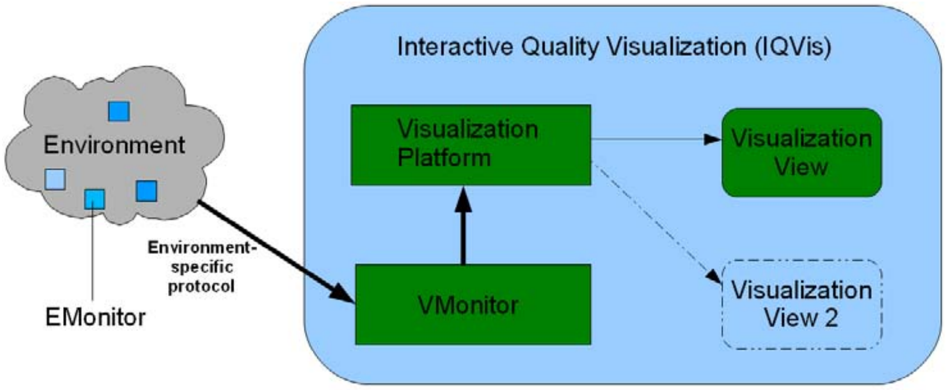


Figure 12. Conceptual architecture of the visualization system.

The Visualization Platform is responsible for the construction of the visualization, i.e., data transformations, visual mappings and view transformations explained in Chapter 2. The Visualization Platform model receives the data elements from the VMonitor component and stores the data with timestamp values so that they can provide history data for the view(s) and not just the latest values. The visualization tool supports multiple views to be constructed from the same model. The view(s) can be built from different visualization techniques, e.g., timelines, scatter plots or graphs.

The architecture of the IQVis tool follows the Reference Model and MVC patterns in multiple ways. First, the Visualization Platform of the tool follows the MVC pattern and provides functionality for selecting used monitors and visualizations views, and handles the internal data representation and communications. The Visualization Platform is different from the Reference Model's DataSet because it only supports one DataSet at a time from the VMonitor. Second, the VMonitor component(s) is/are the data sources and the visualizations views are the views that are produced from the abstract data. Third, the Visualization Views are constructed from visualizations, different visualization techniques, which refer to visual items. These separations allow the tool to be extended easily.

Figure 13 presents the architecture of the IQVis tool in a component diagram. The *Visualization Platform* comprises Model and ControlView components. The ControlView component is a hybrid version of the Controller and the View of the MVC pattern, and the Model component is named as the MVC pattern assumes. The Controller and the View of the MVC pattern are combined into a single ControlView component graphical user interface (GUI) component because it is only used to select the used monitor and Visualization View components and possibly save the data from the model.

The Visualization Platform makes up the DataSet and part of the Control of the Reference Model pattern. The Visualization Platform uses a configuration file that lists the available monitor and Visualization View components. The Visualization Platform therefore handles mainly configuration, data representation and storage. The Visualization View(s) make(s) up the real visualizations by itself/themselves or by using some visualization component (Vis in Figure 13) or open source visualization technique.

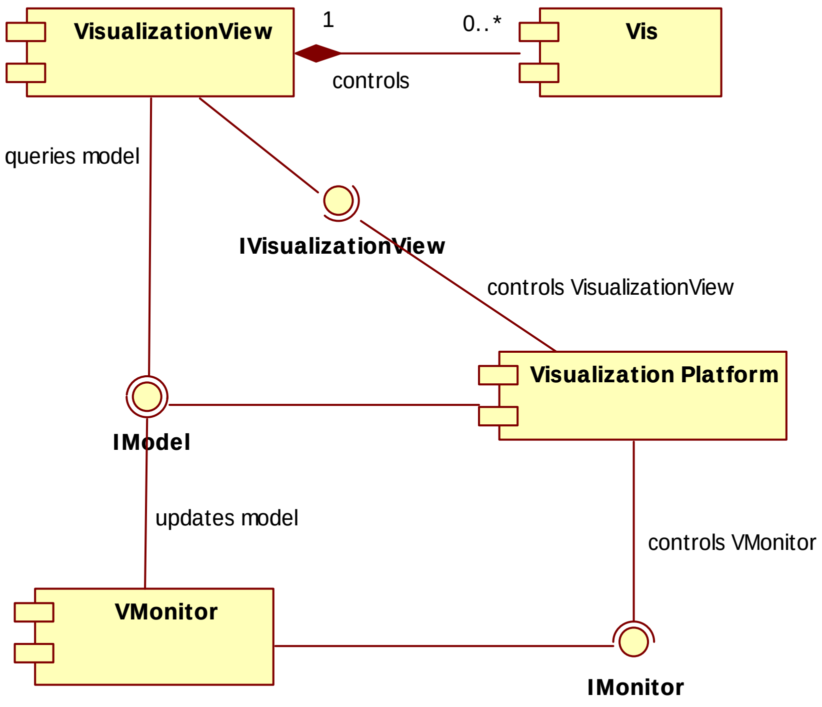


Figure 13. Architecture of the IQVis tool.

The *Visualization View* in Figure 13 corresponds to the view component of the Reference Model pattern. The *VMonitor* in the Figure 13 component corresponds to the DataSource component of the Reference Model pattern and provides the data to the Visualization Platform through the IModel interface. The Visualization View and VMonitor components can both be installed at run-time. Essentially, the *VMonitor* modifies the model directly and the *Visualization View(s)* update(s) itself/themselves accordingly by querying the model directly instead of communicating through the controller as in the MVC pattern.

Multiple IQVis applications can be opened from the same monitoring component, if the data source (VMonitor and EMonitor) is split into a client-server implementation. Multiple EMonitors can be used on the environment side and then either use different instances of the IQVis tool or a combine the information in the VMonitor component. The Visualization View(s) can be constructed separately from each of the monitored quality attributes, or all the quality attributes can be shown in a single Visualization View.

The VMonitor, model and Visualization View all provide interfaces so that the communication between these components can be assured. This makes it possible to change the VMonitor and Visualization View components without having to rebuild the entire software. The VMonitor and Visualization View components can be compiled on their own when the required interfaces are available. These interfaces are described when the different parts of the IQVis tool are discussed in more detail in the following section.

3.3.2 Visualization Platform

The Visualization Platform consists of two major components: Model and ControlView. Together, these components make up the core functionality of the IQVis tool. These functionalities include saving the current state of the platform and saving the monitored data up to a given moment so that it can be used at a later time. The Visualization Platform is designed to meet the quality requirements (Table 4) Q1 extensibility and Q2 interoperability, and it therefore defines an interface for the use of its functionalities. The Visualization Platform is also designed to meet functional requirements F3, F5, F6, F7 and F10 described in Table 3.

The model is used by the VMonitor and Visualization View components. The Visualization Platform allows the user to choose a VMonitor to be used as the data source. The user can change the current VMonitor component, causing the model to be cleared for the new monitored data. The Visualization Platform supports only one monitor component at a time because it saves all the monitored data and it could therefore slow down significantly if several environments were visualized at the same time. This is considered because of the F10 requirement. If the user wishes to analyse/monitor another environment, he/she can always open another instance of the IQVis tool. The following describes the different functionalities provided by the Visualization Platform and its interfaces.

Data representation of the model

The data representation model of the Visualization Platform’s model follows a graph-type representation in its naming convention. This data representation model was chosen because it can provide a very good representation of the different entities found in environments where this tool can be used, e.g., devices, software and their relationships. A similar approach was used in NEXThink REFLEX, as described in Section 2.4.7.

The model separates three different types of entities: nodes, edges and attributes. This data representation model partly follows the Relational Graph and the Proxy Tuple patterns presented by Heer and Agrawala in [49]. The data representation model uses the basic form of the Relational Graph by providing different tables for different entity types and data access that is similar to the Proxy Tuple.

Nodes represent all the entities found in the environment, e.g., devices, sensors, smart agents, software and so on. Nodes have some required properties that the platform and predefined visualizations use and, in addition, they can also have customizable properties. Nodes are used for visualizing the different devices or software being monitored in the environment so that the user can, for example, select a device that he/she wants to look at in more detail and analyse its quality variability. Nodes and edges are not required to be able to visualize quality attributes and their variability, but they help the user obtain an overview of the environment and the software currently being monitored. The pre-built properties of nodes are:

- id: a unique identifier for the object
- label: a text used to represent this object
- time: the time this object was modified
- type: type identifying this object, e.g., device, software.

Edges represent the connections and dependencies of nodes. Edges can be used, for example, to describe that some software runs on a certain device or that a device is connected to some smart environment. Edges can also be used to describe the metadata of the connection, e.g., a device connection to a smart environment is via the Transmission Control Protocol / Internet Protocol (TCP/IP) and some arbitrary encryption is used. Edges have the following pre-built properties:

- id: a unique identifier for the edge
- label: a text used to represent this edge
- time: the time this object was modified
- source: the source object id for this edge
- target: the target object id for this edge
- directed: Boolean, whether or not this edge is directed, i.e., the connection between nodes.

Attributes are the most important entities because they describe the different quality attributes of software. Any entity can have attributes, but attributes do not require the existence of nodes and edges. Attributes are defined to be attributes of a certain id that maps to the id values of nodes and edges. The attributes can be used without nodes or edges. Attributes also have some pre-built properties that must be defined as follows:

- id: a unique id that may map to a node or edge id, i.e., a foreign key to a node or edge id
- type: describes to which quality attribute this attribute corresponds, e.g., security or performance. The platform recognizes quality attributes, but this property may also be used to support possible use cases other than quality attributes in the future.
- time: the time this attribute was modified
- value: the value of this attribute.

Attributes are separated from the actual nodes and edges because attributes are more likely to change over time. A separation between these three entities is therefore necessary so that the model can store all the changes in these entities without having to duplicate all the entities that do not change frequently. This separation is made because of the F10 functional requirement (Table 3). When there are fewer objects in memory, the system will respond faster to data queries.

Updating the model

After the selected VMonitor component has been loaded successfully onto the platform, it is started and the VMonitor component starts to update the model using the changeNode, changeEdge and changeAttribute methods described in the IModel interface (Table 5). These three methods take entity types defined earlier (nodes, edges and attributes) and properties as a parameter. The parameter is supposed to be an object that has the required properties. The model is designed to support timeline features, so it saves all the changes made to the model. The model uses a predefined period of time to determine whether or not the change in the model is a new event. This period of time could be set to, for example, 1000 milliseconds depending on the accuracy requirements. The duration determines how accurately the user can view the changes in the entities and affects the memory requirements of the tool. This is because the model saves all the events identified as new events into the memory, and every new event therefore reserves more memory.

The entities are expected to have a time value of when they were monitored in the environment. This value may not be available at all times, however, so the

VMonitor component adds a time value to the data regardless of the given time value so that the data will not violate the Visualization Platform data format. Only one timestamp should be used, however, either the one from the EMonitor or the one from VMonitor, so that the events can be correctly compared and placed in the timeline of events. The VMonitor component changes the given time value, e.g., date value, into the internal representation, which is a number representing milliseconds between midnight on January 1, 1970, universal time, and the time specified in the time value. This representation is also known as UNIX time. UNIX time is chosen over a textual representation of time and date so that the model can make a direct comparison of the numerical time values. The UNIX time is changed into a textual representation in the visualization component. If the EMonitor adds a time value to the monitored entity, it must also use the aforementioned format of time so that the model can compare the time values correctly.

Querying the model

The Visualization View components and the Visualization Platform can make queries to the model to request data from it. The query methods are simple methods that can be used to obtain entities according to different limits. A more advanced querying language was not included because it was not seen as a necessity and the methods defined are adequate for the needs of this tool.

The model uses the following reasoning when entities are queried with a time value or time range: if there is no occurrence of the entity at the given time(s), the model takes the last occurrence of the queried entity and returns it as the latest one. This is because the model only updates an entity when the time difference between changes is bigger than defined, as explained earlier. This time limit can be set or checked with the setTimePeriod and getTimePeriod methods.

The model data can be saved and loaded at any time. The model provides a getJSON method for returning a JavaScript Object Notation (JSON) encoded String of the model data that can be saved into a file. The model data can be set by using the setJSON method, which loads a previously saved JSON String of model data. When the setJSON method is used, the model sends an event that the model has been loaded. The getHistory method provides the events that have occurred in an array that can be used to show events in the order in which they occurred.

When querying the model with a time parameter, it is important to remember to use the time value relative to the time values inserted into the model, i.e., relative to the time received from the monitored environment. It cannot be assumed, for example, that a call requesting entities with the current timestamp from the device on which the IQVis tool is used will return the latest entities in the monitored environment. Although this is taken into account when designing the current components that use these querying methods (and by wildcards), this has to be considered when new components are designed and implemented in the future.

The getOccurrences method (Table 5) returns all the timestamps related to changes in nodes, edges and attributes to provide easy access to the timestamps of occurred events. The rest of the query methods involve querying the types of objects explained earlier. An empty string equals a wildcard in the id and time parameters, i.e., an empty string substitutes any possible value. The type parameter cannot contain wildcards because this would make the returned object more complex to handle, and the user now knows the kind of entity he/she will receive.

The methods for querying a single entity type can contain the following parameters: a type parameter “node”, “edge” or “attribute”, an id of the entity and one or two time values. The time value is represented as a number, so that the model can quickly compare different times values and fetch the queried entity from the data repository. The methods of querying a single entity type are: getEntityAtTime and getEntityBetweenTimes. The getEntityAtTime method simply returns the newest entity at a given time. The getEntityBetweenTimes method returns all the entities, which have timestamps between the times given in the parameters, in an array.

The methods for querying attributes are: getAttributeAtTime and getAttributeBetweenTimes. These methods are helper methods for querying attributes of a specific entity. Attributes can of course be queried with the methods for querying a single entity, but it is necessary to know the id of the specific attribute for these methods. The methods for querying attributes are needed because attributes are stored so that the attribute id maps to the node or edge id, and one attribute id can contain multiple attributes of different types. These two methods return an entity’s attribute of a given type, e.g., a security quality of a device. Table 5 summarises the interface IModel by presenting the name, input and output values, and descriptions of all the methods of the model explained earlier.

Table 5. IModel interface description.

Name	Input	Output	Description
clear	-	-	Clears the model so that it can be used with another monitor component.
setTimePeriod	Number value	-	Sets the period of time for an event to be considered a new event.
getTimePeriod	-	Number	Returns the current period of time for an event to be considered a new event.
getJSON	-	String	Saves the current data in the model into the JSON format String.
setJSON	String data	-	Sets the model data from the given JSON String.
getHistory	-	Array	Returns the history of events dispatched. Used to provide the order of occurred events.
getOccurrences	-	Object	Returns the timestamps of nodes, edges and attributes in an object.
changeNode	Object	-	Changes the given node to the model with the given timestamp.
changeEdge	Object	-	Changes the given edge to the model with the given timestamp.
changeAttribute	Object	-	Changes the given attribute to the model with the given timestamp.
getEntityAtTime	String type, String id, Number time	Object	Returns the entity type requested with a given id at a given time.
getEntityBetweenTimes	String type, String id, Number time1, 2	Object	Returns the requested type of entity with a given id at a given time range.
getAttributeAtTime	String id, String type, Number time	Object	Returns attributes of a given type and id (an id maps to a node or edge id) at a given time.
getAttributeBetweenTimes	String id, String type, Number time1, Number time2	Object	Returns attributes of a given type and id (an id maps to a node or edge id) in a given time range.

3.3.3 VMonitor

The VMonitor component is the data source for the IQVis tool. VMonitor components can be loaded at run-time and activated by the user. Only one VMonitor component can be activated at a time because of the aforementioned performance issues. The VMonitor components can communicate with the Visualization Platform and the model through the IModel interface. The VMonitor components are also designed to meet the F3 requirement (Table 5). The VMonitor components are listed in the configuration file and can be installed on the platform. The VMonitor component uses the setTimePeriod method defined by the model to set the monitoring accuracy for the specific environment.

After the VMonitor component has been loaded successfully, the Visualization Platform calls the start method with a model component that implements the IModel interface (Table 6) as a parameter. After this, the monitor starts to update the entities to the model. The VMonitor must use at least the required properties of the model's data representation. The VMonitor component can be stopped with the stop method, which stops the monitor from sending updates to the entities. The monitor component must also support showing and hiding of its configuration GUI by calling the showConfiguration or hideConfiguration method. The configuration GUI of the monitor component is not predefined, but it can show properties such as update frequency, status of the monitor's (VMonitor) connection to the environment (EMonitor), etc. Table 6 summarises the required IMonitor interface.

Table 6. IMonitor interface description.

Name	Input	Output	Description
start	IModel	-	Starts the monitor component. The monitor then starts to update data to the model by calling the change methods described in the IModel interface.
stop	-	-	The monitor stops updating the model. This is usually called when the monitor is changed. After that the monitor is deleted.
showConfiguration	-	-	Shows a GUI for configuring the monitor. The user can, for example, configure where the server-side of the monitor is located.
hideConfiguration	-	-	Hides the configuration GUI for this monitor.

