An architecture for flexible, real-time monitoring and control of real or simulated devices

*challenges for a major oil-company in the 21st century*

Mark W. Hissink Muller
<m.w.hissinkmuller@tbm.tudelft.nl>
<mark@hissinkmuller.nl>

30th November 2001
# Contents

1 Situational description .......................... 1
   1.1 Sun Microsystems Inc. .......................... 1
   1.2 iForce initiative ............................. 2
   1.3 iForce Ready Center Menlo Park - Trade exchanges group ... 2
   1.4 Systems Engineering Group at Delft University of Technology 2

2 Introduction E-Gas case ......................... 4
   2.1 Company description .......................... 4
   2.2 Business processes at a gas station ................ 5
   2.3 Problem for E-Gas company ........................ 7
   2.4 Problem description, thesis goal, thesis questions ........... 8
      2.4.1 Problem summary .......................... 8
      2.4.2 Thesis goal ............................... 8
      2.4.3 Thesis questions ........................... 8
   2.5 Methodological approach ....................... 9
      2.5.1 Way of thinking ............................ 9
      2.5.2 Way of working ................................ 10
      2.5.3 Way of modelling ........................... 11
      2.5.4 Way of controlling .......................... 11
   2.6 Structure of the report ........................ 11

3 Conceptualization of an architecture .......... 13
   3.1 Requirements for a solution from the interview .............. 13
   3.2 Suggested solution; an ERP-system ........................ 14
   3.3 E-Gas case example ................................ 14
   3.4 Conceptualization of the architecture ....................... 15
      3.4.1 Actors have a need for information from devices .......... 15
      3.4.2 Devices have a virtual representation ................... 16
      3.4.3 Actors have a virtual representation ..................... 16
      3.4.4 Information-extracting filters deliver relevant events to virtual actors .................................. 17
      3.4.5 Access to system functionality is restricted by an admin-istrator ........................................ 17
      3.4.6 An actor interacts with system functionality through a business rule .................................. 18
3.4.7 Various GUI-panels allow actors to receive events anytime, anywhere .................................................. 18
3.5 An architecture as solution ................................................. 19

4 Detailed overview of the architecture 22
4.1 Architecture interaction .................................................. 22
4.2 Anarchic topology based on a standard component ............ 22
4.3 Various components in the architecture ......................... 25
  4.3.1 Architecture components ........................................ 25
  4.3.2 Message objects ................................................... 26
4.4 Generic design questions for architecture components .... 27

5 Generic building blocks and sub-components 28
5.1 Basic Component ......................................................... 28
  5.1.1 A Basic Component’s passport ................................... 28
  5.1.2 A Basic Component’s sub-components ....................... 29
5.2 NaradComponent ......................................................... 30
5.3 FrameworkMessage ....................................................... 30
5.4 Messaging component ................................................... 30
  5.4.1 Component interface ............................................... 30
  5.4.2 One messaging sub-component for all components .... 31

6 Real Device 32
6.1 Answers to generic design questions ............................... 32
6.2 Link between a virtual device and a real device ............ 33
6.3 Wrapped devices ........................................................ 34
  6.3.1 Wrapped real devices ............................................ 34
  6.3.2 Simulated devices ................................................ 35
6.4 Real Devices in E-Gas Station ....................................... 35

7 Virtual Device 36
7.1 Answers to generic design questions ............................... 36
7.2 Connection between virtual and real device .................. 37
7.3 What information should be made available? ................. 37
7.4 Virtual devices in E-Gas Station ................................. 38

8 Virtual Actor 40
8.1 Answers to generic design questions ............................... 40
8.2 A new actor joins the architecture ............................... 41
8.3 Suggested virtual actor sub-components ....................... 42
8.4 Virtual actors in E-Gas Station .................................... 42

9 Virtual Administrator 44
9.1 Answers to generic design questions ............................... 44
9.2 Multiple administrators cooperating ............................ 45
9.3 Technical Administrator .............................................. 46
9.4 Virtual Administrators in E-Gas Station ....................... 46
10 The GUI Panel and its components
10.1 Answers to generic design questions ......................... 47
10.2 Two types of panel components ................................. 48
  10.2.1 Display Panel Component ................................. 48
  10.2.2 Control Panel Component ................................. 48
10.3 Actor panels in E-Gas Station ................................. 49

11 Filter
11.1 Answers to generic design questions ......................... 50
11.2 Trusted Filters .............................................. 52
11.3 Filter location .............................................. 52
11.4 Subscription to a Filter ..................................... 53
11.5 Types of Filters ............................................. 53
11.6 Different types of filters .................................... 53
  11.6.1 State Change Filter ...................................... 54
  11.6.2 State Enter Filter ....................................... 54
  11.6.3 Threshold Filter ......................................... 54
11.7 Filters in E-Gas Station ..................................... 55

12 Business Rule
12.1 Answers to generic design questions ......................... 56
12.2 Examples of Business Rule functionality .................... 57
12.3 Business Rule impact assessment .............................. 57
12.4 Compound interactive component .............................. 58
12.5 Business Rules in E-Gas Station .............................. 58

13 Moving components
13.1 Request For Rights ........................................... 59
  13.1.1 Subscribing to a Filter .................................. 59
  13.1.2 Issuing a Business Action ............................... 60
  13.1.3 Request For Rights properties .......................... 60
  13.1.4 Requesting Rights in a Multi-Actor environment ..... 61
13.2 Requests for Rights in E-Gas Station ......................... 61
13.3 Business Event .............................................. 62
13.4 Business Action ............................................. 63
13.5 Business Actions in E-Gas Station ........................... 63

14 Reflection on the architecture
14.1 Architecture meets E-Gas requirements ....................... 64
14.2 Architecture flexibility for E-Gas ........................... 65
14.3 Generalization of architecture applicability .................. 66
14.4 Limitations of the architecture’s conceptual model ........... 67
14.5 Limitations of the current implementation of the architecture . 69
14.6 Answers to the research questions ............................ 69
15 Conclusion and Recommendations

15.1 Conclusion

15.2 Recommendations

15.2.1 Recommendations for E-Gas management

15.2.2 Recommendations for Sun Microsystems and the iForce Ready Center

15.2.3 Recommendations for further research on the architecture

15.2.4 Validation of the architectural concepts and its implementation

A Definitions

A.1 Definitions regarding the architecture

A.2 Security specific definitions

A.3 General definitions

B E-Gas Station

B.1 Actors involved

B.2 Means-Ends analysis

C Theoretical notions

C.1 Organizational engineering / Business Engineering

C.1.1 Organizational engineering

C.1.2 Business engineering

C.2 Decision support in a multi-actor environment

C.3 Systems engineering / Software engineering

C.3.1 Systems engineering

C.3.2 Software engineering

C.4 Architecture confrontation with theory

C.4.1 Architecture requirements from theory

C.4.2 Disciplinary intersections

C.4.3 Combined requirements for the architecture

C.4.4 Confronting the architecture with theoretical requirements

D Concepts on Decision Support, Systems Engineering

D.1 Decisionmaking theory

D.1.1 Problem driven approach

D.1.2 Requirements for an information architecture

D.1.3 Usefulness of formal models for decision making

D.2 Event-based way of asking questions

D.3 Theory of Subsidiarity

E Concepts on Administration, messages between systems

E.1 A new paradigms in administration

E.1.1 Traditional distribution of rights

E.1.2 Requests for Rights in a formalized way

E.1.3 Automatic generation of requests

E.1.4 Granting rights only once
E.1.5 Right decisions with the right actors .......................... 105
E.2 Synchronous vs. Asynchronous communication .................. 105
  E.2.1 Synchronous communication ................................. 105
  E.2.2 Asynchronous communication ............................... 105
  E.2.3 Messaging component reference implementation ........... 106

F Concepts on filters 108
  F.1 Filter instantiation; various conceptual alternatives .......... 108
    F.1.1 Virtual Actor Instantiates Filter .......................... 108
    F.1.2 Virtual Device instantiates Filter ......................... 109
    F.1.3 Administrator Instantiates Filter .......................... 109
  F.2 Objective vs. Normative information in Business Events ...... 110
  F.3 Data, information or knowledge? ............................. 111
  F.4 Cooperative Interactive Components in a Trusted Environment 111
    F.4.1 Example of components in a trusted environment ......... 113
    F.4.2 Introduction of a Third party ............................ 113

G Concepts on Security and Cryptography 114
  G.1 Concepts of Cryptography ..................................... 114
    G.1.1 Introduction .............................................. 114
    G.1.2 Cryptography without a key ................................ 115
    G.1.3 Secret key cryptography ................................... 115
    G.1.4 Public key cryptography ................................... 115
  G.2 Trust models .................................................. 116
  G.3 Security ...................................................... 117
    G.3.1 Different dimensions of security ........................... 117
    G.3.2 Security at connection level vs. Security at message level 118
  G.4 Architecture security .......................................... 119
# List of Figures

2.1 E-Gas Supply chain .................................................. 5  
2.2 Way-of framework .................................................... 9  

3.1 Graphical overview of E-Gas carwash example .................... 15  
3.2 E-Gas Station instantiation .......................................... 20  
3.3 Architecture instantiation for the carwash example ............... 21  

4.1 A generic instantiation of the architecture ......................... 23  
4.2 Possible architecture interaction .................................. 23  
4.3 Topology of components in an architecture instantiation .......... 24  
4.4 Hierarchy of non-moving components ............................... 26  
4.5 Hierarchy of moving objects ....................................... 26  

6.1 Connection between a real device and a virtual device ........... 33  

11.1 Filter hierarchy ..................................................... 50  
11.2 Administrator instantiates filter ................................ 53  
11.3 Threshold filter explained ....................................... 55  

12.1 Actor invokes business rule ....................................... 58  
14.1 Architecture application: A Jini-based surveillance camera ...... 66  

B.1 Means-Ends hierarchy for E-Gas ................................ 82  

C.1 Authorization in a distributed environment ....................... 86  
C.2 Waterfall model of software life cycle ............................ 88  
C.3 Java community process .......................................... 89  
C.4 Intersection between theoretical disciplines ..................... 91  

D.1 Relation between decision and system ............................. 98  
D.2 Information flows .................................................. 99  
D.3 Actor takes action after receiving an business event ............. 101  

E.1 An actor requests rights in a formal way .......................... 104  
E.2 Message Handling Component, reference implementation ....... 107  

F.1 From data to information to knowledge ........................... 111  
F.2 Cooperative filter in a trusted environment ..................... 112  
F.3 Filter instantiation in a trusted environment .................... 112
G.1 Cryptography, basic concept ........................................ 114
G.2 Secret key cryptography ............................................ 115
G.3 Public key cryptography ............................................ 116
Abstract

As the use of electronic systems has increased rapidly over the past three decades, both executives as operational managers in today’s distributed business environment are being overloaded with information. Gigabytes of data are generated and written to storage systems each day and yet managers complain they don’t have the right information to take decisions in their demanding business environment. A large number of companies still rely heavily on difficultly accessible legacy systems, which poses a threat to having an adequate picture before taking a decision.

This document describes an architecture to unlock information available in all kinds of devices at various organizations within in a distributed environment. Driven by the business problems of a large oil-company, where thousands of different actors need information from legacy devices available at 23,000 gas stations throughout the country, a solution has been conceptualized which makes it possible for any actor to interact with any device in the business environment. Special listening filters can be instantiated to notify actors of events in the system relevant for their work. Information from devices would enable the oil-company to optimize processes and take decisions to cut operational costs considerably.

Using distributed and unconventional concepts on the administration of business-systems, the architecture supplies different actors in the value chain with flexible tools for both monitoring of devices as system control, allowing actors to remotely act on one or more devices. From generic building blocks an actor-specific management cockpit can be built which delivers the right information to support decisions, to the right actor, at the right time.
Chapter 1

Situational description

This report is a general introduction of a six month project carried out at Sun Microsystems’s iForce Ready Center in Menlo Park, CA, USA. The project took place in iForce’s Trading exchanges group and was funded by Sun Microsystems Inc. The research work resulted from a cooperation between Delft University of Technology (TU Delft) and Sun Microsystems Inc. (Sun), and was carried out in the first half of 2001 by three students of TU Delft’s faculty of Technology, Policy and Management as a Master Thesis project. Objective for the project was to develop a solution, based on new concepts, which would allow a wide range of actors to subscribe to information and interact with devices in a distributed business environment. The project resulted a generic architecture for flexible, real-time, remote monitoring and control, which is presented in this thesis. The solution was developed using requirements and examples form the E-Gas business case, but has much wider applicability.

The project was coached by Prof. Dr. H.G. Sol, Dr. Ir. A. Verbraeck and Dr. P.W.G. Bots on behalf of the faculty of Technology, Policy and Management and by Drs. L. Bonebakker on behalf of the Sun Microsystems in Menlo Park, CA. The following paragraphs introduce Sun Microsystems, the iForce initiative, the Trading exchanges group and the Systems Engineering Group at Delft University of Technology.