Figure 14 presents a class diagram for a sample monitor component designed for the context of a Smart Space [5]. The monitor comprises two separate components, one of which is located in the Smart Space and the other on the visualization side. The visualization side monitor implements the IMonitor interface and communicates with the Visualization Platform. It receives monitoring data in JSON format via a socket connection from the EMonitor component from the environment side (smart environment) and parses the data into the data representation described earlier in the Visualization Platform description in Section 3.3.2.

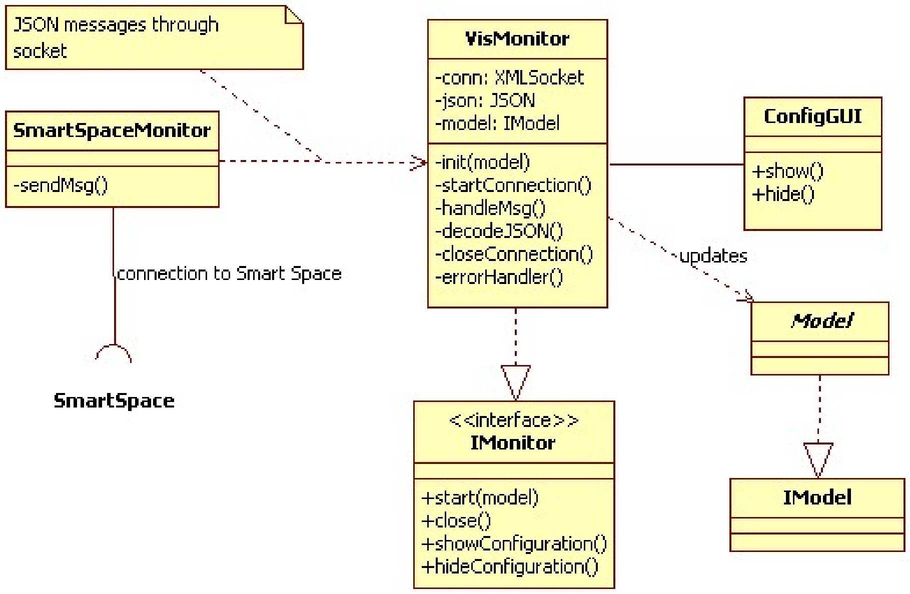


Figure 14. Class diagram of the VisMonitor component.

3.3.4 Visualization View

The Visualization View component takes care of the visualization itself, i.e., mapping the data into a visual form. The Visualization View component obtains data from the model and constructs the visualization from it. The Visualization View can use many visualization techniques to realize the visualization by using various visualization components. These visualization components are basically the different visualization techniques described in Sections 2.3 and 2.4. These visualization components can be implemented from available open source visualization toolkits, e.g., prefuse [43], or designed and implemented for a special purpose. The Visualization View component is designed to meet functional requirements F1, F2, F3, F8 and F9 described in Table 3.

The Visualization View component decides how different entities, quality attributes and their variability are visualized. Different views can be used for different purposes and contexts. The view should also provide interactivity with the user by allowing him/her to explore the visualization, look at finer-grained details on demand and provide overall support to the principles presented in Sections 2.3.2 and 2.3.3.

In order to be able to create and add these Visualization Views easily to the platform in the future, an interface is required. The interface provides functionality to add the same kinds of entities as defined earlier in the model description. The Visualization View interface also provides methods for analysing the data at different points in time by using the showValuesAtTime and showValuesBetweenTimes methods. Table 7 summarises the specification IVisualizationView.

Table 7. IVisualizationView interface description.

Name	**Input**	**Output**	**Description**
start	IModel	-	Starts the Visualization View component. The view asks the model for the latest values and starts listening for change events.
stop	-	-	The Visualization View stops listening for change events from the model.
modelLoaded	Event	-	The Visualization View is informed that the model has been loaded. The view updates the latest data from the model.
modelCleared	Event	-	The model instructs the view that it has been cleared, e.g., a new monitor has been activated. The view clears itself.
nodeChanged	Event	-	The view is informed of a change in a node element; the Event contains the changed node.
edgeChanged	Event	-	The view is informed of a change to the edge element; the Event contains the changed edge.
attributeChanged	Event	-	The view is informed of a change to the attribute element; the Event contains the changed attribute.
showValuesAtTime	Number time	-	Instructs the view to show data at a given time. The view obtains the values from the model.
showValues-BetweenTimes	Number time1, 2	-	Instructs the view to show data between two given times. The view obtains the values from the model.

Figure 15 presents a description of the Visualization View. Although this view is designed to fulfil the requirements defined earlier and is supposed to visualize Smart Space qualities, it should also be applicable to other environments. The only question is whether the design of the Visualization View fulfils the requirements of other contexts, e.g., how the mapping of quality attribute data into a visual form is realized. In another context, another visualization technique might be preferable to the ones used in this context. The implementation and actual visualization techniques used will be discussed in more detail in the implementation chapter.

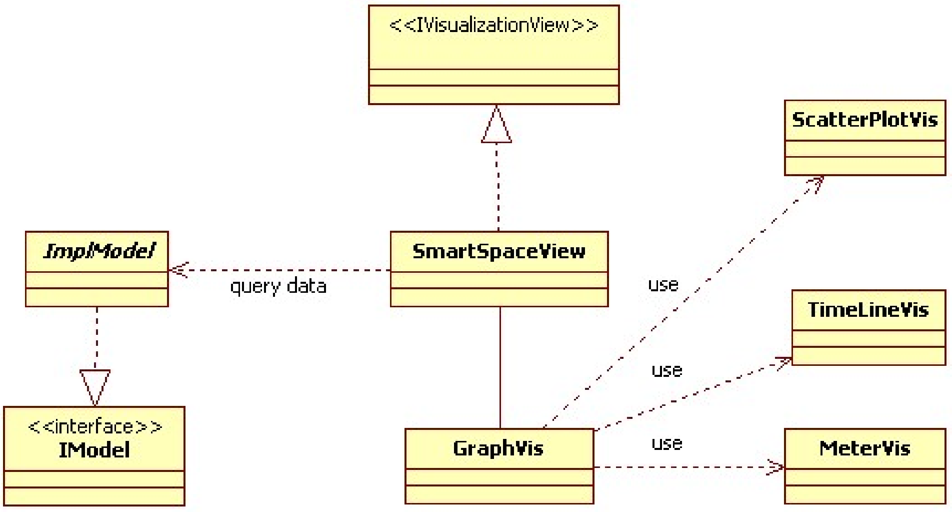


Figure 15. Simplified class diagram of the SmartSpaceView component.

3.3.5 ActionScheduler

As views update themselves when the model sends events, the view components require a scheduler to manage the data changes in the view accordingly. The ActionScheduler is necessary so that the changes to the data can be shown in the right order, and possibly in a sequence, rather than showing all the changes at once. Many quality attributes may change in a period of one second and the user may not notice the change if these changes are shown all at once. The design of the ActionScheduler can follow the mediator design pattern, i.e., it partially handles the communication between the model and the Visualization Views' components.

The Visualization Views attach the change events from the model to the ActionScheduler's handle method (Table 8), which determines the kind of change that has occurred from the type of event. The ActionScheduler then schedules the change and calls the appropriate method in the Visualization View when the view has finished animating possible earlier changes. The ActionScheduler receives notification when the view's last animation has stopped. If there are no commands in the queue when the model sends a change event, the ActionScheduler immediately calls an appropriate method in the view. The paused method defines whether or not the ActionScheduler mediates messages. When it is set to paused, it just stores the values into the history array. When the ActionScheduler is unpaused, it continues to mediate events from the point in time when it was paused.

The ActionScheduler also makes it possible to replay the events from a previous monitoring session or a point in time in the current monitoring session, in the same order as they occurred. The setHistory method (Table 8) is used to set the history of events and corresponds to the getHistory method of the model defined in Table 5. The ActionScheduler keeps a history of the events the model sends so it can provide them from a point in time if needed. Table 8 summarises the required interface for the IActionScheduler.

Table 8. IVisualizationView interface description.

Name	**Input**	**Output**	**Description**
ActionScheduler	IVisualizationView	-	Constructor. Takes a view as a parameter.
handler	Event	-	This method receives notification of the change events from the model.
endScheduler	Event	-	This method is called when the view has finished animating a change.
paused	Boolean	Boolean	This method pauses or unpauses the ActionScheduler from sending events.
setTimeRange	Number fromTime	-	Sets the last occurred event to the given time parameter. When the ActionScheduler is unpaused, it continues from this point in time. It uses the history array set in the setHistory method.
setHistory	Array	-	This method sets the history of events that have occurred. It is used when the previous monitoring session is used as a source to provide a history of events.

3.4 Visual mapping

This section discusses how quality variability can be shown in visualizations in a way that helps the user to see the patterns and relationships between different qualities. Visual mapping is the most important face in visualizations because it determines how different data are represented by visualization techniques. As said in [28, p. 6], visualizations can amplify cognition by perception, and visual mapping is responsible for realizing this. Different visualization techniques enable the user to view the data from different perspectives. The visual mapping partially meets the functional requirements F1 and F2 (Table 3) because these requirements are met with the help of visualization techniques. Quality requirement Q3 usability (Table 4) must also be considered so that the use of the chosen visualization techniques is intuitive. Different visualization techniques that are useful in visualizing quality variability are presented in [52], where the use of the IQVis tool is discussed.

Different visualization techniques require different types of data to be shown, for example, a scatter plot (see Figure 4 in Section 2.4.2) requires one or two numerical values of data to be mapped to both of the axes (x and y), a label for the items and an optional numerical value representing the size of each item. When thinking in terms of visualizing variability, the second numerical value can be replaced with a time value, as time can easily be mapped to a numerical value. A treemap (see Figure 4 in Section 2.4.2 or [29]), however, requires two numerical values that map to the size and colour, and a number of textual values to define the hierarchies of the map. Animation techniques can be used to provide the element of change in visualization techniques that cannot visually represent it so clearly, e.g., a scatter plot.

3.4.1 Graphs

Graphs can visualize relationships between different entities. Graphs are chosen to visualize the devices and software present in the smart environment. Different layouts for graphs allow the user to view the topology of the environment at different levels. Figure 16 presents a graph showing devices and applications joined in a smart environment.

The graph-type presentation was also used in the visualization tools that were discussed in Section 2.4, for example, Streamsight [44] used graphs to show connections between streaming application processing elements, and NEXThink

REFLEX [47] used a grouping technique to present network items in a tree structure. A graph can be presented in a tree structure. Graphs can also convey information about the communication channel in edge types.

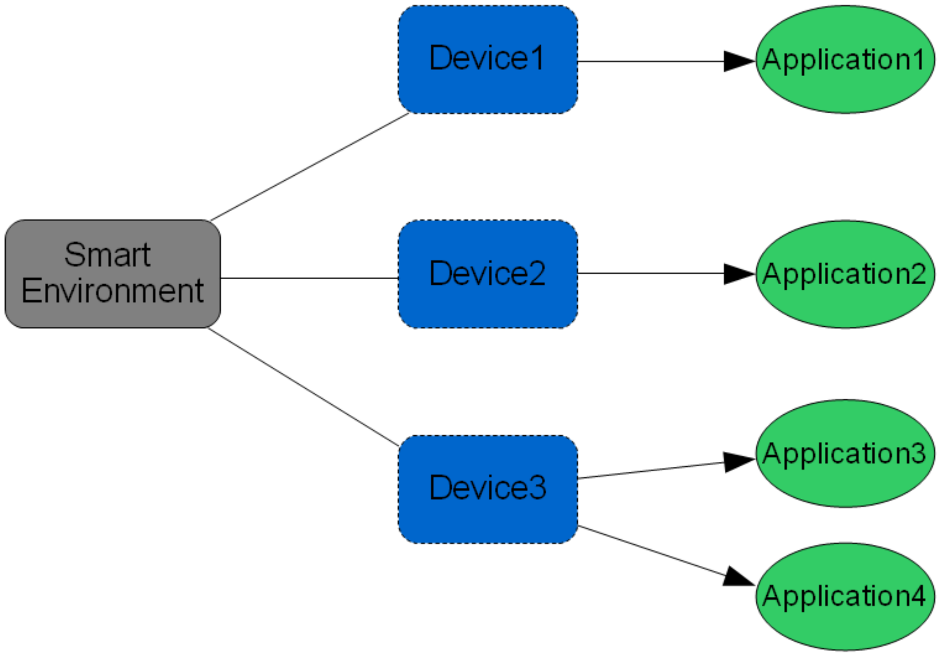


Figure 16. Graph representing smart environment devices and their applications.

3.4.2 Treemaps

A treemap is a two-dimensional, space-filling approach for visualizing a tree structure so that certain attributes of a node are mapped to the size and colour of the nodes visualized as rectangles. Child nodes are visualized as rectangles inside the parent node so that the whole area of the parent node is used. A treemap can visualize change in various ways. It can change either the size or colours of the items on the map. The size of an item is the usual way of representing relationships between some values. Colours can also be used to visualize how often the size of the object changes by, for example, mapping the colour of an entity to red when its properties change often and to green when its properties change rarely.

Treemaps are used in, for example, GAMMATELLA [53], which represents different kinds of program-execution data in order to support the analysis of program behaviour. GAMMATELLA uses treemaps to present the system level, the most abstracted level in its visualization. In GAMMATELLA, treemaps keep the same layout of colours in different nodes so that a possible non-uniform appearance of nodes will not cause confusion to the user. GAMMATELLA uses hue and brightness components to convey one- or two-dimensional information. The hue component uses red, yellow and green as the colours for conveying concepts of danger, caution and safety in the same way as traffic lights do. Different kinds of colour spectra can be used to account for different types of colour blindness. The brightness component maps the brightness from the min and max value of the information. The root node of the treemap represents the whole system, and the size of leaf nodes is determined by the number of executable statements in the source file it represents. Figure 17 [53] represents a treemap visualization from GAMMATELLA. [53]

Treemap visualization can be used to visualize all the quality attributes of all the devices and software in the smart environment, e.g., if the user wishes to see the current situation in the environment. As seen in Figure 17 [53] different colours and levels of brightness can easily be spotted from treemap visualization.

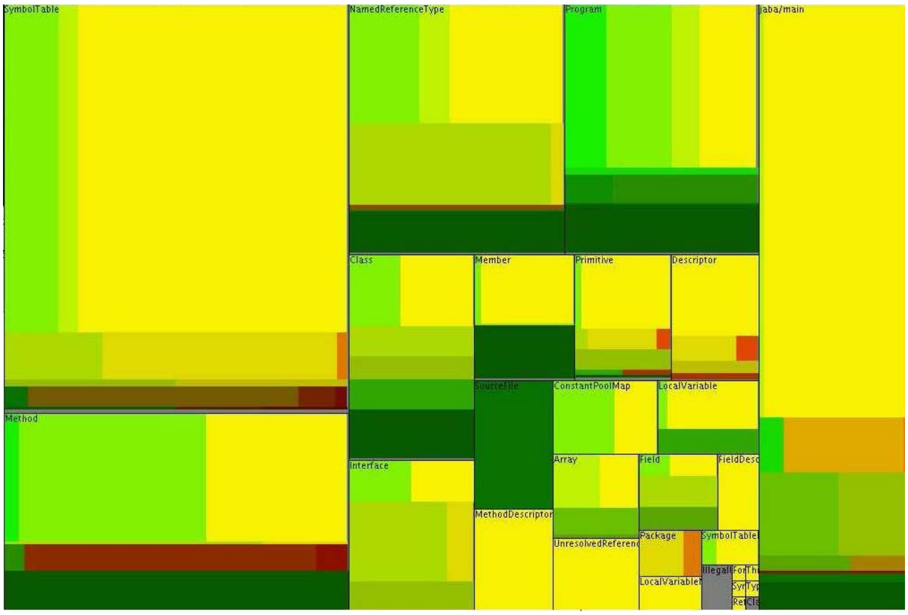
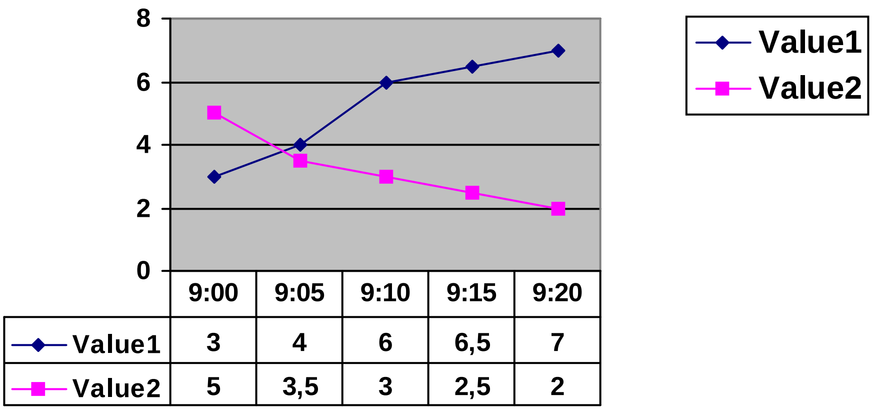


Figure 17. Treemap visualization from GAMMATELLA.

3.4.3 Timelines and scatter plots

Timelines are useful for presenting changes in values over some time period because the user can see the value. Timelines are basically normal line charts in which the x-axis value can be timestamped to illustrate the change in value over some time. Figure 18 presents a common line chart from Microsoft Word, showing a line with markers displayed at each data value. As seen in Figure 18, this kind of visualization enables the user to see how the values have changed over time and how they may affect each other. If the visualized values differ greatly, the velocity of change can be distorted, for example, if both values increase by 10 per cent but one is 10 times bigger at the start, the quantitative scale will not show the correct velocity of change [38]. This problem can be fixed by changing the quantitative scale into a logarithmic scale in which two data sets that exhibit the same rate of change will also exhibit the same velocity of change, or slope, despite the differences in values [38].

Scatter plots are very similar to timelines, except that they do not have lines connecting the different values. Scatter plots can convey information, e.g., with size, colour and shape. Although scatter plots can only visualize one point in time, it is possible to convey change information by animating the values or by changing the transparency of the values, for example, the older the measurement, the more transparent it becomes.



	9:00	9:05	9:10	9:15	9:20
Value1	3	4	6	6,5	7
Value2	5	3,5	3	2,5	2

Figure 18. Line chart.

3.4.4 Meters

Meters can be used to visualize an individual value, e.g., quality, risk or safety level. A visualization technique that is probably familiar to many is an analogue meter resembling an analogue clock or a car dashboard gauge. Another visual form of meter is a bar meter showing the value with coloured bars. The change in values, i.e., qualities, can be visualized by moving the pointer to another location on the gauge with animation. This helps the user recognize the changes as they occur. Meters are specially designed to meet the functional requirement F2, security variability (Table 3).

Figure 19 shows a conceptual image of a meter gauge on the left-hand side and a bar meter on the right-hand side. Both meters in Figure 19 illustrate meters that would be used to visualize a quality that should be below a threshold, visualized with a green line. The colours of the meters can be used to convey information about which values are good (green), which values are not good (red) and which values are in the middle (yellow). In Figure 19, for example, the background of the meter and the font on the left-hand side are green because the value of the meter is below the threshold (green line). The value of the right-hand side meter, however, is bigger than the threshold (green line), and the font is red. This red-yellow-green mapping of colours is probably well known to most people, from traffic lights, and therefore provides previous experience, which makes the software intuitive to use. Different kinds of mappings can be used to account for different types of colour blindness.

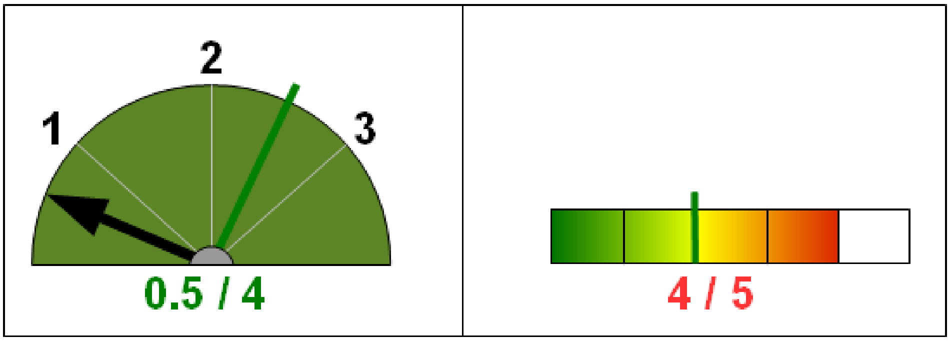


Figure 19. Conceptual view of meters.

The drawback of using meters is that they can usually only visualize one quality at a time. Meters are good for visualizing change in one individual value. They can be used to show, for instance, a real-time value of some individual quality in an overview of a device. Changes can be shown by, for example, animating the pointer to a new location and changing the colour of the meter. By adding multiple meters (or multiple pointers on one meter), it is possible to visualize multiple different qualities at the same time. Overall, meters are not as good for analysing changes over time as timelines because it would be necessary to follow multiple changing objects on the screen at the same time, which was found to be a weakness in human perception capabilities in Section 2.3.3.

3.4.5 Time slider

Navigation plays a fundamental role in applications that support history actions. Heer et al. [54] state that undo and redo (or back and forward) are found to be the most common navigation operations in many applications. Another approach to navigation is the timeline slider, which Heer et al. compare to a “time travel” metaphor. Another important aspect of history systems is the ability to search and filter history by means of metadata such as time, action and bookmarks. [54]

The time slider is specially designed to meet the functional requirement F4, time slider (Table 3). Timeline slider navigation is the most suitable selection for the navigation in the IQVis tool history of the monitored qualities because with it the users can jump to different times or choose time ranges in which to analyse the quality attributes. In the context of viewing the quality attribute adaptations in the environment, there is no reason to support, for instance, editable histories as explained in [54], which allow the user to modify past actions. This is because all the actions that influence visualization come from the monitored environment.

Figure 20 presents a conceptual view of a time slider that shows a selected time range with arrow pointers and times above them. Another (simpler) version of this slider would be to just have one arrow and the user select only one point in time, i.e. vertical lines in Figure 20. It is clear that the time slider should support some sort of selection of the time period shown because as events occur in the monitored environment the time range grows, for example, if the time slider in Figure 20 has a fixed width, then the space between different timestamps (the time between different timestamps is 30 minutes in Figure 20 decreases when the time range itself grows. As the vertical lines in the time slider move closer and closer to each other the time slider component can become fuzzy.

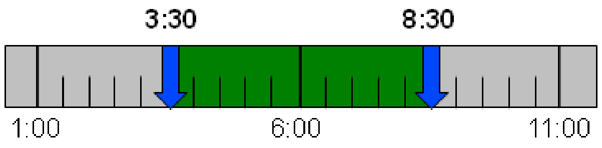


Figure 20. Conceptual view of a time slider.

The fuzziness of the slider can be taken into account by, for example, allowing it to zoom in and out of selected sections on the time slider. Zooming in and out enables the user to view smaller or larger periods of time. Figure 21 presents the zoomed in view of the time slider in in Figure 20.

The time slider in Figure 20 shows a ten-hour time period, and Figure 21 shows a more detailed view of the selected five-hour time period in the Figure 20. The sliders presented in Figure 20 and Figure 21 conform to the requirement of the F4 time slider (Table 3) with the history functions presented earlier in the model description. History functions allow querying of data at a certain point in time or a time range.

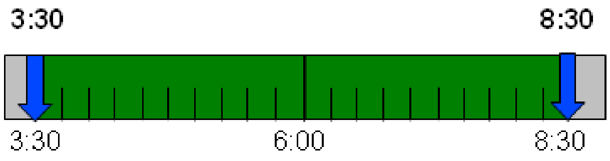


Figure 21. Conceptual view of a zoomed in time slider.

Further more, the time slider could also show the changes that have occurred, for example, in qualities in the timeline itself, so the user can quickly see where the changes have occurred on the timeline and zoom into that time period. The addition of visual cues into user-interface components has been studied by Willet et al. [55]. Visual cues help to make timeline control more user friendly and also help the user to gain an idea of when and how often the changes have occurred.

4. Implementation and testing

This chapter provides a description of the implemented IQVis tool and a case study using the tool in a smart environment. The chapter also looks at the testing of the tool with an implemented VMonitor component to demonstrate, validate and test the IQVis tool in a smart environment. The aim of the testing process was to validate that the tool fulfils the requirements set in Section 3.2.

4.1 Implementation environment

IQVis is implemented as an Adobe Flash [56] application written in the Actionscript 3 language. The tool uses the open source Flex Software Development kit (SDK) [57] and open source visualization toolkit Flare [58], which is based on its predecessor, prefuse [43]. As open source Flex does not provide Flex builder as open source, open source FlashDevelop [59] is used as a code editor. In addition, an open source utility library, as3corelib [60], is used for decoding JSON messages in the VMonitor component.

As we wanted to have a lightweight and modular visualization tool for the analysis of the run-time behaviour of quality variability, we chose Adobe Flash [56] as the target programming environment. Other possible environments were Java, Java Eclipse plug-in, Matlab, etc. Adobe Flash Player is a cross-platform browser plug-in and the IQVis tool can therefore be used on various platforms that have Adobe Flash Player. Adobe Flash Player is also a more lightweight environment compared with, for example, Matlab or Eclipse, which require installation and much more disk space. Flash applications are compiled into the SWF file format (SWF is not an acronym, although it is associated with Shockwave Flash) [61] and are run on an ActionScript Virtual Machine 2 [62] (AVM) inside Flash Player.

4.1.1 Constraints

The following section presents the constraints that come from the chosen implementation environment, Adobe Flash [56]:

- Flash player 10 (FP10) is required to save files to the local computer. The save process can of course be modified so that the monitored information is sent to a remote server, which then saves it to a file.
- The AVM in the Flash Player currently does not support multithreaded applications. This means that we cannot assign different threads for the components of the IQVis tool, and this will have an effect on the performance.

A workaround to obtain threading support into the IQVis tool would be to separate the VMonitors and Visualization Views into different Flash application files and play them in separate AVMs (i.e., Flash players). The VMonitor and Visualization View components are actually separate SWF files so they can be installed at run-time. Although this would guarantee different processes for them, the connection between the components could slow the application down. The communication could be handled via a socket connection or Flash LocalConnection, which uses shared memory to communicate between different Flash applications. The only downside to the usage of the LocalConnection is that all the Flash applications would have to be running at the same time, e.g., in a browser window or different standalone Flash players. This would require the user to open the different applications manually. [56]

4.1.2 Graphics libraries

Graphics are an essential part of information visualizations. Some of the visualizations used in IQVis are based on the Flare visualization toolkit [58], which incorporates many common visualization techniques for Flash such as graph drawing, scatter plots and various animation functionalities. The IQVis tool uses a subset of these visualizations and animations. The rest of the visualizations are implemented using the Adobe Flash [56] drawing API.

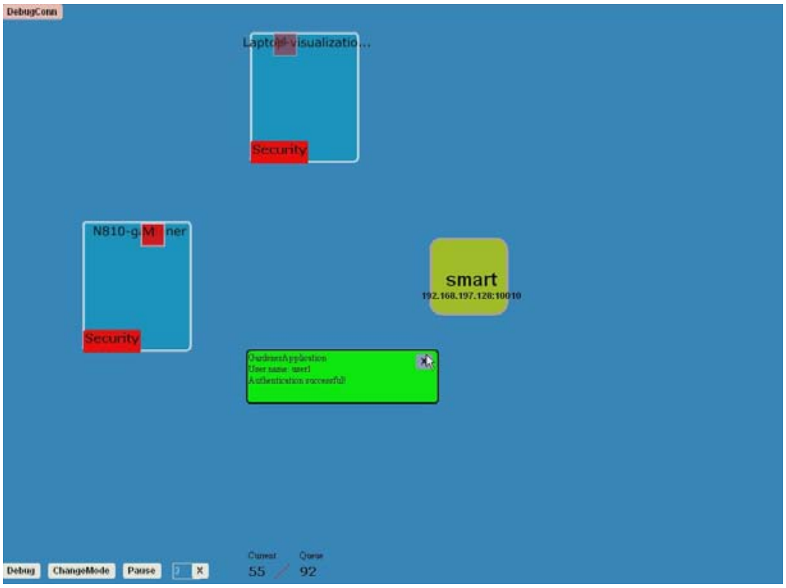
4.2 Implemented components

The Visualization Platform defined in Section 3.3.2 is implemented to use only one VMonitor component at a time, and only one model of the data is therefore provided at a time. This is done because the chosen implementation environment (Adobe Flash) does not support multithreaded applications. The model requires a large amount of memory and performance as it saves all the events that have occurred and provides functionality for querying and managing the data.

The components that are explained in Section 4.2.2 to Section 4.2.7 follow the design explained in Section 3.3, so a picture of the architecture has been left out of this chapter.

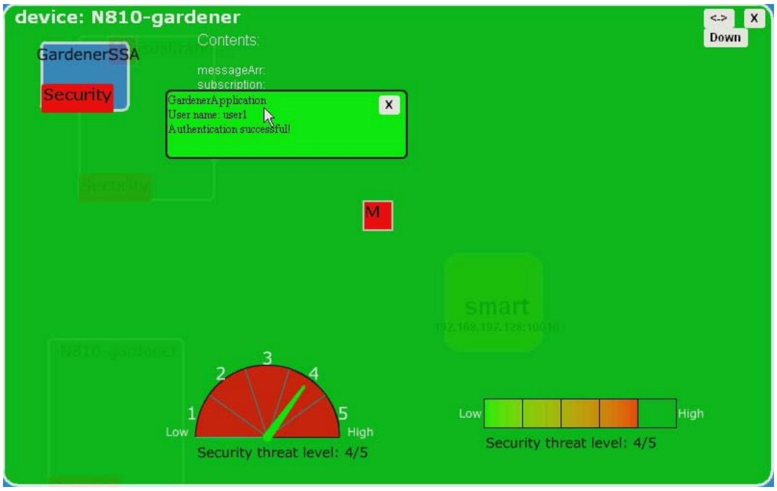
4.2.1 Initial version of the tool

The initial version of the IQVis tool was used to visualize the security adaptations in the semantic information interoperability demonstration described in [63]. The initial version (also implemented by the author) only showed the smart environment devices and software, the security techniques used and the current security threat levels of specific software. Screenshots of the initial IQVis tool are shown in in Figure 22, which (a) shows the topology view of the smart environment and (b) shows a detailed view of an application and its properties, e.g., meters showing the security threat level.



GardenerApplication
User name: user1
Authentication successful!

a) Smart environment devices.



GardenerApplication
User name: user1
Authentication successful!

b) Security threat level of a specific entity.

Figure 22. Initial version of the IQVis tool.

4.2.2 VMonitor

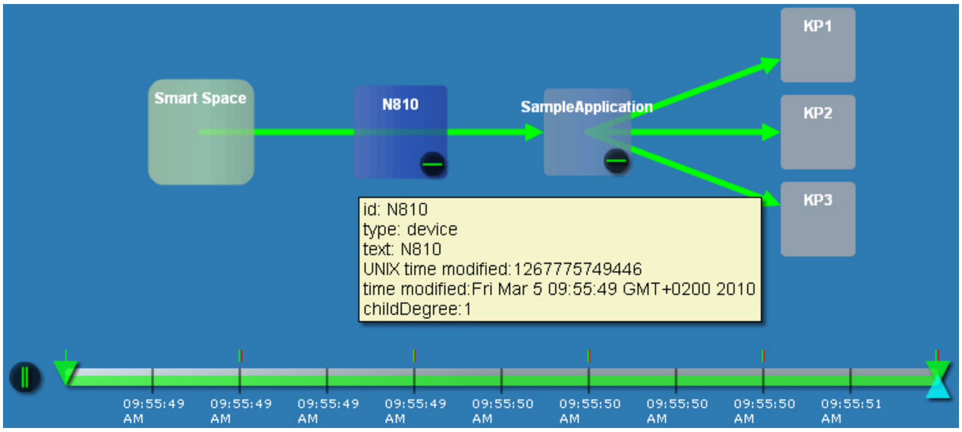
For the second version, the implemented VMonitor sets the model’s duration parameter to 10 milliseconds, i.e., the model should only store events to a certain element that has a period of at least 10 milliseconds between them.

The EMonitor on the environment side sends JSON messages to the VMonitor, which converts them to the form explained in Section 3.3.2. Figure 23 shows an example of the kinds of messages the VMonitor receives from the EMonitor. In Figure 23, a device called N810 joins the SS and reports that it contains a SampleApplication SSA that comprises three Knowledge Processors (SSA, KPs, explained in more detail in the case study, Section 4.4.1). The aforementioned names (N810, SampleApplication, KP1, KP2 and KP3) are emphasized to clarify the format. The implemented VMonitor creates the connections (edges) between the entities by adding 5 milliseconds to the creation time of the connected entity if the edges are not explained in the JSON message, as they are not in Figure 23.

```
#1:{"add": {"text":"Smart Space", "node_id": "SS1", "type": "ss", "time":
"1267775748946"}}
#2:{"add":{"node_id":"N810", "text":"N810","type":"device",
"connectedTo":"SS1", "time":"1267775749446"}}
#3:{"add":{"node_id":"SampleApplication", "text":"SampleApplication", "sub-
NodeOf":"N810","type":"ssa", "time":"1267775749946"}}
#4:{"add":{"node_id":"KP1","text":"KP1","subNodeOf":"SampleApplication",
"type":"kp", "time":"1267775750446"}}
#5:{"add":{"node_id":"KP2","text":"KP2","subNodeOf":"SampleApplication",
"type":"kp", "time":"1267775750946"}}
#6:{"add":{"node_id":"KP3","text":"KP3","subNodeOf":"SampleApplication",
"type":"kp", "time":"1267775751446"}}
```

Figure 23. Short example of the JSON format produced by the EMonitor.