1.1 Sun Microsystems Inc.

Sun Microsystems, a $20 billion company founded 18 years ago with offices in 170 countries, provides end-to-end solutions for doing business in the network age. Sun’s products and services cover a wide range the aspects of ICT\(^1\), from reliable and scalable high-end enterprise servers, to the platform independent and object-oriented Java programming language.

\(^1\)Information and Communication Technology
1.2 iForce initiative

The 'iForce Initiative' within Sun as a part of Global Technical Operations aims to provide solution blueprints for mission critical applications that Run-On-Sun to deliver sustained business advantages for large companies.

The blueprints are not only designed, but also tested by the customer in the 'Proof of Concept' trajectory. During a 'Proof of Concept', the critical parts of the customer's solution are actually built in the lab to be able to establish and 'prove' system and application performance of the proposed blueprint.

The iForce Ready Centers consist of various groups with different focus. There are product-specific groups like the SAP-group, the Oracle/PeopleSoft-group and the iPlanet-group, as well as functional-specific groups like the Supply Chain Management-group and the Trade exchanges-group, where this project was conducted.

1.3 iForce Ready Center Menlo Park - Trade exchanges group

The Trade exchanges-group within iForce focusses on large e-Business problems that involve trade exchanges, also known as virtual marketplaces. Many Fortune 500 enterprises come to the iForce Center get Sun's help on realizing maximum benefit from the opportunities of today's networked digital economy. Because of the complexity in technology involved and the fact that trade exchanges are such a new subject, very specific expertise is necessary to get an E-business system that involves a trade exchange working. In the iForce lab 'Proof Of Concept' is established for these complex cases.

1.4 Systems Engineering Group at Delft University of Technology

Founded in 1864, Delft University of Technology is the oldest, largest, and most comprehensive technical university in The Netherlands. With over 13,000 students and 2,100 scientists, including 200 chaired professors, it is an establishment of both national importance and significant international standing. Renowned for its high standard of education and research, the University collaborates with other educational establishments and research institutes, both in the Netherlands and overseas. It also enjoys partnerships with governments, branch organizations, numerous consultancy firms, the industry, and companies from the small and medium business sectors.

Systems Engineering (SE) is a research group within the faculty of Technology, Policy and Management (TPM). As such its main strengths are in multi-disciplinary projects. Central research issues for TPM are the process of problem analysis & solving and complex (system) design trajectories. One of the research directions for SE is ICT, focusing on E-commerce applications in multi-actor settings. Group technology and simulation tools are being developed, tested
and used within SE to conduct this research. A current research theme, in which these tools are used, is the modelling of complex systems, including their dynamics and control with component based architectures.

The next chapter introduces the E-Gas business case which was used throughout the project for development of the architecture.
Chapter 2

Introduction E-Gas case

This chapter introduces the case of a major player in the oil-business that came to Sun’s iForce Ready Center because its management felt the company is losing money due to inefficiency of its business processes and downtime of services.

Currently, many actors throughout the company’s business chains lack the information needed to optimize the primary and support processes which, as an internal report turned out, could significantly reduce costs and improve competitiveness. Sun’s iForce Ready Center management feels the problems found in the E-Gas business case are seen in many large organizations.

This chapter starts with a section containing background information on the company, describes the company’s relevant business processes from the perspective of a single gas station and introduces the company’s problem. This chapter concludes with a short summary of the problem for E-Gas, the goal of this thesis and the research questions.

2.1 Company description

E-Gas\(^1\) describes the case of a company, an alliance between two major oil-companies, cooperating in the field of exploration, production and sale of oil-products and services throughout the US. Production facilities in the US cover 8 refineries, which refine over 1.3 million barrels of oil per day. The alliance owns 29,000 miles of pipelines\(^2\) and distributes its oil-products over 23,000 gas stations. In the year 2000 the alliance’s total revenue was US$69 billion, with a market share of 14.5%. The alliance employs 13,000 people, working from exploration to the sale of gas and services.

Figure 2.1 depicts a typical representation of the E-Gas supply chain, showing how crude oil is won in exploration, refined at one of the refineries and eventually brought to one of 23,000 gas stations all over the US. As indicated by the actors on the picture, a wide range of different internal and external parties are involved in the process of exploration, production, distribution and sale of oil-products. From top-management all the way down to maintenance

\(^{1}\) throughout this thesis E-Gas refers company and E-Gas Station refers to the networked gas station

\(^{2}\) information gathered from one of the the company’s websites
contractors and people responsible for delivery of new candy-bars to any of the many gas station locations, actors need information to support their decision making.

In over 23,000 gas stations in the US, not all company-owned, money is made in sales of gas and services, such as food and beverages and carwash-services. The operational margin on the sale of gas is relatively low, but because of the huge volume of the market, still a considerable amount of money is made. The margin on the sale of services and non-gasoline products, such as food, beverages and carwash services, is considerably higher, however, this sale is much smaller in volume.

Since substantial amount the costs are made and all revenues are generated close to the end-customer, the next section gives an overview of the most important downstream business processes in the company, zooming in on a single gas station, since one gas station is representative for each of the 23,000 throughout the entire US.

2.2 Business processes at a gas station

From the perspective of one gas station, several processes and devices are relevant for the business case. The processes and devices described in this section are an abstraction of the real business situation, since not every gas station is identical in size and services offered.

At first it might seem, that the processes which are most important to the

---

3 downstream processes are all the business processes regarding the sale of end-products and services, compared to the upstream processes which are involved with the exploration and production of oil-products
gas company are those which involve the sale of gas and services, such as the sale of carwash-services and non-oil products. Although these are the business processes where the actual money is made, the sale of gas and services halts soon if no supplies are delivered or broken devices are not repaired.

Several business processes are relevant, since they are the primary, in the sense that reflect where customers bring in their money:

- sale of gas
- sale of food and beverages
- sale of carwash-services

For the support of the primary processes other processes are important also. Support processes are, for example:

- delivery of gas to the gas station
- delivery of supplies and products to a gas station, e.g. soap for the carwash-installation
- maintenance and repair of gas station infrastructure

Over the entire US several hundreds of different suppliers and maintenance contractors are responsible for the support processes just mentioned. Responsible actors want to optimize these processes, reduce costs and leverage revenue. They need information to do that.

When looking at the business processes just described in more detail, it becomes clear that with all of the processes devices are involved, either because at these devices the customer directly interacts with the system, or because they support or enable the primary processes. The following list identifies some of the devices which are relevant in the business processes at a gas station:

- gas pumps
- underground pump
- point of sales terminal
- carwash installation
- refrigerator inside the shop

From the description of the most important processes and devices it is obvious that with every business process, devices play a key part. Information available in the devices contains vital steering information for actors in the business environment, with regard to business processes. As an example, a description of various types of information available within a carwash-installation is given:

- soap level
- wax level
• water pressure level
• air pressure level

An example of an actor using information from this device is the following:

Example 2.1 A gas station clerk working behind the counter of the gas station needs to act whenever the soap level drops below a certain threshold. If so, he/she needs to refill the soap from the stock which is available in the gas station.
To support his business process of refilling the soap and keeping the carwash operational, the clerk working behind the counter in the gas station would like receive the right information (soap went below a certain threshold level) at the right time (the moment the event occurred).

Currently, the only way for the clerk to know the soap level is to physically walk up to the carwash-installation.

2.3 Problem for E-Gas company

Due to various external factors, such as the economic downturn and the strong competition, the company’s profit margin is under pressure.
In order to boost overall profitability, company management is looking for ways to cut costs, while at the same time increasing the company’s overall levels of service. Currently, the company is under the assumption that several of the business processes do not perform at high levels of efficiency and that the right steering-information is lacking to optimize these processes. Apart from that, company management thinks that information available in devices can be used (in an aggregated form) to level the information asymmetry between maintenance contractors and the E-Gas company. As contract-terms for maintenance and delivery of supplies and getting ever shorter, information regarding these processes would allow the company to re-negotiate sharply.

Downtime of services, such as the carwash being broken or a gas station being out of gas, not only lead to missed business opportunities, but also damage to the company’s strong brand-names. The following examples are typical for what is met in every day business, hundreds of times every month in more than 23,000 gas stations across the US.

Example 2.2 A carwash installation at a gas station was not functioning and in need of service for more than a week before maintenance staff was notified of this breakdown. During that week, the carwash did not contribute to company revenues.

Example 2.3 A maintenance employee visited a gas station location to do routine checks on certain installations. The next week, one of the gasoline pumps breaks down. If the maintenance employee would have had insight in the power-use of the pump and had been be able to see that the power-use had been increasing considerably over the past few weeks\(^4\), the pump could have been replaced.

\(^4\)an increase in power-use is a sign for a pump which is about to break down
during a regular visit as a precaution. Now an extra, unscheduled visit, at extra costs, is needed, in addition to the loss of revenue since the system is non-operational for a period of time.

This lack of insight in critical operations has substantial financial consequences for the company which has over 23,000 gas stations in large parts of the United States.

Sun’s iForce Ready Center was asked by the management of E-Gas to contribute to a state of the art, technical solution which would unlock various sorts of information available within devices all over the company’s distributed business environment. This information is to be used by different actors involved, to optimize processes with the ultimate goal of lowering operational costs.

2.4 Problem description, thesis goal, thesis questions

This section describes shortly summarizes E-Gas’s problem, highlights this thesis’s goal and poses various thesis questions.

2.4.1 Problem summary

Throughout the entire E-Gas business environment, information available in devices doesn’t reach the right actor at time. Because actors do not have the right information at the right time to steer and optimize processes, money is lost due to inefficiency every day.

2.4.2 Thesis goal

Goal of this thesis is to conceptualize a solution, which will give actors in the distributed business environment of the E-Gas case a tool for remote monitoring and control of various types of devices, so information can be used to improve business efficiency and levels of service. This thesis aims to conceptualize a solution to supply the right actor with the right information at the right time and allow him/her to act upon it.

2.4.3 Thesis questions

The thesis goal leads to three thesis questions:

1. What are the requirements for a solution which gives various actors in E-Gas’s distributed business environment the ability to remotely monitor and control devices?

2. Based on these requirements, what would a solution which meets these requirements look like?

3. What does an implementation of the architecture’s conceptual model in the Java-language look like?

The following section describes a way-of framework to characterize the research project.
2.5 Methodological approach

This section reflects on the design process during the project using a way of framework [Sol, 2000], describing the way of thinking, way of working, way of modelling and way of controlling, as depicted in figure 2.2

![Way-of framework](image)

Figure 2.2: Way-of framework

2.5.1 Way of thinking

The way of thinking for the project is best characterized by the theoretical notions found in appendix C and C.4, which is the result of extensive study. In short, the following items are illustrative for the way of thinking:

- multi-actor view on various parties involved in business
- actor information-need based on problem-perception
- dynamics of organizations, business environments
- business processes transcend single organizations
- a systems perspective on organizations and (inter-) organizational processes
- importance to have the physical actor and administrator in the loop (see the remark made by Peter Keen, al little further in this subsection)
- request-based interaction with human-administrators to solve (administrative) bottlenecks
- the system notifies the actor, instead of the actor constantly queries the system
- flexibility to shift the panels between a technical and organizational solution to cope with system dynamics
- security and authentication as a design principle
• open standards and well-defined interfaces, wrapping of components allowing legacy devices to be a part present of the solution, instead of being the problem

Peter Keen⁵, an authority in the field of business engineering, made a striking remark:

"When every firm in an industry has access to the same technology, the competitive edge comes from making the management difference."

Peter G. W. Keen

During the project, this idea was used to recognize the need for a flexible solution which allows actors to automate, but also to detach automated processes which require human actor-steering.

2.5.2 Way of working

The way of working during the research project is characterized as a generic middle-out approach, together with an combined incremental and iterative approach.

Middle-out: This section describes three different approaches the E-Gas business case; top-down, middle-out and bottom-up. It argues why the middle-out approach was most applicable for this case. Different ways of approaching the E-Gas business case are:

• Top-down: A top-down approach would start with interviewing various actors in the business chain, examine high-level (maintenance) procedures, inter-organizational contracts, various suppliers in different area’s, etc.

• Middle-out: The middle-out approach starts at the intermediate-level of one gas station, identify relevant devices there and match these with actor information-need and business processes.

• Bottom-up: The bottom-up approach would start looking from individual devices, their specific interfaces and information exchange, from there look up to match the information with specific actors.