Figure 24 presents the way the IQVis tool visualizes the commands shown in Figure 23. As seen, the IQVis tool shows a central node, Smart Space and the connected device, N810. The N810 device has the SampleApplication and contains KP1, KP2, and KP3, as seen in the JSON message in Figure 23. In Figure 24, the mouse cursor is over the N810 device and some of the details on it are shown in a tooltip. Notice how the UNIX time shown in the tooltip in Figure 24 is the same as the highlighted time for the addition of the N810 in Figure 23.



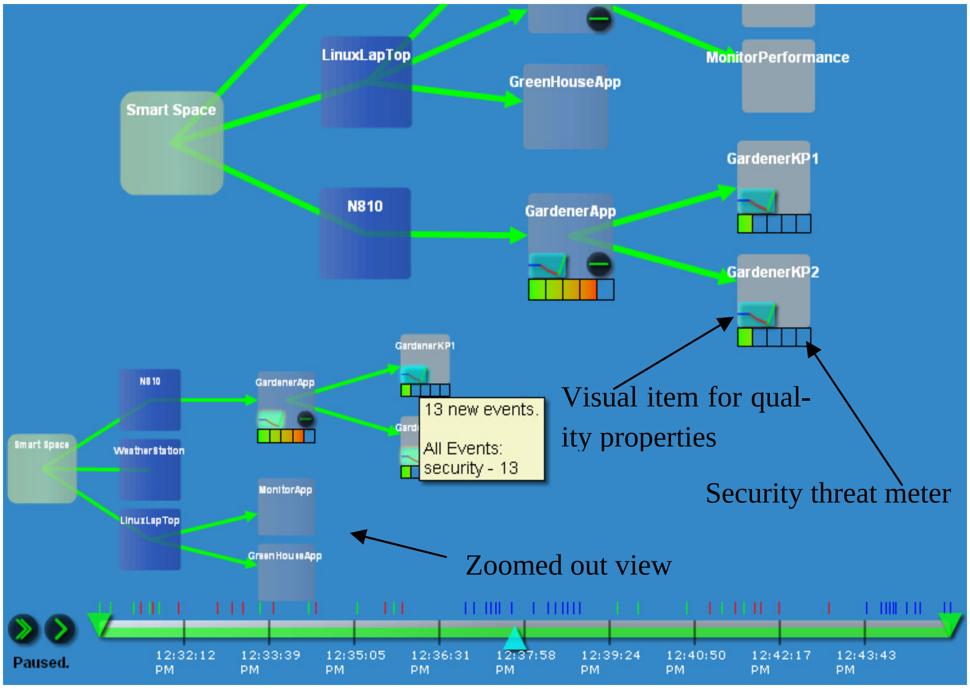
id: N810
type: device
text: N810
UNIX time modified:1267775749446
time modified:Fri Mar 5 09:55:49 GMT+0200 2010
childDegree:1

Figure 24. Visual representation of the JSON message in Figure 23.

4.2.3 Visualization View

Figure 25 presents the actual implementation of the Smart Space View visualizing simulated entities with the graph visualization technique in which two views are shown with different zoom levels. A Smart Space is visualized on the left-hand side, and connections to the devices (N810, WeatherStation, LinuxLaptop) are undirected edges, illustrating that it is a two-way connection. The devices include applications, which may comprise different KPs, e.g., the GardenerApp has two KPs as shown in Figure 25. The connection between the devices and their application is always directed from the device to the application and from the application to the KP. This is because an SSA may be constructed from multiple KPs.

The idea of the Smart Space View is to allow the user to see the topology of the smart environment that he/she is analysing. If there are security threat levels monitoring events for the entities shown in the view, a minimized security threat level meter is visible to the user (e.g., in ‘GardenerApp’ in Figure 25). If the entity has quality attribute measurements, this is visualized showing a small diagram in the bottom left corner of the rectangle representing different entities, as seen in Figure 25. When the user moves the mouse cursor over the quality icon, a tooltip is shown.



13 new events.

All Events:
security - 13

Figure 25. Implemented Smart Space view.

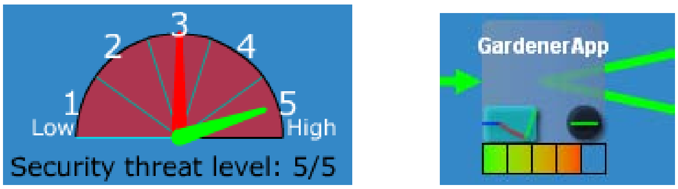
This Smart Space View can be used to view the topology of the smart environment and select certain entities from which to analyse quality variability. The Smart Space view also allows the layout of the graph representing the topology of the smart environment to be changed. The layout shown in Figure 25 is called NodeLinkTreeLayout.

The Smart Space View also allows the user to zoom in or out on the graph and to pan the graph, i.e., move it on the screen. Zooming is done by scrolling the mouse wheel or pressing the Ctrl key and selecting the background by pressing the left mouse button and moving the mouse up or down. Panning is done by pressing the left mouse button and moving the mouse cursor in the preferred direction.

If an entity has child nodes, these can be expanded or unexpanded to show or hide more details. The graph visualization technique is implemented using the Flare visualization toolkit [58]. A custom renderer (a class that draws the entity according to its properties) was implemented to visualize the node instances for different types of entities found in a Smart Space [5].

4.2.4 Meters

Meters were implemented to be able to visualize the variability of one particular quality attribute. In the context of this thesis, that quality attribute was security. The implementation was based on the design presented in Section 3.4.4. The security meter was implemented as both a gauge and a bar meter (currently only the gauge meter is in use). Figure 26 (a) presents the implemented security meter, which shows that the security threat level is five (5) and is over the threshold value of three (3). Figure 26 (b) presents a minimized bar-type meter used to show the security threat level in an overview of all the devices (used in the Smart Space view that is explained in Section 4.2.3).



a) b)

Figure 26. Different implementations of meters.

The security meter is used to show visually how the security threat level of the device changes due to the dynamic behaviour (i.e., changes happening) in a Smart Space. When the value of the meter changes, the meter pointer or bar is animated to the new value so that the user can spot the changes more easily.

4.2.5 Time slider

The timeline control was implemented according to the design presented in Section 3.4.5. Figure 27 presents the current time slider of IQVis. The timeline control is used for the Visualization Views (higher level) and the visualization techniques (lower level). Figure 27 shows a selected time range (delimited with two downwards triangle thumbs) and the currently selected point in time with a cyan triangle thumb.

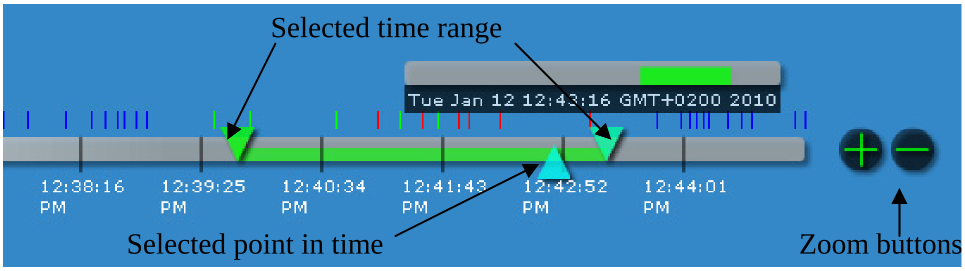


Figure 27. Time slider used to select a point in time.

As this application supports both online and offline visualization of the monitored values, the time slider component has to support both features. There are two major use cases for the time slider component: 1) when the user wants to see the current situation in the environment, the main time slider is not used other than for zooming in and out of the range end, and 2) when the user views the state of the environment in the past.

As the user views past actions in the environment, he/she is not able to view the current run-time situation. When the user decides to look at past actions by moving the thumb backwards on the x-axis or if he/she decides to pause the time in order to analyse a device in more detail, the time slider component and the whole view stop updating values. The main time slider component shows the actions that occur at run-time with visual cues (as described earlier), but otherwise it moves the currently viewed point in time (the selection thumb) backwards in time.

The different action buttons in the pause situation are illustrated in Figure 28. The user can take three different approaches when returning to the current situation from earlier points in time. First, the user can simply press the “unpause” button (the second button from the left in Figure 28), which causes all the occurred events in the environment to be played out in sequence until the current run-time situation is met. Second, the user can press the “jump forward” button (the first button from the left in Figure 28) to jump to the current situation in the environment, which causes the Visualization View to load the current values and show them. Third, the user can drag the selection thumb to the end of the slider component, which causes all of the occurred events to be updated at once as in the second approach.

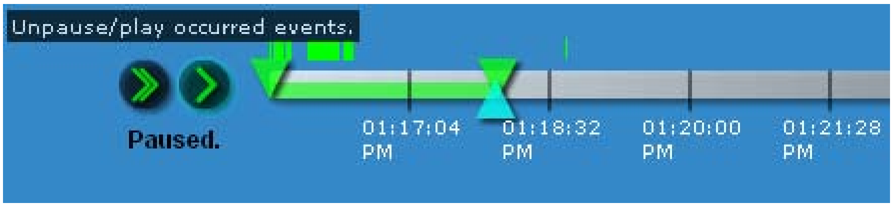


Figure 28. Time slider in paused mode.

Figure 28 also illustrates how the time slider functions in the paused mode, i.e., new events, are shown to the right of the green selection area (delimited by downward thumbs). This means that the user now views the time range selected on the green line but that there have been new occurrences in the real-time situation. If the time slider is zoomed in to a certain time range, the new events are not shown on the actual timeline. Instead, a textual notification is shown under the zoom in and out buttons in Figure 27 to notify the user.

Figure 29 shows the time slider used to select a range in time. When the user moves the selection thumbs (downward triangles), he/she is shown a visual tooltip of the currently selected time range. This tooltip visualizes the duration of the currently selected time range and where it is compared with the whole time range and it therefore helps the user to gain an understanding of which section of the time range he/she is viewing.

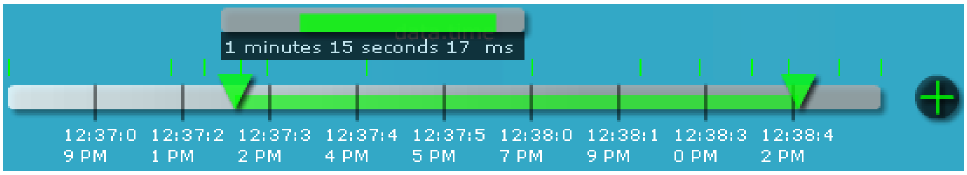


Figure 29. Time range selection.

The mode shown in Figure 29 is used for, for example, timeline visualization to select the range of time that is to be shown in the visualization. This feature is also shown in the next section, which describes the implemented timeline visualization.

4.2.6 Timeline visualization

The time slider component is used with the timeline visualization component so that the user can view different sections of the analysed quality. The timeline visualization is shown to the user when the user clicks the visual item representing quality properties, which was shown in Figure 25. Figure 30 shows how the user first limits the range of time that is shown and then zooms in on the timeline control to view the events in more detail. Notice that the user is shown a green vertical line (current selection limit in Figure 30) on the timeline visualization when he/she limits the time range shown in the visualization. The user can select the time range more quickly when he/she can see the vertical line moving across the values shown in the timeline visualization. When the user moves the thumb to a new location and releases the mouse button, the timeline visualization zooms in to the chosen time range with the help of animation.

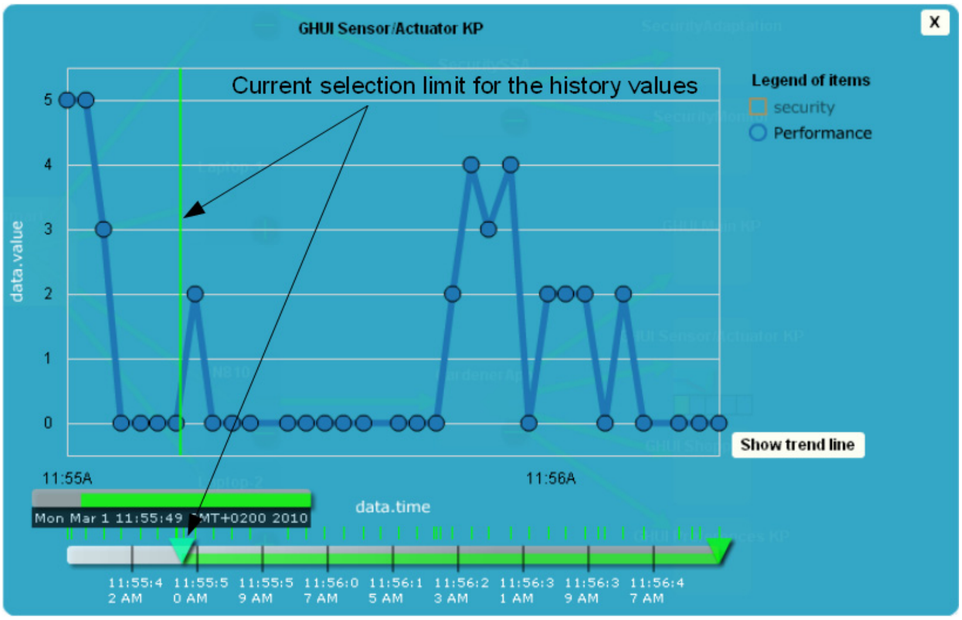


Figure 30. Timeline control and timeline visualization.

The timeline visualization also allows the user to select the time range (time span) he/she wants and to reconstruct all the events that occurred within this specific range. This enables the user to view how the events that have occurred have affected other qualities or why they happened. This feature is presented in Figure 31.

As the scatter plot visualization technique is basically the same as the timeline visualization (only missing the edges connecting the entities), a separate image or explanation of it is not shown.

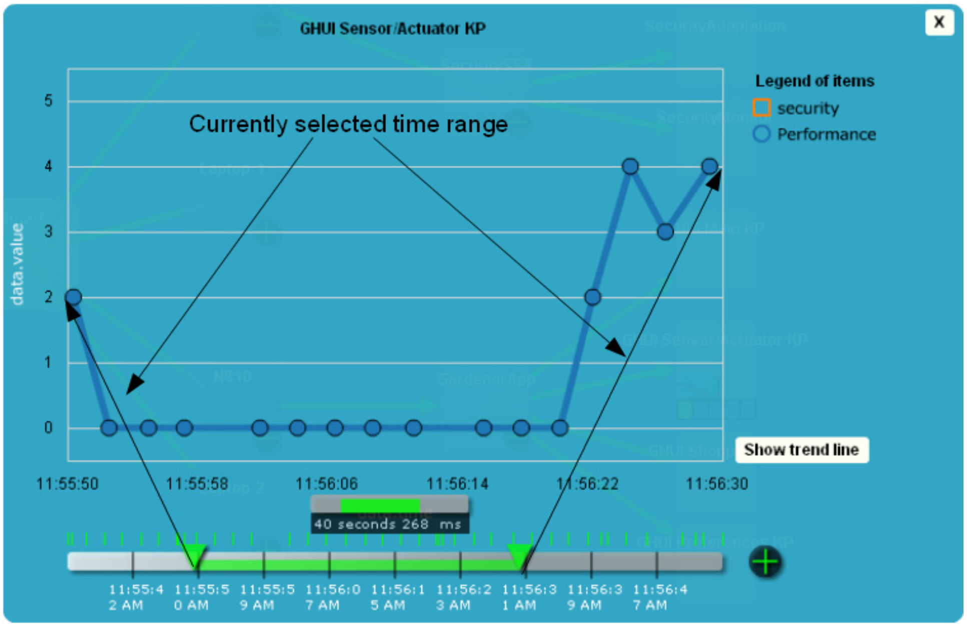


Figure 31. Timeline control and timeline visualization.

4.2.7 Treemap visualization

The treemap visualization is implemented so that the user can see many quality attributes and their changes at the same time. The colour of the treemap squares is mapped to the quality attribute value. The colours go from green to yellow to red, with green being good and red bad. Figure 32 illustrates a treemap visualization technique in which 29 different quality attributes are visualized in a treemap (simulated).