The middle-out approach, focussing on the gas station level was considered most applicable for the following reasons:

• a lot of the cost-intensive, inefficient, support and maintenance processes are related to operations at gas station level

• a gas station, at the end of the business chain, is where the money is made by selling products and services to the customer

⁵www.peterkeen.com
Incremental and iterative approach: The way of working is also characterized by a combination of an incremental and an iterative approach. The incremental process is characterized by the following steps:

- analysis of the business case and problem
- generation of requirements for a solution
- conceptualization of the solution and its implementation in the Java-language
- reflection on the solution and its implementation

Step 3 in the incremental process was iterative in nature, with iterations of about 3 to 5 weeks between shaping of idea’s and conceptualization, and implementation of these concepts in the Java-language, which lead to re-evaluation of the concepts.

2.5.3 Way of modelling

As 'Way of modelling', UML-techniques [Fowler, 2000] for modelling interactions between software components have been used. Special emphasis is put on the difference between symmetric communication and asymmetric communication, as described in E.2 on page 105. The asymmetric communication between various components in the architecture allowed loose coupling of functionality and moving responsibility for certain functionality to other components.

2.5.4 Way of controlling

During the project, at several delivery moments, progress was established by freezing the conceptual idea’s during a joint session with either the coach at Sun, or the coach from Delft University of Technology. Based on these sessions, direction and targets for the following weeks were discussed and established.

Based on the steps described in the way of working the following section describes the structure of the report.

2.6 Structure of the report

Chapter 2 introduces the business case and problem for E-Gas. The chapter also covers the problem summary, thesis goal and thesis questions.

Based on the problem description presented in chapter 2, chapter 3 continues with the requirements for a solution as derived from the interview with one of E-Gas’s senior managers. After that this introduces the conceptual architecture which is the result of this thesis work by means of an example typical for the business case. Chapter 4 looks at the architecture in more detail.

The chapters following the introduction of the architecture (starting with chapter 5 until chapter 13) give an detailed description of the concepts of the architecture with examples of how these concepts are instantiated for and used in the E-Gas business case.
Reflection on the architecture and thesis questions is presented in chapter 14 by showing the conceptual architecture meets the requirements as derived from E-Gas’s business problem. The chapter also reflects on the thesis-work by answering the research questions.

Chapter 15 on page 71 concludes the thesis and looks ahead at future development and application of the architecture.

In the appendix, chapter C sketches theoretical background from relevant scientific fields used as a reference and way of thinking during the project. This chapter explores a set of requirements for the architecture based on theoretical disciplines, which are used in the next chapter to confront the conceptualized architecture with the theory.

Chapter C.4 of the appendix continues with the theory from chapter C to highlight interesting notions which lie at intersections of the various scientific disciplines and re-evaluates the theoretical requirements for the architecture based on these new insights. After that it presents a complete set of theoretical insights for the architecture, which are used to confront the architecture with the theory in the final section of this chapter.
Chapter 3

Conceptualization of an architecture

This chapter describes the conceptualization of an architecture based on the requirements for a solution presented in the first section. In the second section briefly looks at a traditional solution and argues that it is not feasible for the E-Gas case. Section 3.3 on page 14 introduces the carwash-example which is used to conceptualize the architecture, found in section 3.4 on page 15.

3.1 Requirements for a solution from the interview

The following list of requirements for a solution to the business challenges of E-Gas has been derived from the interview with one of the E-Gas’s senior managers. The means-ends diagram which was made as a result of this interview is found in appendix B on page 79.

- **make devices more visible to the net**: Being able to address all (legacy) devices over the net through a single interface will reduce the need for multiple systems and reduce number of site visits.

- **use of open standards**: Open standards prevent vendor lock-in and the need for multiple systems, since systems can be integrated.

- **remote software distribution**: Distribution of software to 23,000 different sites is a time- and cost-intensive process. Remote distribution of software can significantly reduce the number of site visits, as well as the time needed to update software on various systems.

- **remote monitoring and administration**: Remote monitoring of systems allows various actors to be notified of events that require their attention, without being present at the location of the event. Remote monitoring also incorporates alarming through various channels, such as e-mail, pager-alerts, etc.

- **solution flexibility, extendibility**: The solution should be flexible in a sense that it can quickly adapt to new actors and their new questions, as well
as connect with different systems. Apart from flexible, the solution should also be extendible to support new functionality in the future.

- collect sales data for datamining and trend analysis: The solution should support bringing information from multiple locations in a distributed environment to a single location for analysis.

The requirements as just presented invite to speculate about a feasible solution. The next section makes a suggestion.

### 3.2 Suggested solution; an ERP-system

Based on the previous requirements, many IT-consultants today, would recommend a centralized ERP\(^1\)-system for storing the business state and business rules, in conjunction with a large data-warehouse for storing all the business events, to be used for trend-analysis and derivation of (longitudinal) management information. Centralized solutions as large enterprise ERP-systems have been implemented for almost a decade now and day to day practice learns that their centralized concept leads to system inflexibility. Apart from that, the concept of storing all information in a single data repository, in practice, makes retrieving the right information for a specific actor often like searching for a needle in a haystack, apart from the fact that mining the data in a centralized repository is a very time-consuming process.

In the E-Gas case, implementing an ERP-system, together with a data-warehouse, will only solve part of the company’s problems. An ERP-system cannot meet the company’s demand for flexibility, multi-actor support and its geographical requirements, without bending the centralized concept of enterprise computing to inflexible, specifically configured connections between all kinds of different sub-systems. ERP-systems, as a centralized solution, do not do just to the geographic dispersal of 23,000 different gas stations, nor to the wide range of actors and their constantly changing information-need.

As an ERP-system does not meet all the requirements for a solution for E-Gas and proves to be an sub-optimal solution, this thesis explores new concepts, which better meet the requirements of organizations with many distributed business locations and need for a flexible solution.

### 3.3 E-Gas case example

To be able to conceptualize an architecture which meets the requirements as presented in section 3.1, one particular example from the E-Gas station case is used throughout this thesis. Let us recall the previous example, depicted in figure 3.1:

To ensure operability of the carwash at the gas station, the level of various fluids (soap level, wax level, etc.) in the machine have to be within specs. In this example, the clerk working behind the counter of the gas station needs

\(^1\)Enterprise Resource Planning
to be informed when the soap level drops below a certain level. If so, he/she needs to refill the soap from the soap stock which is available in the gas station. Whenever the clerk uses some of the soap from the soap stock to refill the carwash, the amount of soap in the soap stock diminishes. Whenever the soap stock falls below a certain level, the procurement department at the oil company’s headquarters needs to be notified, so they can re-order soap and to have the stock refilled with the next scheduled delivery of supplies.

The various devices on the bottom of the picture (carwash and gas station) have properties which are relevant to the particular actors seen at the top of the figure. The main question is how the information available in devices is going to reach an actor present at an entirely different location.

The next section conceptualizes an architecture based on the example as just presented and the requirements as set by E-Gas management.

3.4 Conceptualization of the architecture

This section conceptualizes an architecture based on the example of the carwash installation as presented in the previous section. Over the following subsections, the concepts are added to the solution, until in the end the entire architecture has been presented. Since the figures depicting the various concepts are found with the particular subsection describing the respective concept, captions for the figures in the following subsections have been omitted.

3.4.1 Actors have a need for information from devices

As a starting point, it is recognized that actors in the business environment have a need for information available in devices at various locations to support their work, to allow them to control processes and make decisions. Figure 3.4.1 depicts actors with an information-need and devices containing information, separated by a huge white gap. Currently information cannot not reach the actors in any way.

Figure 3.1: Graphical overview of E-Gas carwash example
3.4.2 Devices have a virtual representation

In figure 3.4.2 the concept of the virtual device is introduced. Hiding technical complexity and specific implementation, this concept introduces a common interface for access to information available in devices and wrapped to a virtual representation. Still, information does not reach actors.

3.4.3 Actors have a virtual representation

The virtual actor represents the physical actor in the architecture. Events which are received when the physical actor is off-line are stored, as well as granted rights and actor-specific preferences. Figure 3.4.3 shows the virtual representations of various actors which allows them to be a part of the network-interaction. Now that actors and virtual devices are both brought into the architecture, let us look at ways to extract information.
3.4.4 Information-extracting filters deliver relevant events to virtual actors

A filter is a component which extracts information from one or more virtual devices in the framework and sends the information to the subscribed actor. Figure 3.4.4 shows that special listening filters listen to the virtual gas station and virtual carwash and send business events to certain actors.

3.4.5 Access to system functionality is restricted by an administrator

In a distributed environment stretching many organizations, it is of importance that access to information and system functionality is restricted to authorized users. Figure 3.4.5 introduces the virtual administrator, who is the gateway to the physical administrator. The physical administrator is the one who grants
access to system functionality or information in his organizational domain, relevant both for systems administration, as for business administration.

3.4.6 An actor interacts with system functionality through a business rule

A business rule defines the specific system functionality for one or more virtual devices, which allows an actor to act on the system. Figure 3.4.6 shows the maintenance employee on certain carwash functionality, through sending a business action to a business rule. The maintenance employee might remotely reset the device.

3.4.7 Various GUI-panels allow actors to receive events anytime, anywhere

To support receipt of information by actors when they are on the road, or at a different location, the concept of the GUI-panel allows actors to interact with
their virtual representation through various means. The GUI-panels depicted in 3.4.7 could represent a workstation, mobile phone, internet-kiosk or perhaps a pager.

3.5 An architecture as solution

As a summary to the introduction of the architecture, numbered items in the following list reflect the various concepts introduced to a solution for the E-Gas case. Figure 3.2 depicts a graphical representation of the numbered items presented.

1. Information generating (real or simulated) devices are wrapped to a virtual representation called a virtual device, making their relevant properties available within the architecture. Device-specific implementation is hidden from the architecture, abstracting from technical complexity and creating a single interface for addressing information throughout various organizations. As such, virtual devices could be connected to real, legacy devices, simulated devices, or any information source.

2. Actors interact with the system through their virtual representation called virtual actor. The virtual actor represents the physical actor in the architecture, so events which are received when the physical actor is off-line can be stored. Apart from that, the virtual actor stores granted rights and received business events and is able to process business events based on additional preferences set by the virtual actor.

3. Actors are notified of events through filters, to which they can subscribe. Introduction of special listening filters not only allow actors to query a device for a specific property-value, but also support actor-notification of relevant system-events, discarding all irrelevant information.
4. **Access to information available in virtual devices or particular system functionality is restricted by an administrator, which also has a virtual representation.** The concept of information-ownership and an information-gatekeeper who guards access to information or functionality available within his/her organizational domain allows organizations to share particular information while hiding other information.

5. **An Actor can control the system (one or more virtual devices) by sending a business action to a business rule.** The introduction of the business rule allows specific functionality to be brought into the architecture for single virtual devices, as well as for multiple virtual devices in a distributed environment.

6. **Actors are able to receive events while they are on the move by the concept the GUI-panel.** A GUI-panel might be a workstation, but also pager or mobile phone, equipped some sort of wireless access protocol. A loosely connected GUI-panel disconnects the virtual actor-state from the way information is presented.

Virtual representations of the various information generating devices found throughout the organization are brought into the architecture where information extracting filters listen to these properties. As soon as a property-change is relevant to one or more actors, a business event is sent out to the actor’s virtual representation. Based on the business event the actor can act or maybe decide that no immediate action is eminent.

![Figure 3.2: E-Gas Station instantiation](image)

To stay informed of the conditions that require their action in the carwash example, as mentioned in section 3.3 on page 14, respective actors subscribe to a particular **ThresholdFilter** which listens to the relevant property and sends a

---

*2 invoke an action which causes a state change*
message as soon as the value drops below the indicated threshold. A graphical overview of the example emphasizing the various types of filters and components is found in figure 3.5. The figure also illustrates the architecture's concepts are equipped to extract information from both real, as simulated carwashes.

Figure 3.3: Architecture instantiation for the carwash example
Chapter 4

Detailed overview of the architecture

This chapter gives a detailed overview of the various concepts presented in the previous chapter, which make up the proposed architecture and which are described in the following chapters. The first section focuses on the interaction between the various components of the architecture, while the second section highlights some of the properties of the foundation on which the various components were built.

4.1 Architecture interaction

While figure 4.1 shows a generic instantiation of the components in the architecture, figure 4.2 on page 23 shows the architecture’s interactions, by depicting how filters are updated by virtual devices on state changes and how business events are then forwarded to virtual actors. All components in the architecture extend functionality from a generic component called NaradComponent\(^1\) which is ‘loaded’ with functional sub-components. The sub-components offer functionality like a-synchronous communication and security through the use of cryptography.

4.2 Anarchic topology based on a standard component

The previous section gave an overview of the possible interactions between various components in the architecture. This description focused on the semantics and more logical translation. How these components relate to each other in ever changing networks in the most efficient and flexible way, formed the basis for research on distributed anarchic topologies. This research is presented as *Distributed components in a visualization environment* by [Jacobs, 2001].

\(^1\) Narad, name of the the Indian God of messaging was using during the project as a working-title
Figure 4.1: A generic instantiation of the architecture

Figure 4.2: Possible architecture interaction
Figure 4.3 gives an overview of the resulting topology of this research and therefore of the topology of the architecture. The main characteristics are:

- The topology of the architecture supports multiple organizations or federations, each possibly consisting of multiple groups. Each group or federation has clear boundaries, but connections between them may be established by components.

- Since the topology is dynamic, one must assume redundant components and complex but partly hidden relations in providing business logic to other components.

- Components or services within the architecture are arranged in a chaotic order which implies the creation, movement and deletion of components and their mutual relations.

- The topology is based on a peer-to-peer model focusing on the equal base on which components communicate. There are no conceptual differences between components in the perspective of the network topology.

- The lack of centralized management implies that every component must be able to bootstrap itself, provide intelligence to its surrounding environment and must be able to lookup its neighbors.

- The outline color of the components points to the basic interface each component in the architecture extends. These basic components serve as a module providing most of the above abilities.

For more information on the anarchic Jini-components that were the foundation for the components in this thesis, see [Jacobs, 2001]
4.3 Various components in the architecture

As presented, the architecture consists of components and message objects which are sent between these components. This section first describes components, after that the messages which are sent between the components.

4.3.1 Architecture components

Figure 4.1 on page 23 presents the core-components of the architecture. Real devices or simulated devices are wrapped to a virtual representation called a virtual device. Actors have a virtual representation in the architecture called virtual actor, which allows them to interact with the system and store persistent information whenever the physical actor is not present. Actors can control\(^2\) one or more virtual devices by sending a business action to a business rule. The business rule takes care of the invocations with specific virtual devices/real devices. When a certain condition is met that the actor is interested in, the filter notifies the virtual actor by sending out a business event. Figure 4.1 on page 23 shows how filters are updated by virtual devices on state changes and how the business events are forwarded to virtual actors.

The following components are part of the architecture-core:

- Virtual Device
- Filter
- Business Rule
- Virtual Actor
- Virtual Administrator

The following components are not part of the architecture-core, but interact with components in the architecture. These components are also covered in the following chapters:

- Real Device
- GUI-panel, Actor panel
- GUI-panel, Administrator panel

As can be seen from figure 4.4 on page 26 all components in the architecture extend functionality from a basic component which is based on \textit{Jini}-technology\(^3\). The basic component contains the functional sub-components for the particular role the component fulfills. Sub-components offer functionality like a-synchronous communication through JMS\(^4\) and security through the use of cryptography. For more information on the basic component, see [Jacobs, 2001].

\(^2\) invoke an action which causes a state change

\(^3\) for more information on \textit{Jini}-technology, see \url{www.jini.org}

\(^4\) Java Messaging System
4.3.2 Message objects

The core components in the architecture\(^5\) are able to communicate by exchanging messages. A virtual actor, for instance, is able to subscribe to a Filter by sending it a signed 'Request For Rights'. See appendix E.1 on page 103 for a description of the non-conventional, distributed paradigm for administration, through requesting rights for certain system functionality. An actor sends a signed 'Request For Rights' to an instantiated filter which places a listener with a virtual device. A filter communicates with a virtual actor by sending it a business event. The virtual actor can inform the physical actor through a notification to a dedicated user interface. Figure 4.5 contains the hierarchy for the message objects.

The following moving objects are part of the architecture:

- Request For Rights
- Ticket (digitally signed Request For Rights)
- Business Action
- Business Event

---

\(^5\)The core components are Virtual actor, Virtual Administrator, Virtual Device, Filter and Business Rule
Properties, functionality and design considerations of the dynamic components and message objects are described in the following chapters based on the set of generic design questions presented in the last section of this chapter.

4.4 Generic design questions for architecture components

Each of the components in the architecture performs specific tasks and has specific functionality to ensure the architecture meet its requirements. For each of the components the following generic questions will be answered to be make sure the component is complete in design and function. These questions can be divided in three different categories which together cover all relevant aspects with respect to a component’s design; its functionality, life-cycle and its interaction with other components.

1. Component functionality
   - What specific functionality does the component add to the architecture?
   - Who makes the important decisions about the components functionality?

2. Component life-cycle (instantiation, modification, destruction)
   - Who instantiates the component?
   - When is the component instantiated?
   - Where does the component live? (In what domain does the component live? Does it live close to one component or close to another?)
   - How is a component modified?
   - When is the component destroyed?

3. Component interaction (authorization)
   - Who is allowed to interact with the component?
   - What other components does this component interact with directly?

Each of the architecture components, which are covered in the next chapters, is put in the context of the E-Gas case with an example. Many of the examples have been taken from the reference implementation in the Java-language.
Chapter 5

Generic building blocks and sub-components

This chapter describes some properties of the basic components on which the framework was built, and touches on implementation-specific components used to reduce complexity in the development-phase. It covers the Basic Component, Narad Component, Framework Message and Messaging sub-component in four different sections.

5.1 Basic Component

The basic component is the foundation on which all the different components in the architecture are built. The basic component offers functionality for different various sub-components to be added to the component repository. A basic component can fetch the sub-components it needs from its repository or search for particular functionality on the network. The description for the basic component presented in the following subsection is limited. For more on the basic component see [Jacobs, 2001].

5.1.1 A Basic Component’s passport

A Basic Component is uniquely identified by its Passport which contains information about what type of component its holder is, what its particular name it has and to what organizational domain it belongs. The passport also contains a component’s public key so other components can encrypt messages to the particular component and verify that the component has signed certain messages. The passport is a component which contains the following attributes:

- **Name**: The name for the particular component.
- **Type**: The type of component, e.g. VirtualActor, a VirtualDevice of some sort, etc.
- **Domain**: The organizational domain the component lives in.
• **PublicKey**: A component’s public key so other components can send it encrypted messages and verify signed messages.

• **Administrator-passport**: The passport object for the component’s administrator, used to verify granted rights and find the relevant administrator for a component.

Within one domain\(^1\) there can be only one component with the same name of a particular group. The Administrator attribute is the passport of the component’s Administrator, which allows an actor, for instance, to see which component he should ask rights to interact with a particular Virtual Device.

### 5.1.2 A Basic Component’s sub-components

All the non-moving components in the architecture extend from a generic component (the basic component) which can be filled with sub-components to allow specific functionality. The following list summarizes sub-component functionality added to a basic component for a virtual device.

• **Core component**: This component contains the virtual device state which is specific for a particular virtual device. Since the virtual device extends from the basic component, it can be loaded with sub-components.

• **Passport**: A component is identified uniquely in the architecture by its passport, which is part of a basic component when it is instantiated.

• **Communication-component**: Send and receive messages synchronous, asynchronous. Any basic component can receive messages through its JMS-communication component. Components of the same type (within a domain) can also listen to a common message bus. Due to the fact that the messaging-component receives the incoming moving components its structure defines how incoming messages are routed to the different functional components in the network.

• **Authorization-component**: Not every actor may be allowed to request a service from a component. The authorization-module restricts is called by the communicating component in case of an incoming message and routes access to core and sub-component functionality.

• **Operational components**: This components define the ‘core-business’ of the specific component and how it deviates from other components. The virtual device components allows functionality for generic querying of property-values, for instance, as well as setting property-values.

For more information on different sub-components please refer to the documented Java-source code, which is available at [www.sk.tbm.tudelft.nl/narad](http://www.sk.tbm.tudelft.nl/narad).

---

\(^1\)see definition A.1 on page 77
5.2 NaradComponent

The NaradComponent was introduced as an extension of the basic component which offers an interface for calling the communicative and cryptographic functionality available as sub-components in a component’s repository. Introduction of the NaradComponentInterface significantly reduced the size of the source-code. Further description of the NaradComponent is not relevant for the storyline of the thesis. The interested reader is referred to the source-code available at www.sk.tbm.tudelft.nl/narad.

5.3 FrameworkMessage

All moving components in the architecture extend functionality from an abstract moving component called FrameworkMessage (see figure 4.5 on page 26). The FrameworkMessage is best compared with an envelope which can contains basic functionality for identifying, sending and receiving messages in the architecture. The FrameworkMessage has the following properties:

- a unique identifier: An identifier which uniquely identifies a message
- a sender: The component which has sent the message
- a receiver: The component which the message should be delivered to
- requestType: A String which indicates the kind of request/action packed in the message

Both the sender as the receiver are referenced to by their Passport-component. Based on the previous standard-properties components that extend the FrameworkMessage’s functionality can add their own specific attributes and methods.

5.4 Messaging component

This section describes the generic messaging component which allows each component in the architecture to send, receive and process JMS-messages\(^2\).

5.4.1 Component interface

Security and the need for a flexible solution with respect to authentication disallow one component to call methods on another component using RMI\(^3\). The only way to contact one of the major components in the architecture is to send it an asynchronous (authorized) message. As a consequence, a component’s interface doesn’t look very interesting. The message a component receives is processed in the messaging component which calls specific processing functionality.

---

\(^2\)JMS, short for Java Messaging System, is a system for asynchronous delivery of messages through a messaging server. JMS is the de-facto standard for sending XML-messages in B2B-solutions.

\(^3\)Remote Message Invocation, the Java implementation for Remote Procedure Calls
Whereas normally developers would discuss the interface a component offers to other components, in this architecture the discussion should be about the processing code of the messaging component, and how this external interface is translated into the internal interface. See appendix E.2 on page 105 for a description of the difference between synchronous and a-synchronous communication.

5.4.2 One messaging sub-component for all components

In the current Java-implementation, all major components use the same messaging sub-component. No matter whether an actor needs to process a received business, an administrator needs to process an incoming Request For Rights or a filter processes an signed ticket, it can all be handled by one and the same Java-class. Using the same messaging-component has the advantage that if necessary, a virtual actor can at some time start to behave like an administrator for Requests for Rights concerning information that he/she owns, without security has to compromised, since the rights-model is unbroken.

A diagram which is added as figure E.2 on page 107 of the appendix shows how an incoming FrameworkMessage is parsed by the messaging-component.

From the next chapter on a detailed coverage of the various components in the architecture is given.
Chapter 6

Real Device

Real devices are the devices that generate or pass through the actual information an actor is interested in. A real device can be many things; any electronic device or the output of a measuring device, an ERP-system, large legacy database, etc. Anything that has changing properties which an actor could be interested in is a potential real device, which could be wrapped into a virtual representation. Since the real device is not an official part of the architecture, how the component communicates with its virtual representation is not defined. In the reference implementation, one of the virtual devices’s is updated through XML-messages it receives from its real counterpart. For more information see the source-code which is available at www.sk.tbm.tudelft.nl/narad.

6.1 Answers to generic design questions

The following answers to the generic design questions give insight in a real device:

1. Component functionality
   - *What specific functionality does the component add to the architecture?* Real devices are the source for information actors are interested in. The relevant information is presented within the architecture in a real device’s virtual representation.
   - *Who makes the important decisions about the components functionality?* The engineers who initially designed the real-device make the design-choices for a real-device’s functionality. It is unlikely that a real-device will be modified because an actor in the architecture needs different information.

2. Component life-cycle
   - *Who instantiates the component?* The answer to this question lies outside the boundaries of the architecture. The real device is assumed to be available to the architecture the moment its virtual counterpart is initiated.
Virtual Device

Real Device

Figure 6.1: Connection between a real device and a virtual device

- **When is the component instantiated?** Since a real device is located at the border of the architecture it is considered to be already alive the moment its virtual representation is instantiated.

- **Where does the component live?** Since a real-device can be of any sort, the place where it lives is not limited. A real device lives outside the organizational domain its virtual representation belongs to.

- **How is a component modified?** This differs on a per real device basis, depending on the defined business rules.

- **When is the component destroyed?** A real device can only be destroyed through external influences.

3. Component interaction

- **Who is allowed to interact with the component?** Only a real device’s virtual representation communicates with a real device. The real device’s state is changed by external influences, though, which is reflected in its virtual representation.

- **What other components does this component interact with directly?** Either through push or pull a real device feeds its virtual representation with information regarding changes in its properties. The real-device can receive method-calls initiated by a business event received by its virtual representation. No other component except the virtual representation in the architecture is allowed to contact or even see the real-device. The virtual representation is updated by a real device’s state, *never* vice versa. Loose coupling between a real device and its virtual counterpart prevents the virtual representation from disappearing the moment a real-device breaks down. Depending on particular implementation, one of the virtual device’s properties might indicate the fact that contact with the real device is lost.

6.2 **Link between a virtual device and a real device**

It is of considerable importance to dimension the necessary interaction between a real device and its virtual counterpart. A real device that changes its state once
a day doesn’t need to be polled every 10 seconds for a state change. Preferable a real device should be forward state changes to its virtual counterparts, but this is not always possible. The following characteristics are important:

- # updates / second
- when to choose a push or when a pull implementation

There are two different possible means for virtual device-real device interaction:

- **Information Push**: A real device (wrapper) pushes `propertyChanges` to its virtual representation which has subscribes to these events.
- **Information Pull**: A virtual device contacts its real counterpart with regular intervals to see if a relevant property has changed

A situation where a real device pushes information to its virtual representation is preferable since it minimizes interactional overhead, except when state changes occur very often and it is expected that the filters will not be able to process the detailed time resolution of the messages.