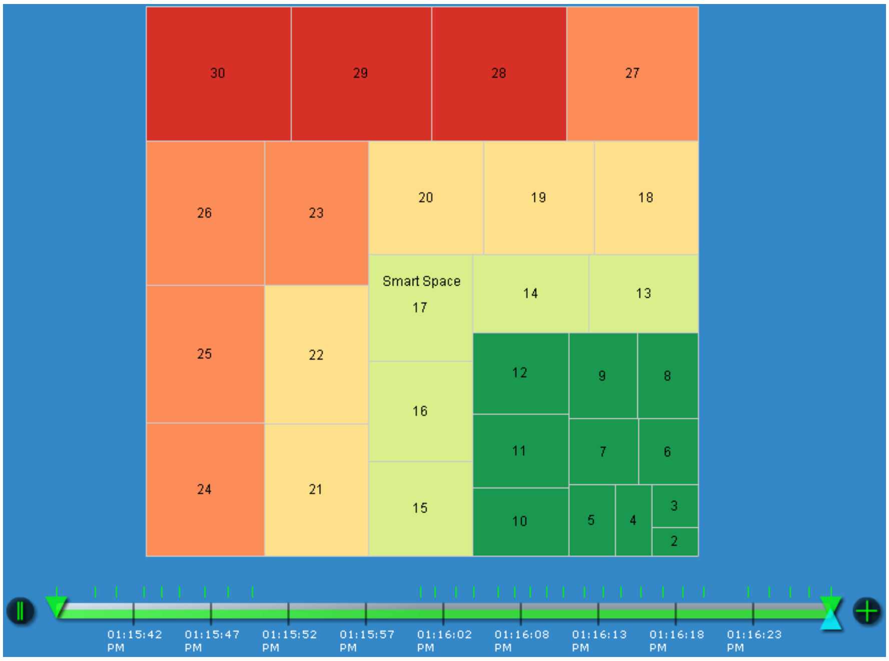


Figure 32. The treemap visualization technique in IQVis.

The idea is for the user to be able to spot quickly the qualities that are currently good or bad according to predefined limits. In Figure 32, only different quality attributes are shown in the boxes, but the visualization could be modified in the future to show the different devices and their combined quality values. This technique can be used to visualize all the quality attributes in the smart environment to monitor or analyse them all at once.

4.3 Testing

The testing process verifies that the implemented environment conforms to the defined requirements and contains the required functionality for the tool. The requirements of the IQVis tool have been defined in Section 3.2, which includes both functional requirements (Table 3) and quality requirements (Table 4). The framework for the testing process is constructed from these requirements.

The testing process is divided into three parts: unit testing, component testing and integration testing. First, the unit testing covers testing of the components' internal methods. Second, the component testing covers testing of the component as a whole (e.g., Visualization View, which comprises different visualization technique components). Third, the integration testing tests that the components are able to co-operate and exchange information. Testing is performed with a black-box approach, i.e., observing the results the components returned when sending series of inputs to the component. Visual components were tested by sending data to the visualization component and observing whether or not the produced visualization was correct. AsUnit [64] was used in unit testing to provide a low-level automated testing system for testing the different methods of the components. AsUnit was chosen because it provides the necessary functionality for testing methods and events in Actionscript. Another unit test tool is the FlexUnit, but as all of the implementation is done with the Actionscript (Flex components are also accessed with Actionscript) language, we chose AsUnit.

The first component that was implemented and tested was the Visualization Platform and the data model, which are responsible for data access and the functionality of the whole IQVis tool. The tests covered all the explained interfaces in Section 3.3.2, e.g., adding and modifying, querying and deleting data.

The second component that was implemented and tested was the time slider component explained in Section 3.4.5. The time slider component is a visualization component and was therefore tested by sending data to it and observing the result of the visualization.

The ActionScheduler, as described in Section 3.3.5, was then implemented and tested. The Visualization View component for the smart environment was then implemented and several visualization techniques were added to it from the Flare [58] visualization toolkit. The visualization techniques added from Flare [58] included a timeline, scatter plot, treemap and graph visualization techniques, as explained in Sections 3.4.1 and 3.4.3. Some layout algorithms were also used and tested for graph visualization.

Before integration testing, a debug VMonitor was created to serve as a data source for the IQVis tool. The debug VMonitor was implemented as described in Section 3.3.3, except that no connection was made to the actual environment. The debug monitor was tested to ensure it sent correct messages to the model, as described in Section 3.3.2. The debug VMonitor was then used to test the whole IQVis tool after all the components had been implemented and tested separately. After finishing the integration tests successfully, an actual VMonitor for the case study was created based on the working debug VMonitor.

The different tests showed that the implementation of the IQVis tool conformed to all of the requirements set in Section 3.2 as described in the design in

Section 3.3. The case study in the next section will illustrate how the implementation of the IQVis tool worked in an actual smart environment.

4.4 Case study

This chapter describes the case study in which the IQVis tool was used to visualize quality attributes and their variability in a smart environment. The purpose of this case study is to validate all the functionalities of the IQVis tool in a real environment, in addition to the simulation tests performed at the laboratory testing phase.

In this case study we will show the structure of the SS and visualize the quality variability of an application at run-time. We will only use visualization techniques that are appropriate to this case study, as this case study only includes a few devices and applications.

4.4.1 SOFIA Smart Space

The smart environment used in this case study is called Smart Objects for Intelligent Applications (SOFIA) [65]. The Semantic Information Broker (SIB) is the main element in SOFIA [65]. Software agents connect to the Smart Space (SS) through the SIB and request services from it, i.e., the SIB is a physical entity that makes up one or more SSs. The SOFIA InterOperability Platform (IOP) aims to make the information "embedded" in the physical world available to smart objects, services and applications in the smart environments [66]. IOP enhanced with context awareness is presented in [66]. The following describes the most important definitions of a Sofia Smart Space (SS) [65]:

- The SIB is an information world entity for storing, sharing and governing the information of one Smart Space as the RDF (Resource Description Framework) triples. RDF triplets present the information in subject-predicate-object expressions.
- A knowledge processor (KP) is an information world entity that processes information and contributes to and/or consumes information content from the SIB according to ontology relevant to its defined functionality. A simple consumer/producer application can be formed from one KP, but one or more KPs are needed to allow useful sharing of contents. Information is shared by complying with a defined ontology.

- A Smart Space Application (SSA) is an application that can comprise different KPs.
- The Smart Space Access Protocol (SSAP) is used by KPs when accessing the SIB. The SSAP defines the basic messages for joining, leaving, inserting, removing, updating, querying and subscribing.

The reason the IQVis tool is validated in SOFIA is that quality variability is present in the environment. The security features of the KPs are adapted at run-time according to the situation in the smart environment. The IQVis tool will visualize the structure of the SS by showing the relations between different devices, SSAs and KPs. The actual quality variability takes place in the SSA or the KPs and is visualized in these entities.

4.4.2 Testing environment

The actual IQVis tool was tested on a Dell Latitude D830 laptop in a web browser (Flash player installed as a plug-in) and in a standalone debug Flash Player. The hardware configuration of the smart environment used in the case study includes a Dell Latitude D830 laptop running an application (legacy adapter) that provides weather station information from the Willab Weather station [67] to the smart environment and runs the Security Adaptation service N810 Internet Tablet, which is used to run the gardener's application. A laptop and a web camera are used to provide the number of customers currently in the smart environment, which is used as context information for the security adaptations. A laptop is used to run the SIB that provides the Smart Space which the aforementioned devices join and communicate with.

4.4.3 Validation scenario overview

To illustrate the IQVis tool in action, it was tested in the context of Smart Spaces. The testing scenario tests a modified version of the whole scenario explained in [63]. A sample SSA in the validation scenario is a gardener application that runs on a Nokia N810 Internet Tablet.

The mapping of the monitored qualities to the levels used in the IQVis tool is done so that low values are good and high values are bad. The mapping is done as explained because, in the case study, the monitored qualities describe the security threat level and the CPU utilization level. In this context, the lower the value the better it is. This is because the use of security algorithms requires more CPU processing power and security adaptations therefore affect the performance level. The quality variability that takes place in the applications is therefore visualized by showing the changes in CPU utilization and the security threat level. When the security quality changes in the gardener's application, i.e., a security mechanism is enabled or disabled, the security threat level of that application is monitored.

The following explanations are provided to clarify the validation scenario and are not included within the scope of this implementation. The SS has a Security Adaptation service that monitors the status of the SS and the devices currently in the SS. The Security Adaptation service adapts the security of the KPs, which have reported a desired security level. An EMonitor component is installed on the SS and monitors different aspects of the devices and software that join, interact and leave the SS. The EMonitor provides quality attribute information to the IQVis tool. It provides security risks for different KPs and information on the dynamics of the SS, i.e., which devices are currently in the SS, which SSAs are in the devices and which KPs make the SSAs. For the purpose of this case study, a performance monitor is also used to monitor the used CPU time of a KP. Performance monitoring is used to provide coarse-grained performance quality information to the IQVis tool so that the trade-off between qualities can be made.

The testing of the tool is performed by connecting the VMonitor (called SSMonitor) into an SS that has no devices joined to it. The validation scenario is described as follows:

Step1: The IQVis tool is started and the VMonitor component (SSMonitor) is activated in order to connect to the EMonitor component in the target environment. Next, the SSMonitor starts to update the model. It starts with the addition of a node representing the monitored Smart Space to the currently used view.

Step2: The N810 Internet Tablet joins to the SS. The N810 and the corresponding SSAs and KPs are shown in the IQVis tool. One of the KPs in the gardener's application (GardenerApp) reports that it requires its security threat level to remain below a certain limit and reports its performance usage. A weather station KP also joins the SS and provides weather information.

Step3: More people join the SS and the Security Adaptation KP adapts the security of the KP in the GardenerApp. The IQVis tool shows the effects of these adaptations by visualizing the change in the meter showing the security risk level and by adding an icon representing the quality attributes of the gardener's application. These qualities are shown at the bottom of the rectangle representing the gardener's application. As people arrive at the SS, the Security Adaptation KP informs the KPs that have set a desired security level that they have to adapt their behaviour to correspond to a certain security level.

Step4: The gardener leaves the SS and this leave is visualized by removing the N810 node from the view. The timeline control is tested by selecting a time before the N810 device leaves to see whether the device is shown correctly.

Step5: Monitoring of the environment stops and the data from this session are saved with the IQVis tool. After this, another IQVis tool is started and the saved data are loaded into the IQVis tool and compared with the IQVis tool that is already open to see whether the visualization is the same.

4.4.4 Smart environment structure

This section covers steps 1, 2, 3 and 4 of the case study. After the SIB is initialized, the IQVis tool is started and the SSMonitor activated. The IQVis tool shows the created SS named "smart". All the devices and software explained in Section 4.4.2 are started and connected to the SS. Figure 33 shows how the IQVis tool shows these devices and software after all of them are joined to the SS. The static figure does not show that the IQVis tool adds the entities with the help of animation, i.e., first it adds an entity and then it adds the edge that connects the entity to another entity. The mouse cursor is over the timeline in Figure 33 and the timeline details are shown in a tooltip. Figure 33 also visualizes the security threat level of one of the KPs that forms the GardenerApp by showing the security threat level meter at the bottom of the rectangle representing the KP (GHUI Sensor/Actuator KP in Figure 33). An icon is also visible in the KP's rectangle indicating that it contains quality attributes.

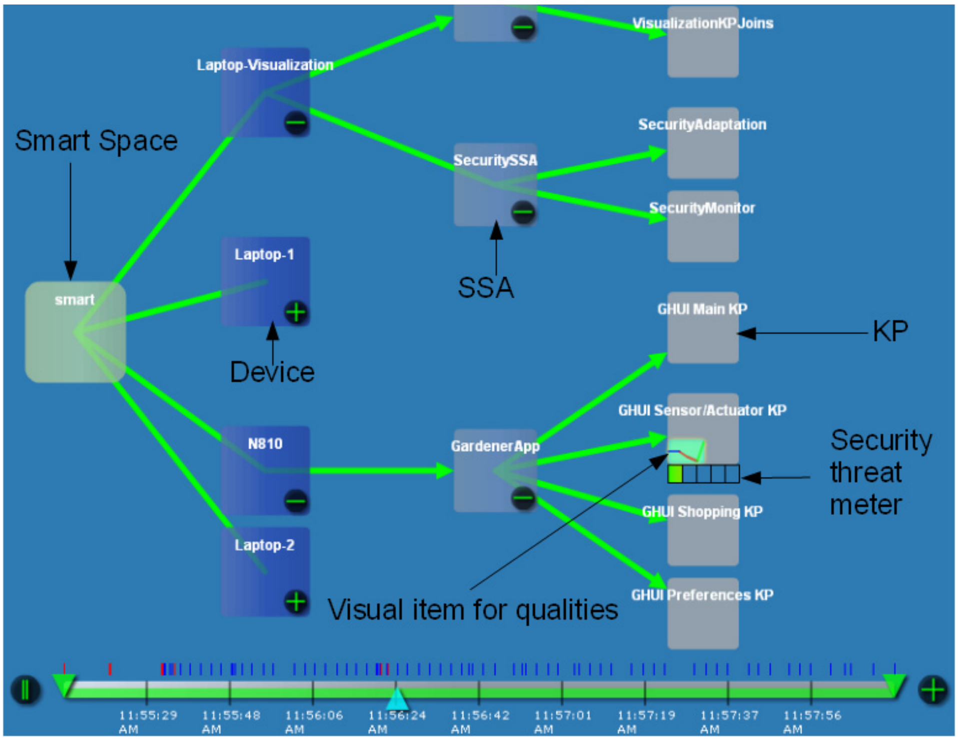


Figure 33. Case study smart environment structure.

The N810 device leave is showed correctly and the time slider can be used to go back to the point in time at which the N810 was in the environment. After the time slider selection thumb is moved to the point before the removal of the N810, the IQVis tool loads the situation at that time and the N810 is again visible, as in Figure 33.

4.4.5 Quality attribute analysis

The timeline visualization technique is used to analyse the quality attributes in this case study. After the devices have joined the SS and the IQVis tool is in the situation shown in Figure 33, the user clicks on the icon representing the quality attributes of the GHUI Sensor/Actuator KP. After this, the user is shown a window with a timeline visualization showing all the quality attributes of the application throughout the monitoring time. Figure 34 shows the timeline visualization after the user has chosen the performance from the legend visible on the right-hand side of the window and used the time slider component to limit the range. The time slider component shows the whole time range in which qualities have been monitored. In Figure 34, the user has pressed the “Show trend line” button to see the trend of the performance quality.

As shown, the trend of the performance is upwards, i.e., more CPU is utilized over time. Figure 34 also clearly demonstrates how the monitored values are 0 for some time in the beginning and then jump into the range 1-4. This shows that the GHUI Sensor/Actuator KP application used the CPU very little at the beginning of the monitoring, but started to use it more after some time. At the end of the visible time range, the CPU utilization again dropped to 0, which tells that the security features have been disabled.

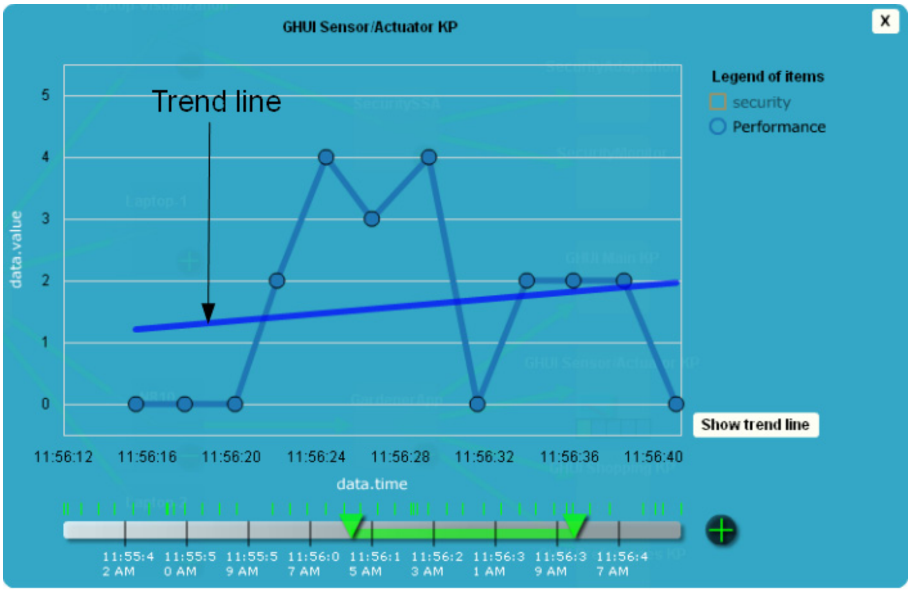


Figure 34. Performance values and their trends.