Wrapping a (single threaded) legacy device requires special precautions to prevent direct method calls to the legacy device which will crash the application. Instead of direct system calls a queue at the wrapper is used to line up communication directed at the specific device. When available the device itself polls the queue to see whether there are any new requests. Using a similar queue in the wrapper the legacy device is able to communicate to the outside world. See [Lang, 2001] for an elaborate description of wrapper components.

### 6.3 Wrapped devices

Two different wrapped devices can be distinguished; wrapped real devices, and wrapped simulations.

#### 6.3.1 Wrapped real devices

The qualification 'Real Device' is a label which enables actors and developers to discuss which devices should be part of the architecture, which properties its virtual representation should offer, and where/how the connection between a real device and its virtual representation should be made. This are some examples of devices that could possible be wrapped:

- an embedded system addressable through PLC
- the engine of a high-tech car addressable through internet-technology
- a web-server which is gateway to a web portal, or
- the database server on which it relies
- the underlying database of an ERP-system
• certain tables of a CRM\textsuperscript{1}-database

Any device that generates or contains information which would be relevant for an actor in a business case is a good candidate to be wrapped into a virtual representation. Examples like the ones previously mentioned are only useful in the proper context of why they should be wrapped, and what information they should make available.

6.3.2 Simulated devices

For testing purposes or to mimic a real world situation without really making claims to expensive resources, a simulated version of a real device can be used. In a number of cases it can be useful to replace a real device with a simulated device, such as:

• training
• decision support / what-if analysis
• visualization of certain processes

An elaborate description of connecting simulations to the architecture can be found in [Lang, 2001].

6.4 Real Devices in E-Gas Station

In the E-Gas case, the oil company management wants the devices in the various gas stations around the country to be ‘net-enabled’ to be able to process all kinds of information on the state of the devices. Various actors in the E-Gas case should have access to different kinds of information from the following devices to base their decisions on;

• the gas dispenser in a gas station
• the gas pump which pumps the gas to the dispensers from the underground
• the car wash in a gas station
• the refrigerator in the gas station store
• the point of sales (and its transactions) in the gas station store
• certain parts of the gas station back-office

Although the way of connecting these devices to the architecture will be different for each of the particular devices and based on the particular (legacy) implementation of the device, the moment the devices are connected to the architecture, they are all addressable in exactly the same way. The carwash (in particular) is described more in detail as a virtual device in the next chapter.

\textsuperscript{1}Customer Relationship Management
Chapter 7

Virtual Device

This chapter describes the design considerations, properties and functionality of a virtual device. The Virtual Device provides an homogenous layer of access to information available in different kind of devices. The virtual device’s information is available in its properties, which can be either numeric or in text-form.

7.1 Answers to generic design questions

The following answers to the generic design questions give insight in a virtual device:

1. Component functionality

   - *What specific functionality does the component add to the architecture?* A virtual device unlocks information which is available in all kinds of real-devices outside the architecture. In the architecture, an abstract virtual device does not offer any other functionality in its interface other than the possibility for another component to place (an authorized) listener. An actor who wishes to interact with the system never sends messages to a virtual device but always to a filter or a business rule, one of the interactive components. Case-specific instances of a virtual device will carry additional functionality for receiving and processing messages from a business rule. The introduction of a virtual representation prevents direct interaction with the wrapped component. This extra layer shields the real device from potential floods of requests which could interfere with a device’s operations or lead to congestion and unacceptably long response times. Some virtual devices might contain some logging information on their previous states (e.g. a sliding register) to be able to answer questions regarding its history. Logging functionality should be dimensioned carefully to prevent attributing resources to unnecessary storage of information.

   - *Who makes the important decisions about the components functionality?* Which properties are relevant for a virtual device is decided by
the total sum of actors’ need for information. Eventually the decision what information to make available is virtual device’s administrator.

2. Component life-cycle

- **Who instantiates the component?** A virtual device is instantiated by its administrator.
- **When is the component instantiated?** A virtual device is instantiated the moment the architecture is brought to life. In special occasions, an actor might request a new virtual device to be defined and instantiated.
- **Where does the component live?** A virtual device lives an organizational domain, on a server within the influence-sphere of the administrator. Preferably the distance between a real device and its virtual representation should be minimal, to prevent communication lags.
- **How is a component modified?** A virtual device can be modified by instantiating a new virtual device with the new functionality and after that taking the original virtual device down.
- **When is the component destroyed?** A virtual device could be destroyed if there is no need for its information from any actor over a longer amount of time. Whether this is desirable depends mainly on the particular business case.

3. Component interaction

- **Who is allowed to interact with the component?** Only a filter with the right credentials is allowed to place a listener on a virtual device and receive PropertyChangeEvent. A business rule can contact a virtual device to execute some kind of action in its real counterpart.
- **What components does this component interact with directly?** A virtual device interacts with filters, business rules and its real device.

7.2 Connection between virtual and real device

**What happens if the connection between a virtual device and its real device is lost?** A virtual device-instance is not destroyed the moment that contact is lost with its real device, but merely put in a waiting state. Based on the particular implementation it might be feasible to put the virtual device in a special state indicating a problem if it is not updated for a certain amount of time. In this respect, the virtual representation in the architecture deviates from the real world to present additional information, which might be useful for the actor monitoring a device’s operational 'heartbeat'.

7.3 What information should be made available?

As explained, a wide range of devices can be wrapped to a virtual representation called a virtual device. What properties should be made available to actors
in the system depends on the information-need of the actors that will be interacting with the virtual device. For example, a virtual device should which is a representation of a carwash does not need to have a property 'nameOfCurrentCustomer', if no actor is ever interested in which customer the carwash is currently helping. Discussion about the relevant properties of a virtual device should always be driven by the properties an actor requires to get an answer to his/her questions\(^1\). The starting issue for the discussion about a virtual device’s relevant properties is: What kind of information does an actor need to support his or her work?

In the current implementation of the architecture, a virtual device has a property of a certain type, either a text-string or a numeric property. Although, in some cases, it may be useful to see some properties as a virtual device on their own, for reasons of minimizing complexity this was not an option on the first implementation of the architecture. More research into this topic is a part of the recommendations.

In the current implementation a virtual device’s properties can be of the following type:

- string\(^2\)
- numeric\(^3\)

### 7.4 Virtual devices in E-Gas Station

The following examples show some of the virtual devices from the E-Gas Station-case and their properties. Which properties are relevant to which actor exactly, save some simple example, lies beyond the scope of this thesis.

- **Gas Pump**
  - gasPumpState (string)

- **Carwash**
  - soap level (numeric)
  - wax level (numeric)
  - airPressureLevel (numeric)
  - waterPressureLevel (numeric)
  - saleRinseAgentLevel (numeric)
  - program (string) - the string *program* indicates which program the carwash installation is running
  - state (string) - the string *state* indicates the current state of the carwash

---

\(^1\)the questions relevant to support his/her work and process of decision making
\(^2\)In the *Java*-language the *String*-class is used
\(^3\)In the *Java*-language the *Double*-class is used
A valuable lesson from wrapping a device is that it is important for send objective information in an event, placed in a normative context, to allow an actor to decide using other formalized and non-formalized information as a part of his/her decisionmaking process. Sending just normative information doesn’t allow an actor to make a real decision since the only decision he/she can make is to either follow a suggestion or discard it. For a description of this issue, see appendix F.2 on page 110.
Chapter 8

Virtual Actor

This chapter describes the design considerations, properties and functionality of a virtual actor. The virtual actor represents the physical actor in the architecture, stores granted rights and received business events and is able to process business events based on additional preferences set by the virtual actor. The virtual actor component receives incoming communication after which it might be forwarded to the appropriate means of display. The concept of a virtual actor enables the introduction of a stateless display component which can log in to virtual representation. The virtual actor component is responsible for keeping the actor-state.

8.1 Answers to generic design questions

The following answers to the generic design questions give insight in a virtual actor:

1. Component functionality

   - *What specific functionality does the component add to the architecture?* The virtual representation of the actor in the architecture stores the actor-state and contains functionality for an physical actor to interact with filters and business rules.

   - *Who makes the important decisions about the components functionality?* Standard functionality for the virtual actor is available in the reference implementation, but actors can always add extra functionality to their own virtual actor, for instance by adding rules how incoming business events should be processed. New functionality like this can be added without authorization from a dedicated system administrator, since the actor is allowed to make its own decisions about its virtual representation.

2. Component life-cycle

   - *Who instantiates the component?* A virtual actor is instantiated by its physical counterpart.
• When is the component instantiated? The component is instantiated the first time an actor wants to participate in the architecture.

• Where does the component live? It is not really relevant where a virtual actor lives, as long as it is a location, computer system, trusted by the physical actor. It is clear however that a virtual actor which lives on a system not closely connected to the system which it is interacting with might encounter communication lagging due to network traffic jams.

• How is a component modified? A new subcomponent can easily be placed and an old one removed without interfering with standard component operations.

• When is the component destroyed? A virtual actor will generally remain persistent on the network after the physical actor logs out of the architecture to keep listening for events that might interesting for the physical actor and present them when the actor logs on again. For very important events an actor might be notified through a medium like a pager or the SMS\(^1\) of a digital cellular phone.

3. Component interaction

• Who is allowed to interact with the component? Only a physical actor is allowed to interact with its virtual representation through its actor panel. Any other component who wishes to interact with this component needs to ask permission to the component’s administrator, which, in this case is the physical actor.

• What other components does this component interact with directly? A virtual actor interacts with filters, business rules and one or more actor panels. The virtual actor does explicitly not communicate with other virtual actors. If necessary, two physical actors communicate through a different medium. The architecture in itself does not restrict the suggestion of certain actions to users with administrative authority. For reasons of inter-personal communication, it is recommended that physical actors share the majority of these thoughts though direct, non-formalized communication.

8.2 A new actor joins the architecture

Due to the web of trust which is suggested to establish credibility between actors\(^2\) it is not necessary that a virtual actor-component is instantiated by an administrator or even is part of the administrator’s organizational domain. Ask the following question: Which administrator can give a virtual actor which is not part of its organizational domain the right to exist? A physical actor who would like to interact with the system could download a kit for instantiating a new virtual actor from a trusted site. Once a new actor-component has been

---

1 Short Message Service
2 See appendix G.2 on page 116
instantiated, it cannot interfere with any system without the proper rights to interact with a particular system, for which rights have to be granted by a system administrator.

Virtual actors thus are instantiated by a real physical actor. Since an actor cannot interfere with the system without the proper rights on the system, there is no risk\(^3\) of malicious actors interfering with business operations. Using a networked PKI\(^4\) an organization signs the public key of actors they trust (employees, suppliers, etc.) and thus allow the actor to be a part of the architecture. Trusted actors might be default be allowed to query the system for available virtual devices, filters or business rules. It should be noted however that the current implementation does not prevent querying of organizational domains, since the architecture is built on \textit{Jini}-technology. In this respect security within \textit{Jini} is a problematic issue. The most logical solution to this problem would be to extend the basic \textit{Jini-LookupService} (Reggie) to an \textit{AuthenticatedLookupService}\(^5\).

\section*{8.3 Suggested virtual actor sub-components}

The following suggested sub-components could add relevant functionality for actors. A sub-component for:

- storing tickets which were granted. Although the administrator keeps a list of the rights he has distributed keeping the tickets is primarily an actor responsibility. In this case an actor wouldn’t have to contact the virtual administrator when logging in to a new session. He or she would still have the rights to re-subscribe to a filter.
- routing the most relevant of the incoming messages to the actor’s mobile phone or pager while keeping the rest in storage for later processing.
- processing incoming messages to aggregate information from various filters to derive information and make it available to other actors through a different virtual device.

\section*{8.4 Virtual actors in E-Gas Station}

From the E-Gas case it is clear that different actors have a different need for information.

One example of an actor in the E-Gas case is the \textit{gas station service clerk} behind the counter in the gas station. For business purposes it is important that the carwash in the gas station is operational. Several people in entire the E-Gas business case each take a bit of the responsibility for the carwash. While the employee with the company hired to take care of carwash maintenance sees

\footnotesize
\(^3\)not regarding potential software bugs, of course
\(^4\)Public Key Infrastructure
\(^5\)Which would contain functionality for denying unauthorized people rights to query an organizational domain or \textit{Jini}-group.
to it that the various electrical and mechanical components are replaced before they reach the end of their life-cycle, it is the service clerk’s task to see that the soap level of the carwash remains at an acceptable level. If the soap level drops below a certain threshold the clerk needs to be notified that he/she needs to refill the soap.

Appendix B.1 on page 79 gives a more elaborate overview for an overview of all the different actors in the E-Gas case. An in-depth analysis of the information-needs of the various actors in the E-Gas case lies without the boundaries of this thesis, considering its goal of presenting an architecture, not solving a particular gas station’s problems.
Chapter 9

Virtual Administrator

This chapter describes the function of the virtual administrator in the architecture. The virtual administrator is the gateway to the physical administrator who takes decisions relevant, both for systems as for business administration. Due to the new concept of distribution of rights for system functionality, described in detail in appendix E.1 on page 103, the physical administrator’s work is limited to saying "I grant", or "I deny" a particular request.

9.1 Answers to generic design questions

The following answers to the generic design questions give insight in the virtual administrator-component:

1. Component functionality

   - What specific functionality does the component add to the architecture? The virtual administrator distributes rights in the architecture by having the physical administrator say grant or deny to formalized requests for rights, submitted by actors in the architecture.

   - Who makes the important decisions about the components functionality? An administrator is its own administrator and has full saying about how its component is organized. Limited only by the physical administrator’s imagination, extra functionality can be added his/her virtual counterpart.

2. Component life-cycle

   - Who instantiates the component? The virtual administrator is instantiated by its physical counterpart.

   - When is the component instantiated? A virtual administrator is instantiated as soon as the architecture is brought to life. Since the component distributes rights in the architecture, its availability is of the utmost importance when other components are to be instantiated or when changes in the architecture are needed.
• *Where does the component live?* The virtual administrator generally lives in its own organizational domain, although this is not an explicit requirement for the distribution of rights. For reasons of minimizing communicative overhead and instantiation of filters and business rules, it is recommended however.

• *How is a component modified?* Extra functionality can be added to the virtual administrator if the physical administrator feels it is necessary. An example of extra functionality might be a database which contains rights that have already been handed out by the virtual administrator, which enables tickets to be re-issued without any human administrative interference.

• *When is the component destroyed?* A virtual administrator is needed as long as there are virtual devices available for which this administrator is responsible.

3. **Component interaction**

• *Who is allowed to interact with the component?* Every actor known to the architecture is allowed to send an administrator a request for rights. An actor must be known in some way to the architecture to in order for the administrator to be able to establish the actor’s credibility.

• *What other components does this component interact with directly?* An administrator can intervene directly by sending an out business actions to various components in the architecture. For reasons of keeping responsibilities separated an administrator only administers a part of the architecture and should not tap information from various virtual devices. If a certain physical administrator has an information-interest of him/her-self he/she is free to instantiate a virtual actor for this specific purpose.

### 9.2 Multiple administrators cooperating

The architecture is designed in such a way that it also allows actors to ask questions which can only be solved if information from different organizational domains is combined in some way. If information from more than one organizational domain is necessary to get an answer to a particular question, rights for the respective information from all the involved administrators is required. Whether or not the administrator feels the need to limit *how* the information from a virtual device can be used, the filter could be made to live in a trusted third-party environment.

Figure F.3 on page 112 found in the appendix shows the initialization protocol for a filter which uses information from different organizational domains to notify an actor of a certain condition in the system.
9.3 Technical Administrator

Depending on the particular business case and its requirements, it might be necessary to monitor whether the connection between a real or simulated device and its virtual counterpart is still active, especially considering the loose coupling of these components. Inside the virtual device a special property could indicate whether the connection between the virtual device and its real counterpart is still within specs\(^1\). If not, a monitoring filter would send an event to a technical administrator, responsible for the architecture 'health', to indicate his/her attention is required. It is emphasized here that the technical administration of keeping the architecture 'healthy' is separated from administration regarding the distribution of user rights. Implementation of the technical administrator and more in particular, required changes to virtual device implementation is a recommendation for further development.

9.4 Virtual Administrators in E-Gas Station

In the E-Gas case, only one administrator was introduced to establish the proof of concept for the new administrative paradigm, described in detail in appendix E.1 on page 103. The administrator in the E-Gas demo grants incoming requests for subscription from the various actors involved in the business case.

\(^1\)a specific interval-timer for each real device - virtual device combinations
Chapter 10

The GUI Panel and its components

This chapter describes the design considerations, properties and functionality of the GUI-panels. A GUI-panel can consist of components with both displaying and controlling functionality. New components can be added to the panel in the fly, if necessary. Since the design requirements for both the the Actor Panel (the GUI Panel for the actor) as the Administrator Panel (the GUI Panel for the Administrator) show a great deal of similarity, the answers to the generic design questions are only answered for one of the two; the Actor panel.

10.1 Answers to generic design questions

The following answers the generic design questions give insight in the actor panel:

1. Component functionality

   - *What specific functionality does the component add to the architecture?* An actor panel allows a physical actor to interact with the architecture. He/she can request new rights, issue business actions or modify the way business events he/she receives are processed.

   - *Who makes the important decisions about the components functionality?* The physical actor who uses a standard actor panel might sooner or later feel the need for additional or different functionality.

2. Component life-cycle

   - *Who instantiates the component?* An actor panel is instantiated by the physical actor who wishes to interact with the system through its virtual representation.

   - *When is the component instantiated?* An actor panel is instantiated whenever a physical actor logs on to the system.

   - *Where does the component live?* Actor panels can exist in many forms and shapes. Common actor panels will live on a physical actor’s
workstation, more specific actor panels might live on a mobile phone or PDA\(^1\).

- *How is a component modified?* The actor panel can be modified by the physical actor by replacing the instantiated panel components.
- *When is the component destroyed?* An actor panel is destroyed as soon as an actor logs out of the architecture. Depending on implementation a display panel component of some sort could remain alive on an actor’s mobile phone.

3. Component interaction

- *Who is allowed to interact with the component?* The virtual actor component is the only component that is allowed to interact with the virtual component.
- *What other components does this component interact with directly?* An actor panel only interacts with an actor’s virtual representation.

### 10.2 Two types of panel components

In a control or display panel, a distinction is made between a component for receiving business events, called a display panel component and a component for sending business actions to a virtual device, called a control panel component. An actor-specific panel might contain various components of either sort, just as the actor requires. When an actor panel contains just display panel components it is called a display panel. If an actor can also issue business actions using the panel it is called a control panel.

The amount of functionality that should be available in a display panel is limited by the functionality of the displaying device. A workstation-based actor panel will contain components for complex interactions with the system, while a display panel on a mobile phone might only contain one a panel component for display of the most important business events. The evident design choice should also be based on the available amount of bandwidth.

#### 10.2.1 Display Panel Component

One display panel component might be a list of incoming business events, another might be a dial or counter which represents the current value of a certain variable. In this context, appendix F.3 on page 111 on whether a filter sends data, information or knowledge to the actor is relevant.

#### 10.2.2 Control Panel Component

An example of a control panel component might be a combination of a slider bar with a button to send the new variable value through a business action to the virtual device. Once an actor has the rights to change a variable within certain range a slider-button-combination might be automatically generated.

\(^1\)Personal Digital Assistant
10.3 Actor panels in E-Gas Station

In the E-Gas case various kinds of actor panels are conceivable. While for some actors (like the procurement department at the oil company’s headquarters) an actor panel on a workstation would suffice, certain maintenance workers who are mostly in the field would be much better supported receiving notifications directly to their pager.

The E-Gas demo shows how an actor panel is loaded with various functionality, for instance for receiving business events, requesting rights and issuing business events. A more elaborate description of the actor panel that was developed for E-Gas is beyond the goal of this thesis, but the interested reader is invited to experiment with the source-code, which is available at www.sk.tbm.tudelft.nl/narad.
Chapter 11

Filter

A filter is a component which notifies an actor of the occurrence of a preset condition of one or more virtual device’s properties. All specific filters extend functionality from a generic filter, as is depicted in figure 11.1. When an actor subscribes to a filter, the filter places a listener with the virtual device for the particular property the actor is interested in. The relevant propertychange events the filter receives are evaluated to see whether there are any actors interested in this particular event or in the outcome of some computation or aggregation done on the new value passed on in the event. Both the business rule and the filter are an interactive component. This section describes the design considerations, properties and functionality of filters in the architecture.

11.1 Answers to generic design questions

The following answers the generic design questions give insight in the basic filter-component:

1. Component functionality

   • What specific functionality does the component add to the architecture? A filter listens to one or more virtual devices and notifies a virtual actor with a business event of the fact that a condition has occurred which the actor indicated interest in.

![Diagram of Filter Hierarchy](image-url)

Figure 11.1: Filter hierarchy
• **Who makes the important decisions about the components functionality?** A filter is usually instantiated from a standard pool of filters, trusted by the administrator of the relevant organizational domain. New filters can be added to this pool after administrator-review, initiated by an actor who needs additional filter functionality.

2. Component life-cycle

• **Who instantiates the component?** A filter is instantiated by the administrator of the organizational domain in which the virtual device it will subscribe to will live.

• **When is the component instantiated?** A filter will be instantiated by a virtual administrator of the virtual device’s domain it will subscribe to, as soon as there is a virtual actor in need of its functionality.

• **Where does the component live?** To minimize communicational overhead, a filter lives as close as possible to the virtual device(s) it interacts with, in a trusted organizational domain.

• **How is a component modified?** A filter is modified by destroying the old filter and instantiating a new filter. More complex online modification protocols are conceivable, but are a subject for further research.

• **When is the component destroyed?** A filter can be destroyed if after a certain amount of time no other component in the network requires its functionality. The specific time is dependent on the type of filter and where it is implemented.

3. Component interaction

• **Who is allowed to interact with the component?** A filter can perform functions for every actor that is allowed to either use the filter in combination with a virtual device, or subscribe to property change events from a virtual device.

• **What other components does this component interact with directly?** A filter receives tickets from a virtual actor, after which if subscribes to one or more virtual device’s property change events if necessary. If a pre-defined condition is met, of which an actor would like to be notified, it sends a business event to the virtual actor. If rights for a specific request are unavailable, it automatically sends a request for rights to the appropriate administrator.

A filter of a particular type can perform actions for all actors on all the virtual devices in its organizational domain. As stated, a filter listens to a virtual device for actor-defined state/variable changes in a virtual device and communicates these to the virtual actor that has subscribed to a certain condition of the virtual device. For performance issues, instantiating a new filter for each new subscription to a filter of a similar type is not feasible. If 5 different actors are

---

1 for a definition of organizational domain, see appendix A.1 on page 77
interested in thresholds on a particular virtual device only one instance of a threshold-filter within the particular organizational domain suffices to monitor this virtual device and send out business events to the virtual actors that have all set a different threshold.

An actor can communicate with a filter by sending it a Request For Rights to subscribe, or an altered self-signed Request For Rights to unsubscribe from the Filter.

11.2 Trusted Filters

Before filters can be used within a certain network domain they need to be reviewed and authorized by the administrator of the domain where the filters will reside. Although an actor could self-define particular filter functionality, before the filter will be instantiated by the administrator, it needs to be reviewed to see if it doesn’t contain any malicious code. For the administrator it is not relevant to see what in particular the filter does with the information it receives from the virtual device, unless the administrator feels the need to restrict how information can be used.

The use of untrusted filters\(^2\) in the architecture is not a feasible option since no organization would ever allow an untrusted filter to reside in its network-domain.

Trusted Filters can be instantiated from a pool of trusted filters with different functionality. A new organizational body is conceivable which will test a specific filter for the functionality that it is claimed to have. If the filter holds this functionality, it will be authenticated by this particular organization. Any organization which trusts this body will trust the filter without hesitation, which would mean the organization would not have to spend scarce resources to develop the filter by itself.

11.3 Filter location

Since a filter only informs actors of virtual device state-updates that are relevant to them (in many cases only a fraction of the actual state updates), a filter should reside as close to the virtual device it is listening to as possible.

When more actors, especially within the same network-domain, are interested in listening to a filter, network traffic can be minimized by sending state changes up to a filter that resides close to the actors and filter state changes to Business Events from there.

A special optimization agent is conceivable which permanently monitors component-interaction between virtual devices, filters and actors and optimizes the system by changing a filter’s location. The question how communication between filter and more virtual actors could be optimized by moving the filter location needs further research.

\(^2\)filters which are not reviewed by the administrator
11.4 Subscription to a Filter

Figure 11.2 on page 53 depicts the protocol for filter-instantiation and actor-subscription to a filter. An actor issues a formalized request which is sent from his/her virtual representation to the relevant virtual administrator. The administrator grants the request by digitally signing the request-object and returning it to the actor. When a filter is available, the actor forwards the ticket to the filter which places a listener with the appropriate virtual device’s property. The virtual device notifies the filter of a change in the relevant property and the filter sends out business event to the virtual actor.