In Figure 35, the user has selected the security quality from the legend on the right-hand side of the timeline visualization. The time range is the same as in Figure 34. The trend of the security is downwards, i.e., the security threat level is decreasing. Figure 35 shows how the security threat level rises to 4 and shortly afterwards drops to 2. This happens because the Security Adaptation KP tells the KP in question to activate security features. After a while, the security threat level drops to the original 1, which shows that the outside security risk has fallen.

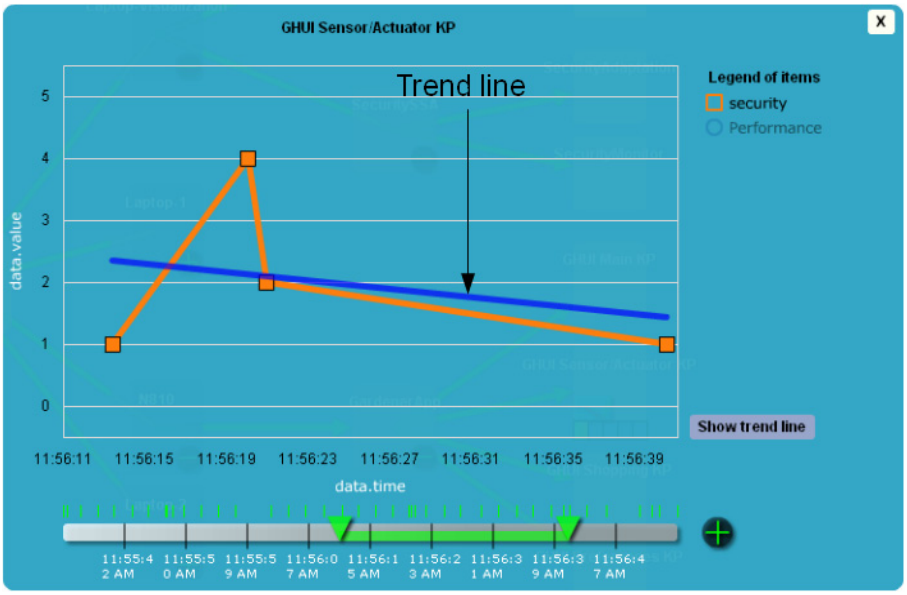


Figure 35. Security values and their trends.

4.4.6 Quality trade-off analysis

This section covers step 3 of the case study. The trade-off between monitored performance and security quality values is viewed by selecting both in the timeline visualization. Figure 36 shows the timeline visualization with both performance and security values visible on a larger timeline than shown in Figure 34 and Figure 35. As shown, the security value drops shortly after the performance quality rises. This can be explained by the fact that there is a small delay between the change in security measures and the time when the Security Adaptation reports that the security threat level has dropped. The trade-off between security and performance is quite clear from Figure 36: more performance is needed to maintain the security threat level below a certain limit. Figure 36 clearly shows that the security threat level has fallen because of the security adaptation, although the change in the actual security algorithm is not visible.

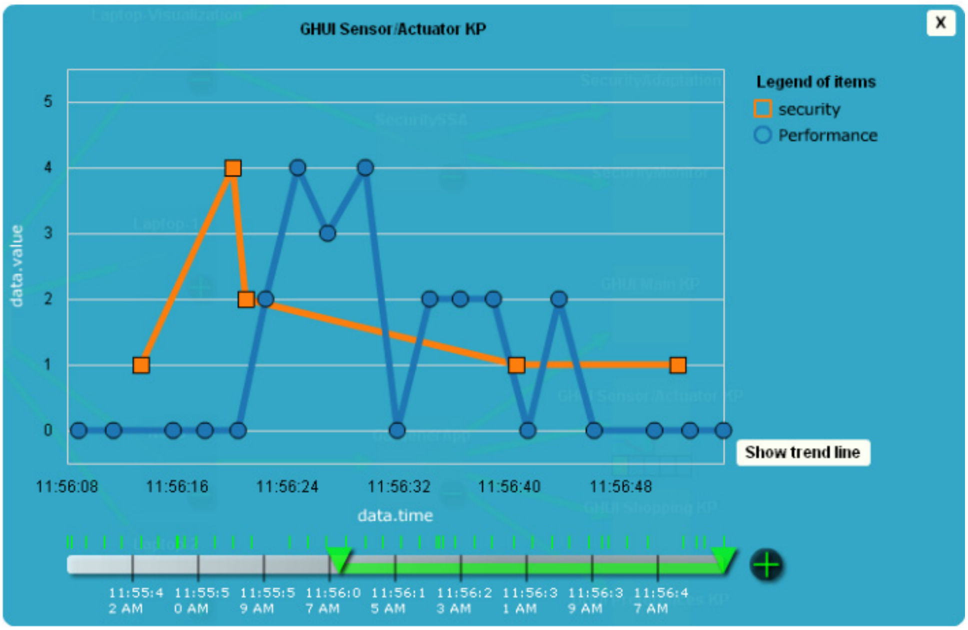


Figure 36. Quality trade-off.

4.4.7 Cross-platform testing

This section covers step 5 of the case study. As IQVis is implemented with the Adobe Flash platform, we wanted to check part of the validation case study with another platform, namely, the Nokia N810 Internet Tablet and the Nokia N900 mobile device. The N810 has Adobe Flash Player (FP) 9,0,48,0 support, and the N900 used in our test had FP 9,0,246,0 support. Due to the Flash Player version we cannot use these devices to save the monitoring data (FP10 supports that), but we can load some previously monitored data with them.

The use of these mobile devices in this case study poses many difficulties. First, the hardware capabilities and physical screen size are much smaller: 4.13” WVGA display on N810 [68] and 3.5” WVGA display on N900 [69], both with 800 x 480 pixel resolution. The physical screen size is a bigger limitation than the pixel resolution. The physical screen size of these devices makes it harder to visualize many entities at once. Second, both of the devices have a touch-screen, and IQVis is not designed for this but can of course be used with it, for example, the IQVis tool uses many tooltips, which are shown when the mouse cursor is over an entity. Fortunately, the N900 browser supports showing of the mouse cursor and the tooltips are indeed visible in the N900 but not in the N810.

We chose to carry out the last step of the case study, i.e., load the monitored data and play the monitored events as they occurred. The test includes animating the devices, SSAs and KPs to the graph in sequence. In order to see the performance differences, we added a simple monitor that records the time it takes to load the data and convert it into a format the IQVis tool supports. The loaded data is a JSON formatted version of the data stored in the model. The monitor also records the average frames per second (FPS) displayed in the IQVis tool, i.e., how many times the screen is updated per second, which has an effect on the usability of the tool.

To eliminate variance in the tests, we created a simple VMonitor for test purposes. The VMonitor loaded the previously saved test data, initiated the FPS recorder, set the data in the IQVis tool and started to play it sequentially. The tests were performed on the Dell Latitude D830 Laptop (Windows XP) with different standalone Flash Players and on some common web browsers. Next, the tests were run on both N810 and N900. All of the tests were run with a high-quality setting of the Flash Player. Each test was run three times and the average values were calculated from these runs. The FPS limit for the IQVis tool was set to 100 FPS, i.e., the application would not update the screen more than 100 times a second, if it was capable. Table 9 presents the results of the performance comparisons.

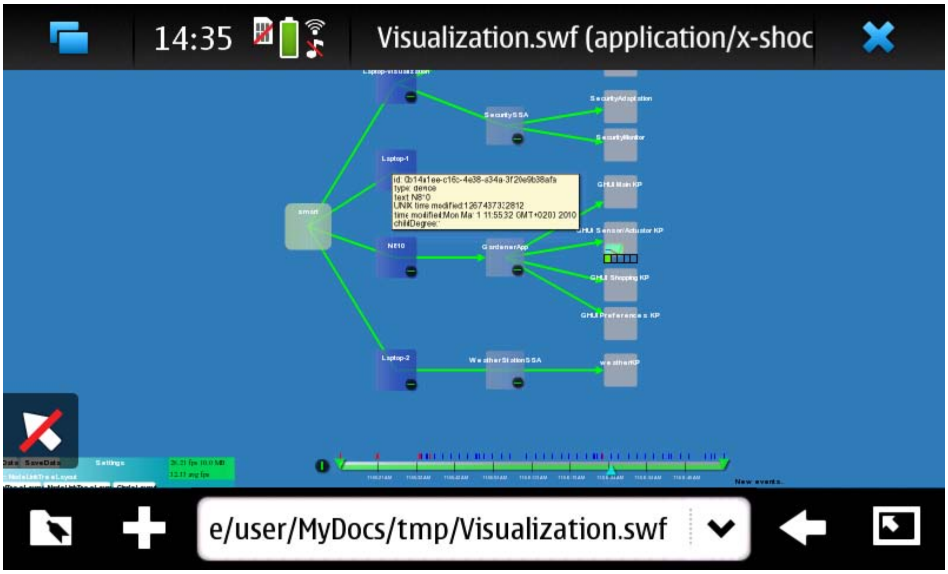
Table 9. Performance comparisons.

Device and FP version	**Time to load and convert monitored data, milliseconds**	**Visualize monitored data, average FPS**
Laptop FP 9,0,246,0 Standalone Player	110	97,27
Laptop FP 10,0,45,2 Standalone Player	136	98,81
Laptop FP 10,0,45,2 in Opera 10.10	125	98,41
Laptop FP 10,0,45,2 in Firefox 3.5.8	94	97,91
N900 FP 9,0,246,0	3721	22,83
N810 FP 9,0,48,0	9940	8,10

As can be seen from the results in Table 9, the laptop tests show that the IQVis tool runs almost at the set 100 FPS limit. The standalone FP 9 is negligibly slower than the standalone FP 10 on the laptop. Both of the web browser plug-in players also almost reach the 100 FPS limit. The loading times are very close to each other when comparing the laptop results. Due to the 100 FPS limit, the differences between the Flash Player versions are not very clear in the laptop tests. The reason different Flash Player versions were included was that the mobile devices only have support for FP9, and we wanted to test whether different versions have clear performance differences on the laptop.

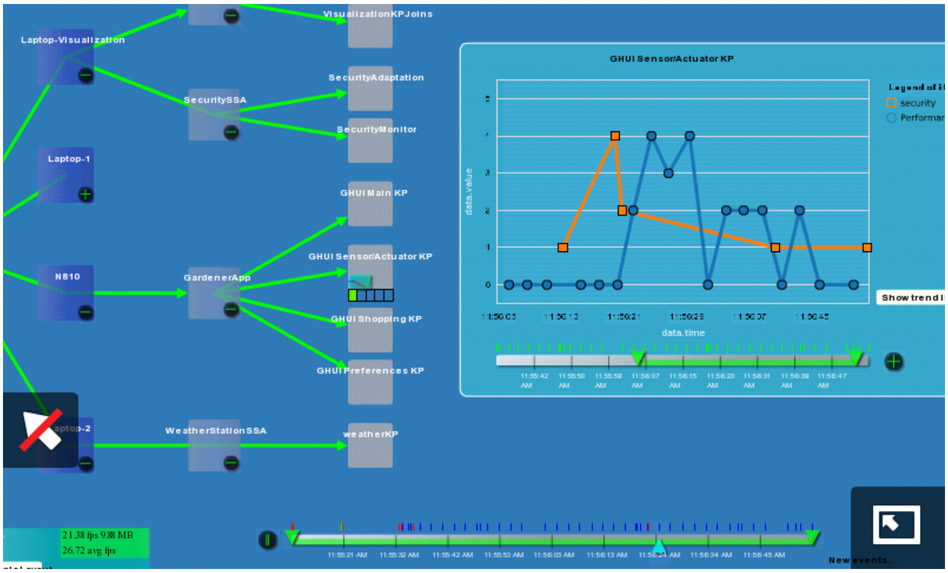
The N900 takes approximately 30 times longer to load and convert the loaded data into the format that the IQVis tool supports than the laptop. The N810 Internet Tablet is even slower – it takes on average about 80 times longer than the laptop. The same trend continues when comparing the FPS of the mobile devices. The performance of the N810 Internet Tablet is very poor compared with the N900 and laptop. The N900 has an almost three times larger FPS than the N810, and the laptop has, on average, a 12 times larger FPS than the N810. The tests show that the mobile devices are much slower at converting the saved data into the form that the IQVis tool supports. The user would most probably want to use the IQVis tool in a mobile device to check the run-time situation in the environment rather than analyse the previously monitored sessions. The N900 mobile device does seem to be able to run the current IQVis tool at a satisfactory level and because it also supports tooltips, it could be used to check the real-time situation of the environment. The physical screen size does lead to a few difficulties, which could be addressed by adding a new customized Visualization View for the mobile device.

Figure 37 (a) and (b) show how the IQVis tool looks on the N900 mobile device. The IQVis tool is in a situation in which a full performance test has been run. The structure of the Smart Space is visible on the node graph in Figure 37 (a) and the timeline visualization is opened in Figure 37 (b). As can be seen, the different entities are quite small but they are still useful, and the quality trade-off, in particular, can be seen quite clearly on the timeline visualization. The different entities shown on the node graph can of course be zoomed in for a more detailed view of the structure.



id: 0b14a1ee-c16c-4e38-a34a-3f20e9b38afa
type: device
text N810
UNIX time modified 1267437332812
time modified Mon Mar 1 11:55:32 GMT+0200 2010
childDegree:

a) Structure of the Smart Space.



b) Timeline visualization.

Figure 37. Screenshots from the N900.

In conclusion, the IQVis tool is usable with mobile devices, at least with the N900. The performance of the N810 Internet Tablet is a little too low to run the IQVis tool at a satisfactory level. Adobe is planning to release FP 10.1 for mobile devices, which should boost the performance of Flash applications on mobile devices [70].

4.4.8 Summary of case study results

This section shows which of the requirements set for the IQVis tool are demonstrated using the case study described earlier. Table 10 presents a summary of the results.

Table 10. Functional requirements and study case steps.

ID	Requirement	Study case step	Result
F1	Ability to visualize the variability of different quality attributes	3	Fulfilled
F2	Security variability	2, 3	Fulfilled
F3	Installation of monitor and view components	1, 2	Fulfilled
F4	Time slider	3, 4	Fulfilled
F5	History functions	3, 4	Fulfilled
F6	Data storing	5	Fulfilled
F7	Multiple views	3,5	Fulfilled
F8	Interaction	3	Fulfilled
F9	Animation	2,3	Fulfilled
F10	Ability to handle quality data over a long monitoring session	5	Fulfilled

The quality requirements, namely Extensibility, Interoperability and Usability are not included in the comparison presented in Table 10 because they are hard to measure. The Extensibility and Interoperability requirements are met, however, because we implemented a couple of different VMonitor and Visualization View components, which were used in the tests and study case. The tests used a simulation VMonitor and the case study used a VMonitor that received real values from the smart environment. The usability quality requirement is not examined in the case study, but the GUI of the tool is designed for ease of use.

In addition, the IQVis tool was tested in mobile devices, and it worked at a satisfactory level at least in the high-end mobile devices of today, namely the Nokia N900 mobile device.

The use of the IQVis tool to visualize the structure and quality properties of Smart Spaces is also discussed in [71].

5. Discussion

This chapter discusses the results of this thesis and provides a comparison of the architecture (presented in Chapter 3) and implementation (presented in Chapter 4) of the IQVis tool compared with other visualization tools reviewed in Chapter 2. The strengths and weaknesses of the implemented IQVis tool are discussed, and further improvements are suggested according to the discovered weaknesses.

5.1 Implementation of the IQVis tool

While carrying out the related research for the thesis, many useful visualization techniques and open source toolkits were discovered, e.g., Flare [58] for Flash and its predecessor, prefuse [43] for Java. Flare was used to provide the essential visualization techniques for analysing quality variability in addition to the visualization techniques implemented earlier. The implementation of the IQVis tool complied fully with the design presented in Section 3.3.

The Flare API was good. The toolkit was helpful as the instructions for using Flare were limited. Flare was mainly designed to be used in a similar way to ManyEyes [51]. It provided functionality for converting a data set from different formats (e.g., JSON) into visualizations. Flare provided many easy-to-implement interaction techniques in the visualizations, for example, by providing functionality for implementing tooltips. The animation functionality provided by the Flare toolkit was also used a great deal. It was also easy to implement new renderers for the different entities used in the IQVis tool graph, although some modifications were needed for visibility filtering. This was because the entities in IQVis graph contain child elements (such as the security threat meter), and the mouse interaction with these needed to be switched off and on when hiding or showing the entity in the appropriate layout algorithm of Flare. A disadvantage of Flare is that it is not implemented according to the MVC pattern, unlike its predecessor prefuse [43]. This implementation decision meant that only a single visualization can be built from a single data set in Flare. The architecture of the IQVis tool therefore uses multiple Flare visualizations to support the option to construct multiple visualizations from the same model.

Although Flare [51] provided many useful visualization techniques, the IQVis tool was designed to work as a visualization tool that could provide visualization at run-time. The architecture of the IQVis tool provided necessary functionality for implementing run-time visualizations. For the most part, Flare was easy to use when adding individual elements into visualizations, but some were harder to implement than others. The treemap visualization (Section 4.2.7), for example, requires the data to be added in a tree format, e.g., a node can contain multiple child nodes but can only have one parent node. The data model chosen for the IQVis tool was graph based, and was easy to use in most of the visualizations. The reason the graph-based data model was chosen was that it was found to be useful and often used in the related research. Only the treemap visualization technique caused minor problems, i.e., the tree had to be formed from the nodes and edges before the visualization could be built.

5.2 Results

The IQVis tool uses a lot of ready-made visualization from the Flare toolkit, namely, the scatter plot, timelines, treemaps, graphs, animation and layout algorithms for the graph visualization. The IQVis tool also includes some self-implemented visualization techniques, e.g., meters and time slider. The meters and time slider were validated in the case study along with the visualization techniques from Flare. The time slider component was found to be useful, especially for analysing the monitored values, the relationships between different values and quality variability.

Most of the visualization techniques that IQVis supports could have been performed with other tools, e.g., Matlab, but IQVis is mainly designed to be used along with an online monitor, e.g., visualizing the real-time situation. Use of the other tools to provide visualizations, e.g., the ones found in Section 2.4, would have required many modifications to them. Different data sets can be visualized quickly and easily with the help of Internet visualization services such as ManyEyes [51]. ManyEyes [51], for example, allows many different kinds of data sets to be used that can be visualized with different visualization techniques.

When comparing the data source requirements, the IQVis is of course not as flexible as ManyEyes because ManyEyes allows different kinds of data sets to be used and IQVis currently only supports the format explained in this thesis.

The IQVis tool is designed to visualize quality variability in smart environments, for example, when comparing it with ManyEyes [51], the IQVis tool is better suited to the task of visualizing quality variability because it 1) can visualize a run-time situation (as opposed to visualizing logs after run-time) and 2) store the history of the monitored qualities. These qualities enable the analyst to analyse the quality variability in more detail with the help of different interactive visualization techniques. The main aid for analysing the qualities afterwards is the time slider component, which allows the user to drill down to certain details in the data. The visualization techniques were chosen according to the properties that were found to be useful for visualization tools. Interaction was found to be the most profound property of visualization tools in Section 2.3.2, and this is understandable because the whole purpose of visualizations is to provide an insight into the data. The user is also allowed to choose which visualization technique he/she uses to visualize in different situations. All these choices were made to enhance the visualization experience provided to the user.

The architecture of the IQVis tool allows the user to change the two main components of the tool, i.e., the Visualization View and VMonitor. This makes the tool easier to deploy in a different environment in which the current visualization techniques or monitors are not applicable. New visualization techniques can also easily be added to the tool due to the architecture. A more advanced querying language was not included because it was not seen as necessary, and the methods defined are adequate for the needs of this tool.

The IQVis tool architecture was presented and the concept of using visualization in the context of V&V of dynamic systems was discussed in a paper that was accepted by the First International Workshop on Validation and Verification of Dynamic Software Systems (ViDaS’ 10) workshop [52]. A paper was also submitted to the Semantic Interoperability of Smart Space (SISS) workshop on the use of IQVis in Smart Spaces [71].

Although the IQVis tool was designed to be used with modern consumer computers and laptops, it was still usable with mobile devices, as was concluded in the case study. The use of mobile devices in visualization is discussed in [72]. It is stated that visualizations designed for desktop computers do not scale well to mobile devices. From the list of limitations stated in [72], the following have the greatest effect on the use of IQVis in mobile devices: 1) displays are limited due to their smaller size and lower resolution, 2) onboard hardware, e.g., CPU, memory and graphic hardware, is less powerful, 3) input techniques are different, e.g., point-and-tap with stylus, and 4) available tools tend to be limited. All of these still apply today, but of course the performance of mobile devices has improved. In addition, the N900 does have an 840 x 480 pixel resolution, which is enough for the IQVis tool.

5.3 Comparison with existing tools

The purpose of the related research was to find a visualization tool that could visualize different qualities at run-time. Unfortunately this kind of visualization tool was not found. It was concluded from the related research chapter that no visualization tool exists that would comply with all the properties presented in Table 2. The related research compared the chosen visualization tools according to different properties, but no tool managed to comply with all of them.

Streamsight [44]**,** explained in Section 2.4.3, used graphs to show the connections between different processing elements in the streaming application. Nodes were coloured according to different criteria, e.g., a performance counter. The IQVis tool's Smart Space view is very similar to that of the Streamsight tool's view. When comparing Streamsight and IQVis, Streamsight provides more information on the actual nodes of the graph as it shows different ports and the user can change the criteria for which the nodes are coloured.

There are many good visualization tools for visualizing program behaviour. One program behaviour visualization tool is GAMMATELLA [53], which uses treemaps to visualize program behaviour. The IQVis tool also supports the use of treemaps, which can be used to visualize simply the values and changes in the different qualities observed.

Compared with the tools reviewed in Section 2.4, IQVis provides better visualization support for the context of visualizing quality attribute variability. In Table 2 in Section 2.4.8 (the summary of the reviewed tools), for example, IQVis would be the most profound tool when compared with the presented columns. These columns were based on interaction techniques, visualization techniques, applicability, extensibility and real-time visualization. The IQVis tool conforms to all of these properties, as was found in the case study.

5.4 Future development

The purpose of IQVis was not to provide a large number of visualization techniques like toolkits, e.g., Flare. We chose a few essential visualization techniques that can visualize the quality variability in an appropriate way. Nevertheless, more visualization techniques could be included in the IQVis tool in the future to allow different kinds of analyses.

The IQVis tool could support saving of monitored data in different forms, e.g., the topology of the environment could be saved in a format such as an XML-based GraphML file format [73] that could be used in other tools as well. By allowing the user to save the results in different formats, he/she can distribute the results more easily to other tools.

The treemap visualization presented in Figure 32 (Section 4.2.7) only visualizes different quality attributes in the boxes, but the visualization could be modified to show the different devices and their combined quality attribute values in the future. In addition, the timeline visualization presented could be extended by showing the used security features in the background, so that the user could also see what is causing the changes in quality attributes.

When personal Smart Spaces become more common, people could use IQVis to view the status of their Smart Space even if they are not at their homes. Suspicious behaviour can be monitored and reported.

The tool could possibly be used in mobile devices, as the case study showed. The performance of the tool was just barely satisfactory in the N900 mobile device, however, and performance considerations should be addressed in the future.

IQVis could easily be modified in a Smart Space configuration tool. With the help of a configuration tool, users could easily view the properties of the devices in Smart Spaces, adjust configurations and then see the monitored qualities.

IQVis could visualize more features, e.g., show the connectivity used in the edges connecting the device to the SS. This would help the user to spot easily how different devices are connected to the SS. In addition, the dynamic behaviour of the SS could be visualized in another view, i.e., show which security features are currently activated.

6. Conclusion

The research problem in this thesis was how software quality variability could be visualized at run-time to aid the user in analysing the adaptation results. This problem included many aspects: 1) how to visualize the change in quality attributes so that the user can analyse the variability effectively, 2) which visualization techniques are used to map the quality attribute data into a visual form, and 3) how to make the tool extensible and easily adaptable so that it can be used in different contexts.

The literature review aimed to describe: 1) what quality variability is in the context of smart environments, 2) what visualization is, and 3) how it can be used to tackle the research problem. The search for existing tools focused on the tools that were used to visualize security, as security was the main quality attribute that was visualized by the IQVis tool. The findings of the related research led to requirements for the IQVis tool to be developed with the requirements that came from the smart environment in which it was to be used.

The design of the IQVis tool was presented by creating an architecture that would comply with the requirements set for the tool. The architecture of the IQVis tool was designed so that the IQVis tool would be extensible. The design was decomposed into different components that provided the necessary functionality to conform to the set requirements.

After designing the IQVis tool, the tool was implemented with the help of a selected implementation environment, Adobe Flash. The tests were carried out with simulated quality values provided by a simulation data source. The tests proved that all of the functional requirements set at the design phase were met with the implemented prototype version. Usability was not tested in the tests or the case study, but the design of the GUI was carried out with this requirement in mind.

In the end, the finished IQVis tool was used in a real smart environment in the case study. The tool was extended with a VMonitor that received real monitoring values from the actual smart environment. The case study validated that the implemented IQVis tool actually met the requirements that were set at the design phase. The analysis of quality variability was carried out with the visualization techniques that were found to be appropriate at the design phase. The case study also showed that the IQVis tool can be used in mobile devices, although the performance tests showed that improvements should be made to increase the performance. The most important contributions of this work are:

- time slider control to allow “timer travel” in the qualities
- timeline visualization to allow the analysis of quality attribute values and their variability
- extensible architecture.

To conclude, the main goal of this work was reached. The implemented IQVis tool was validated in the context of a smart environment. The tool was used to visualize quality variability and the trade-off between different values at run-time. There are tools for design-time quality variability management, but there is a need for techniques and tools to analyse variability at run-time. One contribution of these tools is the IQVis tool designed and developed in this thesis. The IQVis tool helps to verify run-time quality variability.

References

[1] Cook D.J. & Das S.K. (2007) How smart are our environments? An updated look at the state of the art. In: Pervasive and Mobile Computing, Vol. 3, Issue 2, Design and Use of Smart Environments, March 2007, pp. 53–73.

[2] Evesti A. (2007) Quality-oriented software architecture development. Master's Thesis. University of Oulu, Department of Electrical and Information Engineering, Oulu, 57 p.

[3] Niemelä E., Evesti A. & Savolainen P. (2008) Modeling Quality Attribute Variability. In: 3rd International conference on Evaluation of Novel Approaches to Software Technology, Vol. 48, Issue 8, May 4-7, Funchal, Madeira, Portugal, pp. 631–644.

[4] Evesti A., Ovaska E. & Savola R. (2009) From Security Modelling to Run-time Security Monitoring. In: European Workshop on Security in Model Driven Architecture 2009 (SECMDA 2009), June 24, Enschede, Netherlands, pp. 33–41.

[5] Smart Objects for Intelligent Applications. (Accessed 9.10.2009). URL: http://www.sofia-project.eu/.

[6] IEEE (1998) IEEE Standard for a Software Quality Metrics Methodology. IEEE Std. 1061 1998.

[7] ISO/IEC (2001) ISO/IEC 9126-1 International Standard: Software engineering – Product quality. Part 1: Quality model. 25 p.

[8] ISO/IEC (2003) ISO/IEC 9126-2 Technical Report: Software engineering – Product quality. Part 2: External metrics. 86 p.

[9] ISO/IEC (2003) ISO/IEC 9126-3 Technical Report: Software engineering – Product quality. Part 3: Internal metrics. 62 p.

[10] Matinlassi M. & Niemelä E. (2003) The Impact of Maintainability on Component-based Software Systems. In: Euromicro conference, September 1-6, Antalya, Turkey, Vol. 29, pp. 25–32.

[11] Grover S. & Sridhar N. (2009) GenQA: automated addition of architectural quality attribute support for Java software? In: Proceedings of the 2009 ACM Symposium on Applied Computing (SAC '09), Honolulu, Hawaii, pp. 483–487.

[12] Ernst N., Yu Y. & Mylopoulos J. (2006) Visualizing non-functional requirements. In: Proceedings of the 1st International Workshop on Requirements Engineering Visualization, September 11, 2006. Minneapolis, Minnesota, USA, p. 2.

[13] Svahnberg M., van Gurp J. & Bosch J. (2005) A taxonomy of variability realization techniques. Research Articles. July 10, Software: Practice and Experience, Vol. 35, Issue 8, pp. 705–754.

[14] Etxeberria L., Sagardui G. & Belategi L. (2007) Modelling Variation in Quality Attributes. In: 1st International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS'07), January 16-18, 2007, Limerick, Ireland, pp. 51–59.

[15] Merilinna J. & Räty T. (2009) A Tooling Environment for Quality-Driven Model-Based Software Development. In: 9th OOPSLA Workshop on Domain-Specific Modeling, October 25-26, Orlando, Florida, USA, pp. 107–112.

[16] IEEE (2004) IEEE Standard for Software Verification and Validation. IEEE Std. 1012-2004 (Revision of IEEE Std 1012-1998), 2005, pp. 1–110.

[17] Boehm B.W. (1984) Verifying and Validating Software Requirements and Design Specifications. IEEE Software, Vol. 1, No. 1, 1984, pp. 75–88.

[18] Broy M., Jonsson B., Katoen J., Leucker M. & Pretschner A. (2005) Model-Based Testing of Reactive Systems. In: Advanced Lectures, Lecture Notes in Computer Science, Vol. 3472, Springer, 2005, 659 p.

[19] Levy J., Saidi H. & Uribe T.E. (2002) Combining monitors for runtime system verification. In: Electronic Notes in Theoretical Computer Science, Vol. 70, Elsevier, 2002, 16 p.

[20] Open Group, Definition of SOA. (Accessed 22.10.2009). URL: http://opengroup.org/projects/soa/doc.tpl?gdid=10632.

[21] OASIS, Standard Reference Model for Service Oriented Architecture 1.0. (Accessed 22.10.2009). URL: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf.

[22] Huhns M.N. & Singh M.P. (2005) Service-oriented computing: key concepts and principles. In: IEEE Internet Computing, January/February 2005, Vol. 9, No. 1, pp. 75–81.

[23] Nitu (2009) Configurability in SaaS (software as a service) applications. In: Proceeding of the 2nd Annual Conference on India Software Engineering Conference (ISEC '09), February 23-26, 2009, Pune, India, pp. 19–26.

[24] Retkowitz D. & Stegelmann M. (2008) Dynamic Adaptability for Smart Environments. In: Distributed Applications and Interoperable Systems, 8th IFIP WG 6.1 International Conference (DAIS 2008), Vol. 5053 of LNCS, Springer pp. 154–167.

[25] Hermann F., Blach R., Janssen D., Klein T., Schuller A. & Spath D. (2009) Challenges for User Centered Smart Environments. In: Proceedings of the 13th International Conference on Human-Computer interaction. Part Iii: Ubiquitous and intelligent interaction, July 19-24, 2009, San Diego, CA, pp. 407–415.

[26] Stajano F. (2002) Security for Ubiquitous Computing. John Wiley & Sons, Ltd, New York, 268 p.

[27] Nixon P., Wagealla W., English C. & Terzis S. (2004) Security, privacy and trust issues in smart environments. In: D.J. Cook and S.K. Das, Editors, Smart Environments: Technology, Protocols, and Applications, Wiley, 2004, pp. 249–270.

[28] Card S.K., Mackinlay J.D. & Shneiderman B. (1999) Readings in information visualization: Using vision to think. Morgan Kaufmann Publishers, San Francisco, CA, 686 p.

[29] Fekete JD., Plaisant C. (2002) Interactive Information Visualization of a Million Items. In: Proceedings of the IEEE Symposium on Information Visualization, October 28-29, 2002, Boston, Massachusetts, USA, pp. 117–124.

[30] Tory M. & Moller T. (2004) Rethinking Visualization: A High-Level Taxonomy. In: IEEE Symposium on Information Visualization, October 10-12, Austin, Texas, USA, pp.151–158.

[31] Musa S. & Parish D.J. (2007) Visualising Communication Network Security Attacks. In: Proceedings of the 11th International Conference Information Visualization, July 04-06, 2007, Zurich, Switzerland, pp. 726–733.

[32] Diehl S. (2007) Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software. Springer, New York, 187 p.

[33] Ware C. (2004) Information visualization: perception for design, 2nd ed. Morgan Kaufmann Publishers, San Francisco, CA, 580 p.

[34] Shneiderman B. (1996) The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In: Prceedings of the 1996 IEEE Symposium on Visual Languages, pp.336–343.

[35] Yi J. S., Kang Y. a., Stasko J.T. & Jacko J.A. (2007) Toward a Deeper Understanding of the Role of Interaction in Information Visualization. In: IEEE Transac-

tions on Visualization and Computer Graphics 2007, Vol.13 Issue 6, pp. 1224–1231.

[36] Chi E.H. (2000) A Taxonomy of Visualization Techniques using the Data State Reference Model. In: Proceedings of the IEEE Symposium on Information Visualization 2000, October 09-10, 2000, Salt Lake City, Utah, pp. 69.

[37] Rensink R.A. (2002) Internal vs. external information in visual perception. In: Proceedings of the 2nd International Symposium on Smart Graphics, June 11-13, 2002, Hawthorne, New York, Vol. 24, pp. 63–70.

[38] Few S. (2007) Visualizing Change An Innovation in Time-Series Analysis. Visual Business Intelligence Newsletter, September 2007. (Accessed 4.11.2009). URL: http://www.perceptualedge.com/articles/visual_business_intelligence/visualizing_change.pdf.