11.5 Types of Filters

Different types of filters can be identified distinguished by how many administrators are responsible for virtual devices the filter wants to listen to:

- A filter listening to one virtual device in one organizational domain
- A filter listening to more virtual devices in different organizational domains

If an actor wants to subscribe to information from different virtual devices in different organizational domains, the filter initiation protocol is more complex. Every administrator of every virtual device involved has to grant rights for an actor (’s filter) to interact with the system\(^3\).

Another distinction between filters is made, based on whether a virtual device’s administrator wants to restrict use of the information which is available in the virtual device. Based on a possible limitation of use, the administrator might change the request for rights object before he signs it.

11.6 Different types of filters

This subsection gives some examples of a few basic filter components. Filters can be extended and used as building blocks for new (more complicated) filter components.

\(^3\)see page 112 and further
functionality. This section covers respectively, the State Change Filter, State Enter Filter and the Threshold filter.

11.6.1 State Change Filter

The State Change Filter is the simplest filter conceivable. It is a filter which doesn’t filter out any events but forwards all the state changes in a virtual device’s property as a business event to a virtual actor. It does add a certain shell of context to the original property change event, though.

11.6.2 State Enter Filter

A State Enter Filter listens whether one or more actor-defined states have been reached. In case of a gas station, a state change filter might listen to a legacy gas pump which (when broken) enters a state called ‘error’ with some error code. In an extended supply chain, an actor might be interested in the arrival of an airplane and wishes to be notified when the status of the plane changed to ‘landed’.

11.6.3 Threshold Filter

A threshold filter is a filter that monitors a numeric property of a virtual device and sends a business event when the numeric property crosses a certain threshold. Such a device could, for instance, be a refrigerator, with refrigerator-temperature as specific variable. If the temperature goes out of some pre-defined bounds (because the cooling system fails the temperature rises above say 8 degrees Celsius) a message, called a business event, is sent to the appropriate actor, in this case a maintenance person. In the demo, a two-sided threshold filter has been implemented, where a distinction has been made between a filter which sends business events if a value outside the filter-band is perceived, or one that sends out business events if virtual device’s numeric property enters a value inside the filter-band. More on the difference between inbound and outbound threshold-filters is found in the following paragraphs.

inbound/outbound

When setting the filter properties (the threshold bounds), it is important for an actor to indicate whether he/she wants to be notified when the value of the property he/she is interested in becomes larger or smaller than the particular threshold (for a single sided filter), or is within certain bounds or out of these bounds (for a double sided filter).

Threshold filter properties

Figure 11.3 on page 55 explains the possible conditions for an event based on the type of threshold filter. Clearly indicated on the line that depicts possible values for a property are the upper and lower boundary of a threshold filter. An

---

4the band between the lower and upper threshold-value
actor must indicate whether he/she wants to be notified when the propertyValue meets the following condition; situation $A$:

\[
\text{lowerBoundary} < \text{propertyValue} \quad \text{AND} \quad \text{propertyValue} < \text{upperBoundary} \tag{11.1}
\]

or that he/she wants to be notified when the following condition is met; situation $B$:

\[
\text{propertyValue} < \text{lowerBoundary} \quad \text{OR} \quad \text{propertyValue} > \text{upperBoundary} \tag{11.2}
\]

When a threshold filter sends out a business event in case of condition 11.1 it is called an inbound threshold filter. When a threshold filter sends out a business event in case of condition 11.2 it is called an outbound threshold filter.

### 11.7 Filters in E-Gas Station

In the E-Gas demo the filters mentioned in the previous section are instantiated to provide the various actors with information to support their business. Clearly, whether an actor chooses the right filter and subscribes that filter to the right virtual device is his or her responsibility and is particularly dependant on the business case. In the case that was used for the reference implementation, a gas station clerk subscribes a threshold filter to a carwash’s soap level (see 3.5 on page 21). The same demo also supports state enter filters, which notify an actor in case a certain property enters one of one or more states. These filters provide functionality to notify an actor when the carwash went into some error-state or the gaspump is malfunctioning.
Chapter 12

Business Rule

An actor is able to interact directly with one or more virtual devices in the architecture through sending business actions (from its virtual actor) to a business rule. The business rule communicates with the different virtual devices to accomplish the result desired by the actor. As such, a business rule object defines the functionality for one or more virtual devices (see figure 12.1 on page 58). Both the business rule and the filter are interactive components.

12.1 Answers to generic design questions

The following answers the generic design questions give insight in the business rule:

1. Component functionality

   - What specific functionality does the component add to the architecture? A business rule offers virtual actors the possibility to interact with the architecture by controlling individual, or a combination of virtual devices. It is the interface through which virtual actors are able to interact with particular virtual devices functionality in the architecture.

   - Who makes the important decisions about the components functionality? A business rule’s functionality depends on an actor’s need, virtual device’s limitations and what an administrator thinks is necessary. Decisions about the specific functionality are made by the relevant administrator.

2. Component life-cycle

   - Who instantiates the component? A business rule is instantiated by the administrator who is responsible for the virtual devices the business rule will act on. Administrators will want full insight in the business rule’s specific functionality.

   - When is the component instantiated? A business rule is instantiated as soon as there is an actor-need for its specific functionality.
• **Where does the component live?** To reduce communication overhead, a business rule will live as close to the virtual device(s) it will interact with as possible. If a business rule interacts with more virtual devices in different organizational domains, it might be located in a trusted environment.

• **How is a component modified?** A business rule is modified by destroying the old business rule and instantiating a business rule with new functionality.

• **When is the component destroyed?** A business rule can be destroyed if after a certain amount of time no other component in the network requires its functionality. The specific time is dependent on the type of business rule and in what business case it is implemented.

3. Component interaction

• **Who is allowed to interact with the component?** An virtual actor sends a signed business event to the business rule to interact with the system. If the actor’s rights for the requested functionality are unavailable, they are automatically requested with the relevant administrator.

• **What other components does this component interact with directly?** A business rule receives business actions from a virtual actor. After processing, it interacts with one or more virtual devices and optionally sends a response back to the virtual actor.

### 12.2 Examples of Business Rule functionality

Which functionality is make available through a business rule depends on what is necessary in specific cases. Examples of typical business rule functionality are:

- set property value
- invoke a particular (customized) action on a real device
- invoke a control-action on a sub-system; for example, start a simulation connected to several virtual devices

### 12.3 Business Rule impact assessment

A business rule is defined and instantiated by an administrator. An actor is the one who would suggest that he needs certain functionality and would ask the administrator to define and instantiate a business rule. It is important for an administrator to assess the impact a business rule has on the system. Actor-use of the business rule is restricted by the use of the *intrusion factor* which limits the actor’s access to a business rule within the defined time frame. For more information on the *intrusion factor* see the section 13.1 on page 59 which covers the *Request For Rights* component.
12.4 Compound interactive component

It is very likely that in day-to-day application of the architecture feedback functionality is be needed to be able to influence/change the value of a certain virtual devices’s property to a certain preferred value by changing another property of the same or another virtual device. Since a filter is used to query a virtual device for its property-values and a business rule is needed to invoke a system change, a component which offers this simple feedback functionality would need to contain functionality from both a filter as a business rule. The flexible structure of the Basic Component\(^1\) allows a component’s repository to be filled with components which contain both filter as business rule functionality. What a compound component which such functionality would exactly look like remains a subject for further study, as well as which protocols are to be used for component initiation.

12.5 Business Rules in E-Gas Station

In the E-Gas case, the following functionality equivalent to a business rule has been implemented:

- functionality to remotely set a property value on a virtual device, which can for instance be used to remotely reset a carwash system which went into an error-state (of course after verifying that the system-error was not serious)
- queries which return the current number of virtual actors subscribed to a particular filter
- queries returning the number and type(s) of filter(s) instantiated with a particular virtual device\(^2\)

---

\(^1\)see section 5.1 on page 28

\(^2\)The architecture reference implementation deviates slightly from the final conceptual architecture. For more information about the differences please see section 14.5 on page 69
Chapter 13

Moving components

This chapter describes the moving components which flow between the actor and the interactive components; Request For Rights, Business Event and Business Action. The Request For Rights is covered in section 13.1, the Business Event in section 13.3 and the Business Action in section 13.4.

13.1 Request For Rights

This section describes the Request For Rights-object. A Request For Rights is created by an actor and sent to an administrator to request rights to interact with the system in a certain way. Interaction with the system can anything from querying a device, subscribing to a virtual device, to issuing a business action.

The 'Request For Rights'-object is one of the key components of the architecture, since its structure (together with the messaging-component) needs to support the architecture functionality. When an interactive component receives a signed RFR\(^1\) it must know exactly what rights have been granted, so it is important that every messaging-component in the architecture has one and the the same interpretation of a Request For Rights. The messaging-component which parses a Request For Rights must know exactly what rights have been granted.

13.1.1 Subscribing to a Filter

When requesting rights to subscribe to a filter, an actor can ask for either of two things:

- *Rights to receive certain property changes:* The administrator grants the actor rights to have a generic filter updated on a property changes of virtual device \(x\)’s property \(y\).

- *Rights for a particular filter subscription\(^2\):* The administrator grants rights for an actor to subscribe to a specific filter on a specific virtual device.

---

\(^1\)Request For Rights

\(^2\)Rights to ask the system certain questions.
Which of these two Request For Rights is applicable depends on whether an administrator wants to restrict an actor’s direct access to the virtual device’s properties. In the first case, the actor can self-decide what he/she wants to do with the information, by sending the ticket to a filter of his/her choice, while in the second case the administrator restricts use to a particular filter which might send only limited or derived information to the actor. If an administrator chooses to restrict which filter can be used with particular granted rights, he/she limits how information is can be used by the actor.

If an administrator decides that an actor has the right to receive certain information within a particular time frame, from then on it is the actor’s responsibility what happens to the events that he/she receives. It’s his/her responsibility to see that no-one besides him or herself has access to the information. Based on various criteria such as the particular actor, the business case and the information involved this may or may not be acceptable to the administrator.

Rights for property change updates: An actor receives rights to receive property change events of a particular virtual device(’s property). The administrator feels there is no need to restrict access to the actual virtual device’s property value. An actor can use the ticket to send this to an arbitrary (admin-)trusted Filter and have the property change events processed by the filter as he sees fit.

Rights for particular filter subscription: An actor receives rights to subscribe to a particular filter. The rights he received are only valid for the particular filter which could hide the actor property change events from the actor and only forward derived information to the actor.

13.1.2 Issuing a Business Action

The Request For Rights object for a business action is not very different from the RFR-object for a filter, with respect that there is no filter-class included in the object. What type rights have been granted depends on the RequestType-property, since business actions are mostly specific in nature.

13.1.3 Request For Rights properties

This subsection introduces the basic structure of a Request For Rights-object. The modelling of the RFR-object is described in section 13.2 on page 61. The following attributes are all relevant for defining a request for rights:

- **Actor:** Who issued the request?
- **Virtual Device:** What Virtual Device is the actor asking rights for?
- **RequestType:** What does the actor want, exactly?
- **Property:** What property is the actor interested in?
• **Intrusion Factor**: What is the max # updates/sec. this actor is allowed to receive?

• **Duration**: The rights are valid from beginTimeStamp, to endTimeStamp

In case an actor asks for specific rights (rights for a special filter) the Filter(class) or a class-derived hashcode is added to the Request For Rights structure.

Apart from requesting rights to be notified of a property changes (single time query), which can be interpreted by a filter, an actor can also request rights to set a property-value. That the actor exactly wants is indicated in the RequestType.

13.1.4 Requesting Rights in a Multi-Actor environment

When a filter needs to listen to more than one virtual device over different organizations to perform its task, rights from different administrators are needed to place listeners on the different virtual devices. Depending on whether any of the administrators want to restrict access to the actual information the filter is or is not included in the Request For Rights. If any of the administrators wants to restrict this access to the direct use of information, a trusted environment is needed where a trusted Filter can evaluate all the different property changes before sending a business event to the actor that is interested. See appendix F.4 on page 111 for a description of interactive components in a trusted environment.

13.2 Requests for Rights in E-Gas Station

The basic structure of the Request For Rights-object was modelled using concepts of NIAM-modelling technique (see [Guido Bakema, 1996]) which uses example sentences to model information structures. The following example sentence was used to model the structure of the request-object used to request rights to stay updated on property changes:

Actor "Gas Station Service Clerk" requests rights to RequestType "query/listen" (to) property "soap level" of the Virtual Device "Carwash" in domain "Gas Station Menlo Park" with an Intrusion Factor update-level of 5 minutes from beginTimeStamp "01-01-2001, 12:00" until endTimeStamp "01-01-2002, 12:00"

Previous example sentence leads to a "Request For Rights"-object with the following properties and property-values:

Actor: "Gas Station Service Clerk"
Virtual Device: "Carwash"
Property: "soap level"
Domain: "Gas Station Menlo Park"
Request Type: "query/listen"
Intrusion Factor: "5 minutes"
begin_TimeStamp: "01-01-2001, 12:00"
end_TimeStamp: "01-01-2002, 12:00"

The relevant properties for the Request For Rights object are:

- Actor
- Virtual Device
- Property
- Domain
- Request Type
- Intrusion Factor
- begin_TimeStamp
- end_TimeStamp

### 13.3 Business Event

An actor is notified of a changed condition of a virtual device's property through a business event. When a virtual actor receives a business event, it can be displayed directly or be used for further processing.