[39] Chen C. (2005) Top 10 Unsolved Information Visualization Problems. IEEE Computer Graphics and Applications, July-August, 2005, Vol. 25, Issue 4, pp. 12–16.

[40] Nguyen Q. V. & Huang M. L. (2004) A Focus+Context Visualization Technique Using Semi-Transparency. In: Proceedings of the fourth International Conference on Computer and information Technology, September 14–16, 2004, Wuhan, China, pp.101–108.

[41] Jankun-Kelly T.J., Kwan-Liu Ma. (2003) MoireGraphs: radial focus+context visualization and interaction for graphs with visual nodes. In: IEEE Symposium on Information Visualization (INFOVIS 2003), October 21–21, 2003, pp.59–66.

[42] Wang, Q., Wang, W., Brown, R., Driesen, K., Dufour, B., Hendren, L., and Verbrugge, C. (2003) EVolve: an open extensible software visualization framework. In: Proceedings of the 2003 ACM Symposium on Software Visualization (SoftVis '03), June 11–13, 2003, San Diego, California, pp. 37–ff.

[43] Heer J., Card S. K. & Landay J. A. (2005) Prefuse: A Toolkit for Interactive Information Visualization. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '05), April 02–07, 2005, Portland, Oregon, USA, pp. 421–430.

[44] De Pauw W., Andrade H. & Amini L. (2008) Streamsight: a visualization tool for large-scale streaming applications. In: Proceedings of the 4th ACM Symposium on Software Visualization (SoftVis'08), September 16–17, 2008 Ammersee, Germany, pp.125–134.

[45] McPherson J., Ma K-L., Krystosk P, Bartoletti T & Christensen M. (2004) PortVis: A Tool for Port-Based Detection of Security Events. In: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining For Computer Security (VizSEC/DMSEC '04), October 29, 2004, Washington DC, USA, pp. 73–81.

[46] Koike H. & Ohno K. (2004) SnortView: Visualization System of Snort Logs. Conference on Computer and Communications Security. In: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining For Computer Security (VizSEC/DMSEC '04), October 29, 2004, Washington DC, USA, pp. 143–147.

[47] Hertzog P. (2006) Visualizations to improve reactivity towards security incidents inside corporate networks. In: Proceedings of the 3rd International Workshop on Visualization For Computer Security (VizSEC'06), November 03, 2006 Alexandria, Virginia, USA, pp. 95–102.

[48] Buschmann F., Meunier R., Rohnert H., Sommerland P. & Stal M. (1996) Pattern-oriented Software Architecture: a System of Patterns. John Wiley, Chichester, 457 p.

[49] Heer J. & Agrawala M. (2006) Software Design Patterns for Information Visualization. In: IEEE Transactions on Visualization and Computer Graphics, September/October 2006, Vol. 12, Issue 5, pp. 853–860.

[50] Miettinen, T. (2008) Resource monitoring and visualization of OSGi-based software components. VTT Publications 685, VTT Technical Research Centre of Finland, Espoo, 107 p. + appendixes 3 p.

[51] Viegas F.B., Wattenberg M., van Ham F., Kriss J. & McKeon M. (2007) ManyEyes: a Site for Visualization at Internet Scale. In: IEEE Transactions on Visualization and Computer Graphics, November, 2007, Vol. 13, Number 6, pp. 1121–1128.

[52] Kuusijärvi J. (2010) A demo on using Visualization to aid Run-time Verification of Dynamic Service Systems. In: Proceedings of the 2010 IEEE International Conference on Software Testing Verification and Validation Workshop, ICSTW'10 (ViDaS'10). Paris, France, April 6–10 2010, 6 p, in press.

[53] Jones J.A., Orso A. & Harrold M.J. (2004) Gammatella: Visualizing program-execution data for deployed software. In: Information Visualization, September, 2004, Vol. 3, Issue 3, pp. 173–188.

[54] Heer J., Mackinlay J.D., Stolte C. & Agrawala M. (2008) Graphical Histories for Visualization: Supporting Analysis, Communication, and Evaluation. In: IEEE Transactions on Visualization and Computer Graphics, Vol. 14, Number 6, pp. 1189–1196.

[55] Willett W., Heer J. & Agrawala M. (2007) Scented Widgets: Improving Navigation Cues with Embedded Visualizations. In: IEEE Transactions on Visualization and Computer Graphics, November, 2007, Vol. 13, Number 6, pp. 1129–1136.

[56] Adobe Flash platform. (Accessed 1.9.2009). URL: http://www.adobe. com/flashplatform/.

[57] Adobe Open Source Flex SDK. (Accessed 1.9.2009). URL: http://open source.adobe.com/wiki/display/flexsdk/Flex+SDK.

[58] Flare visualization toolkit. (Accessed 1.9.2009). URL: http://flare.prefuse.org/.

[59] Open Source FlashDevelop. (Accessed 1.9.2009). URL: http://www.flash develop.org/.

[60] Open Source as3corelib utility library. (Accessed 1.9.2009). URL http://code.google.com/p/as3corelib/.

[61] Adobe SWF File Format Specification Version 10. (Accessed 1.2.2010). URL: http://www.adobe.com/devnet/swf/pdf/swf_file_format_spec_v10.pdf.

[62] Adobe ActionScript Virtual Machine 2 (AVM2) Overview. (Accessed 1.2.2010). URL: http://www.adobe.com/devnet/actionscript/articles/avm2overview.pdf.

[63] Evesti A., Eteläperä M., Kiljander J., Kuusijärvi J., Purhonen A. & Stenudd S. (2009) Semantic Information Interoperability in Smart Spaces. In: Mobile and Ubiquitous Multimedia (MUM'09), November 22–25, 2009, Cambridge, United Kingdom, pp. 158–159.

[64] AsUnit: Open Source unit test framework. (Accessed 5.10.2009). URL: http://asunit.org/.

[65] SOFIA (2010) Architecture, D5.21 Interoperable Service Architecture (v1.0 2010-01-01, under review, project confidential).

[66] Toninelli A., Pantsar-Syväniemi S., Bellavista P. & Ovaska E. (2009) Supporting context awareness in smart environments: a scalable approach to information interoperability. In: Proceedings of the International Workshop on Middleware for Pervasive Mobile and Embedded Computing, pp. 1–4.

[67] VTT Technical Research Centre of Finland & Vaisala Oyj Internet Weather Station. (Accessed 1.2.2010). URL: http://weather.willab.fi/weather.html.en.

[68] Nokia N810 Technical Specifications. (Accessed 12.2.2010). URL: http://europe.nokia.com/support/product-support/nokia-n810/specifications.

[69] Nokia N900 Technical details. (Accessed 12.2.2010). URL: http://maemo.nokia.com/n900/.

[70] Adobe Labs web site. (Accessed 12.2.2010). URL: http://labs.adobe.com/technologies/flashplayer10/.

[71] Kuusijärvi J, Evesti A. and Ovaska E. (2010) Visualizing Structure and Quality Properties of Smart Spaces. Submitted to: Semantic Interoperability of Smart Spaces, 2010, 6 p.

[72] Chittaro L. (2006) Visualizing Information on Mobile Devices. In Computer, March, 2006, Vol. 39, Issue 3, pp. 40–45.

[73] GraphML File Format. (Accessed 1.2.2010). URL: http://graphml.graphdrawing.org/

Series title, number and report code of publication

VTT Publications 746
VTT-PUBS-746

Author(s)
Jarkko Kuusijärvi

Title
Interactive visualization of quality variability at run-time

Abstract
Smart environments are dynamic in nature, and the software running in these environments requires quality adaptations in order to function efficiently. The result of these adaptations, i.e., quality variability, must be verified in some way, and visualization can be used to aid this verification process. The research problem in this work was to find suitable visualization techniques to visualize quality variability and implement a visualization tool that encompasses these techniques and provides an interactive visualization of quality variability for the user.

As a solution to the research problem, this work presents an interactive quality visualization tool. The requirements specification for the implemented tool was derived from the literature review and the intended usage context of the tool, i.e., smart environments. The literature review explores a set of applicable visualization techniques and compares existing visualization tools with regard to the features required to represent quality variability visually at run-time.

The visualization techniques selected for the tool include interactive timelines, charts and meters that enable analysis of the quality attributes and their variability in different time ranges or points in time. Some additional visualization techniques were also included such as treemaps and graphs to visualize the structure of the smart environment.

The visualization techniques include open source visualization techniques and self-made techniques designed and implemented from the start to cover the specific requirements set for the tool. The main contribution of this work is the visualization tool that can be used to visualize different quality attributes and their variability. Moreover, the tool can easily be deployed in different environments due to its architecture and the selected implementation technologies that make the solution extensible and portable.

The implemented visualization tool was evaluated in the context of a smart environment in which security was adapted at run-time. The case study demonstrated that the implemented tool can be used in the analysis of the variability of different quality attributes. The trend of a single quality attribute can be studied for different time ranges or points in time according to need. The relationships between different quality attributes can also be studied with the help of appropriate visualization techniques. In addition, the visualization tool was successfully tested on mobile devices.

ISBN
978-951-38-7412-4 (URL: http://www.vtt.fi/publications/index.jsp)

Series title and ISSN	Project number
VTT Publications 1455-0849 (URL: http://www.vtt.fi/publications/index.jsp)	24506

Date	Language	Pages
October 2010	English, Finnish abstr.	111 p.

Name of project	Commissioned by
EVOLVE	VTT

Keywords	Publisher
Software visualization, quality attributes, security, smart environments, adaptive software	VTT Technical Research Centre of Finland P.O. Box 1000, FI-02044 VTT, Finland Phone internat. +358 20 722 4520 Fax +358 20 722 4374

Julkaisun sarja, numero ja raporttikoodi
VTT Publications 746
VTT-PUBS-746

Tekijä(t) Jarkko Kuusijärvi	

Nimeke

Laadun varioituvuuden interaktiivinen visualisointi ajoaikana

Tiivistelmä

Älykkäät ympäristöt ovat luonteeltaan dynaamisia ja vaativat niissä ajettavilta ohjelmistoilta kykyä muuntautua vastaamaan ympäristön tilannetta; adaptoitua, jotta ohjelmistot pystyisivät toimimaan suorituskykyisesti. Laadun variointi eli adaptointi pitää pystyä todentamaan, ja todentamisessa voidaan käyttää hyväksi visualisointia. Tässä työssä tutkimusongelmana on ollut löytää laadun varioituvuuden visualisointiin soveltuvat visualisointitekniikat ja toteuttaa visualisointityökalu, joka toteuttaa nämä visualisointitekniikat ja tarjoaa käyttäjälle vuorovaikutteisen visualisoinnin laadun varioituvuudesta.

Ratkaisuna tutkimusongelmaan tässä työssä esitellään laadun varioituvuuden interaktiivinen visualisointityökalu. Visualisointityökalun vaatimusmäärittely johdettiin taustatutkimuksesta ja työkalun oletetusta käyttökohteesta, älykkäistä ympäristöistä. Taustatutkimuksessa esitellään soveltuvia visualisointitekniikoita ja vertaillaan olemassa olevia visualisointityökaluja ottaen huomioon ominaisuudet, jotka tarvitaan laadun varioituvuuden ajoaikaiseen visualisointiin.

Visualisointityökaluun valittuihin visualisointitekniikoihin kuuluvat muun muassa vuorovaikutteiset viivadiagrammit, kaaviot ja erilaiset mittarit, joiden avulla laatuattribuuttien arvoja ja varioituvuutta voidaan analysoida eri aikaväleillä. Yleisten visualisointitekniikoiden lisäksi työkaluun sisällytettiin myös muita soveltuvia visualisointitekniikoita, kuten puukarttoja ja graafeja, joiden avulla älykkään ympäristön rakennetta visualisoidaan.

Työkalun visualisointitekniikat sisältävät valmiiksi toteutettuja avoimen lähdekoodin visualisointitekniikoita, joiden lisäksi suunniteltiin ja toteutettiin visualisointitekniikoita kattamaan työkalulle asetetut vaatimukset. Työn pääsaavutuksena kehitetään työkalu, jota voidaan käyttää eri laatuattribuuttien ja niiden varioituvuuden visualisoimisessa. Sen lisäksi työkalun arkkitehtuuri ja valitut toteutusteknologiat mahdollistavat työkalun käytön eri ympäristöissä sen laajennettavuusominaisuuden ja siirrettävyyden ansiosta.

Toteutettu visualisointityökalu testattiin käyttäen kontekstina älykästä ympäristöä ja siinä tapahtuvaa tietoturvan ajoaikaista adaptointia. Malliesimerkki osoitti, että työkalun avulla eri laatuattribuuttien varioituvuutta pystytään tutkimaan usealla tavalla. Yksittäisen laatuattribuutin kehityssuuntaa pystytään arvioimaan määrätyllä ajanjaksolla ja useiden laatuattribuuttien välisiä yhteyksiä pystytään myös tutkimaan usealla eri visualisointitekniikalla. Lisäksi visualisointityökalua testattiin mobiililaitteissa onnistuneesti.

ISBN 978-951-38-7412-4 (URL: http://www.vtt.fi/publications/index.jsp)		
Avainnimeke ja ISSN VTT Publications 1455-0849 (URL: http://www.vtt.fi/publications/index.jsp)		Projektinumero 24506
Julkaisuaika Lokakuu 2010	Kieli Englanti, suom. tiiv.	Sivuja 111 s.
Projektin nimi EVOLVE		Toimeksiantaja(t) VTT
Avainsanat Software visualization, quality attributes, security, smart environments, adaptive software		Julkaisija VTT PL 1000, 02044 VTT Puh. 020 722 4520 Faksi 020 722 4374

VTT CREATES BUSINESS FROM TECHNOLOGY

Technology and market foresight • Strategic research • Product and service development • IPR and licensing • Assessments, testing, inspection, certification • Technology and innovation management • Technology partnership

VTT PUBLICATIONS

726 Jaana Leikas. Life-Based Design.A holistic approach to designing human-technology interaction. 2009. 240 p.

727 Teemu Kanstrén. A Framework for Observation-Based Modelling in Model-Based Testing. 2010. 93 p. + app. 118 p.

728 Stefan Holmström. Engineering Tools for Robust Creep Modeling. 2010. 94 p. + 53 p.

729 Olavi Lehtoranta. Innovation, Collaboration in Innovation and the Growth Performance of Finnish Firms. 2010. 136 p. + app. 16 p.

730 Sami Koskinen, Sami. Sensor Data Fusion Based Estimation of Tyre-Road Friction to Enhance Collision Avoidance. 2010. 188 p. + app. 12 p.

732 Venkata Gopalacharyulu Peddinti. Data integration, pathway analysis and mining for systems biology. Espoo 2010. 62 p. + app. 67 p.

733 Johanna Kirkinen. Greenhouse impact assessment of some combustible fuels with a dynamic life cycle approach. Espoo 2010. 63 p. + app. 58 p.

734 Antti Grönroos. Ultrasonically Enhanced Disintegration. Polymers, Sludge, and Contaminated Soil. 2010. 100 p. + app. 27 p.

735 Michael Lienemann. Characterisation and engineering of protein–carbohydrate interactions. Espoo 2010. 90 p. + app. 30 p.

736 Jukka-Pekka Pesola. Building Framework for Early Product Verification and Validation. Master Thesis. Espoo 2010. 75 p.

737 Virpi Oksman. The mobile phone: A medium in itself. Espoo 2010. 89 p. + app. 132 p.

738 Fusion Yearbook. Association EURATOM-TEKES. Annual Report 2009. Eds. by Seppo Karttunen & Markus Airila. 2010. 136 p. + app. 13 p.

739 Satu Hilditch. Identification of the fungal catabolic D-galacturonate pathway. Espoo 2010. 74 p. + app. 38 p.

740 Mikko Pihlatie. Stability of Ni-YSZ composites for solid oxide fuel cells during reduction and re-oxidation. Espoo 2010. 92 p. + app. 62 p.

741 Laxmana Rao Yetukuri. Bioinformatics approaches for the analysis of lipidomics data. Espoo 2010. 75 p. + app. 106 p.

742 Elina Mattila. Design and evaluation of a mobile phone diary for personal health management. Espoo 2010. 83 p. + app. 48 p.

743 Jaakko Paasi & Pasi Valkokari (eds.). Elucidating the fuzzy front end – Experiences from the INNORISK project. Espoo 2010. 161 p.

744 Marja Vilkman. Structural investigations and processing of electronically and protonically conducting polymers. 2010. 62 p. + app. 27 p.

746 Jarkko Kuusijärvi. Interactive visualization of quality variability at run-time. Espoo 2010. 111 p.

ISBN 978-951-38-27xx-0 (soft back ed.)
ISSN 1235-0621 (soft back ed.)

ISBN 978-951-38-27xx-0 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1455-0849 (URL: http://www.vtt.fi/publications/index.jsp)