Just like the Request for Rights object, the Business Event has been modelled using the techniques of the NIAM modelling technique, see [Guido Bakema, 1996] and is based on the following example structure:

On TimeStamp "Monday 15 October 2001, 10:31" Virtual Device "Carwash 1"'s property "soap level" (from Domain: "Gas Station Menlo Park") was Old_Value "6" and became New_Value "4" which is less than Threshold Value "5".

The previous example sentence leads to the following properties and property-values for a Business Event object:

- TimeStamp: "Monday 15 October 2001, 10:31"
- Virtual Device: "Carwash 1"
- Domain: "Gas Station Menlo Park"
- Property: "soap level"
- Old_Value: "6"
- New_Value: "4"
- Threshold_Value: "5"

The relevant properties for the Business Event object are:

- TimeStamp
- Virtual Device
It is of importance that business events which are sent out from filters contain objective information. Appendix F.2 on page 110 contains a reflection on importance of objective information in business events.

13.4 Business Action

An actor interacts with the system and the functionality a virtual device offers through sending a business actions to a business rule. Business actions are either authorized beforehand through a ticket which is stored in the business rule, or through a ticket which wraps the business action. If rights are unavailable for the particular business action, a request is automatically generated for the necessary rights. See appendix E.1.3 on page 104 for a description of the administration paradigm which incorporates automatic requests for rights.

13.5 Business Actions in E-Gas Station

A business action can have many functions, of which:

- Invoke a change on one of a virtual device’s properties, like:
  - Reset a gas pump that went into an erroneous state
  - Change (a preferred supplier) or order-threshold on the stockLevel of soap in a gas station (or a wrapped ERP-system in a worldwide supply chain.)

- Invoke a what-if simulation to compute the effects of any of three decision-alternatives.

In the E-Gas case various business actions regarding setting and querying of virtual device properties were implemented, as well as querying which filters are listening to a particular virtual device and which actors have subscribed to a particular filter. It is noted that the actual implementation of the architecture differs slightly from the conceptualized model.
Chapter 14

Reflection on the architecture

This chapter reflects on the presented architecture in several ways. First, in section 14.1, proof is established the architecture meets the requirements as stated in the E-Gas case. After that, in section 14.2 the flexibility of the solution is demonstrated by quickly implementing new functionality based on a new business requirement. The second section of this chapter demonstrates the architecture is applicable to a much wider range of business problems by looking at generic properties of the E-Gas business case. Limitations of the architecture’s conceptual model are covered in section 14.4, followed by the limitations to the implementation (section 14.5) as developed during the thesis-project. In the final part, section 14.6, this chapter answers the research questions, posed in 2.4.3 on page 8.

14.1 Architecture meets E-Gas requirements

This section reflects on the requirements for the solution from the E-Gas case and proves each of the requirements are met by the architecture which has been conceptualized. The following requirements were identified during a planning session with one of the E-Gas’s senior managers, as is described in section 3.1.

- **make devices more visible to the net:** Wrapping each (legacy) device in the organization to a virtual representation enables actors throughout the entire network to subscribe to particular information regarding these components and interact with devices if necessary.

- **use of open standards:** Open standards, such as the Java-language and XML, in combination with well specified component interfaces, prevent vendor lock-in and take away the need for multiple interfaces.

- **remote software distribution:** The flexible structure of the components, consisting of individually replaceable functional sub-components, allows remote updates of software throughout the organization.

- **remote monitoring and administration:** Introduction of concepts as the filter and business rule, in combination with the request-based issuing of certain rights for system functionality, allow actors to remotely subscribe
to events which might be interesting to them and act a part of the system, if necessary. Administration of the components in the architecture is handled through interaction with an administrator-component which allows a physical administrator to either grant or deny requests for rights.

- **solution flexibility, extendibility:** The flexible component-based structure, adhering to open standards, allows new components to be added to the architecture, as well as existing components’s functionality to be extended.

- **collect sales data for datamining and trend analysis:** When wrapping the Point Of Sales terminal to a virtual representation, picking the right component’properties and the right combination of Filters, the architecture can easily be used to collect sales data from Point of Sales-terminals in gas stations all over the country and send it to a data-warehouse for trend-analysis at no extra costs.

The conceptualized framework meets all of the requirements from the E-Gas case. The interested reader is invited to turn to appendix C.4.4 on page 94 to see what is left of the concept after a confrontation with theoretical fields.

## 14.2 Architecture flexibility for E-Gas

From the paragraphs on E-Gas throughout this thesis, it becomes clear that the architecture is a perfect tool to tackle the business challenges E-Gas is facing. The working reference implementation and demonstration of a solution for the E-Gas case can only add to that. The following paragraphs demonstrate the architecture’s flexibility by addressing a new business case within the E-Gas context.

Deployment of the proposed architecture to a company-wide chain of gas stations allows extra functionality (driven by a new business problem) to be added without making any changes to a architecture’s basic structure. Imagine an example where E-Gas management wishes to introduce a surveillance system which registers drivers that do not pay at the Point of Sales\(^1\) by making a short video of their car and license plate.

Using the generic architecture, a new structure supporting this new requirements can be deployed in a short time using minor modifications to the system. Figure 14.1 on page 66 shows that a (virtual) police-officer is introduced as a new actor, who is be notified when a customer does not pay for the gas at the Point of Sales. A *Jini*-based camera is introduced to the system to take video footage of the cars that visit the gas station. The Point of Sales is wrapped to a virtual representation, which shares the property *time-frame* (that the pump was in-operation) with the virtual gas pump. Apart from this property, the virtual Point of Sales keeps track of whether a payment has been made for a particular time-frame. The, in this case, somewhat more complex filter, listens to the gas pump, point of sales and camera and sends out a *business event* with video footage when all the following conditions are met:

---

\(^1\)In Europe, visitors at a gas station pay for the amount of gas they have tanked at the Point of Sales. Customers that ‘forget’ to pay need to be registered.
Figure 14.1: Architecture application: A Jini-based surveillance camera

- the pump was in use for a certain time-frame
- no payment has been made for that time-frame
- more than 10 minutes have past since the last payment

If a payment is met within 10 minutes, the video footage (which is cached in the filter-component) will be deleted for privacy reasons. The amount of 10 minutes is chosen arbitrarily; if at certain very busy gas stations 10 minutes turns out to be too short this time can be modified by someone with the proper rights very quickly, by sending out a new subscription to the filter.

This quick implementation of a solution to a new business problem for E-Gas underlines the flexibility of the architecture to adapt to changes in information-requirement.

14.3 Generalization of architecture applicability

Now that the conceptualized architecture shows to a solution which excellently meets E-Gas’s requirements, it is interesting to show the applicability of the architecture lies beyond the scope of just this single business case. The following list generalizes the E-Gas business case by identifying the generic properties the E-Gas case meets. Properties for a business environment and case are divided into two different area’s; those involving complexity on a business level, and those involving complexity on a technical level. The E-Gas business case and environment is characterized by the following notions:

- Business complexity
  - highly information-intensive environment
  - multi-actor environment
– (same) information is interesting for different actors
– dynamic information-need by many actors
– business processes span several organizations

• Technical complexity
– many types of physical (legacy) devices play an important part in business process, they contain (the most) important control-information for the entire organization
– devices at many different geographic locations

Reflecting on this generalization of the E-Gas problem-field, it turns out that the architecture as conceptualized is also applicable to a wide range of other problems. For a full overview of the confrontation of the architecture with the theory and these more generic properties, see C.4.4 on page 94. Any distributed multi-actor environment, where business processes transcend single organizations and locations are geographically widely dispersed, the particular distributed solution, as proposed, has advantages over an traditional centralized solutions ERP-system, being more flexible, extendible and more catered to multi-actor involvement.

Whether or not it would be feasible to implement the system in another business case should, of course, be a subject for study on a per case basis, also taking into account e.g. the financial aspect, such as the implementation costs of the solution, and potential benefit.

14.4 Limitations of the architecture’s conceptual model

This section reflects on the architecture’s conceptual model as presented in this thesis by addressing the architecture’s limitations. Many of the limitations presented in this section are a recommendation for further research in the next chapter.

• Stacks of Filters: The architecture’s conceptual model presents a single layer of filters through which virtual actors receive their business events. Currently, the layering of filters has not been included in the conceptual model, but its introduction might reduce the number of filters necessary to achieve certain functionality. Later, it might turn out that it is feasible for a special type of filter to present information in an aggregated form, to which another filter could subscribe. The technical, organizational, business and other implications of this conceptual modification have not been a subject of study and are left to as a recommendation.

• Ability to retract granted rights: In the current implementation it is not possible for an administrator to retract granted rights. If this functionality is needed, a revocation list must be added, either to each individual component, or implemented as centralized functionality. A revocation list sub-component could be added quite easily requiring only minor changes
to the messaging-sub-component, by comparing the incoming request with the revoked rights available in the sub-component.

- **Architecture connection with methodologies for business process modelling:** While the conceptual architecture presents interesting new ways of extracting information from various sources in a distributed environment, it does not make any claims about how this information should be used to optimize business processes. In this respect, integrating the architecture with a methodology for business process modelling allows business engineers to expose the choices which lead to business events, which require actor action. Research on how the architecture should/could be combined with business modelling remains a very interesting subject for further study.

- **'Model-based filter capability':** Ideally, advanced filters could track an instantiated (sub-)system based on a prediction model for particular behavior. Any deviation from the prediction model would be sent to the relevant actor as a business event. Current filters do not incorporate behavioral prediction models. Based on the idea that it is not necessary to send events as long as the system (the set of virtual devices) behaves according to expected model, this concept promises to be the next step regarding optimization of complex value chains in a distributed environment. Development of these concepts and integration into the working implementation is a recommendation for further research.

- **Virtual device definition to support complex properties:** In the current conceptualization, information in a virtual device can be either numeric or text-based. The support of more complicated data-types would facilitate more difficult questions and functionality. As new business cases supported using the architecture new and complex properties might be developed.

- **Automatic relocation of filters:** Filter-locations could be dynamically optimized depending on factors like; the number of subscriptions, number of virtual devices a filter listens to. Development and implementation of filter-optimalization algorithms might reduce use of system resources.

- **Compound component:** Section 12.4 on page 58 introduces the compound component, incorporating both filter-, as business rule-functionality. Since the compound component undertakes pre-defined actions based on certain system-conditions it leans towards software agent-behavior. More research on the compound component is necessary on the conceptualization and the implementation of the compound component, as well as the application.

- **The Jini-security model:** In the current concept and implementation, access to the Jini-lookup service is not restricted in any way, which makes it easy for unauthorized actors to query what virtual devices and interactive components are available in the architecture. Since competitors might be
able to derive valuable information from this meta-information, this issue requires attention.

14.5 Limitations of the current implementation of the architecture

This section reflects on the current implementation of the architecture in the Java-language as developed during the thesis-project at Sun Microsystems’s iForce Ready Center in Menlo Park, CA. The following issues reflect differences between the architecture implementation and the conceptualized architecture as presented in this thesis:

- **Instantiation of filters:** The current implementation differs from the presented architecture by the fact that filters are currently instantiated by a virtual device which receives a signed Requests For Rights. In the definitive conceptual architecture, a virtual actor would not communicate directly with a virtual device, but send its requests to an admin-instantiated filter- or business rule-instance. It is recommended to make modifications to the implementation so it fully complies with the conceptual architecture as presented in this thesis.

- **Establishment of trust:** In the current implementation of the architecture trust between components is not established. It is recommended to modify current implementation to support establishment of trust between components.

- **Connection-status between virtual device and real/simulated device:** As pointed out in section 9.3 on page 46, in some business cases, it might be feasible to monitor the status of the connection between a real or simulated device and its virtual representation. Implementation of a special property with a virtual device which indicates this status is a recommendation for further development.

14.6 Answers to the research questions

This section answers the research question as posed in section 2.4.3 on page 8. The architecture presented in this thesis, allows actors in E-Gas’s distributed business environment, to interact with and extract information from wrapped devices to help them make decisions in an information-intensive multi-actor environment.

Using the concepts as, a homogeneous layer of information presentation, maximal flexibility using the request-based distribution of rights, model-based generation of exceptions throughout value chains the architecture has the potential to provide actors in distributed networks with more insight the efficiency of business processes, the role of their own organization in these processes and, based on their problem perception, what they can do contribute to inter-organizational process optimization. Although the current reference implementation does not
yet fully support all conceptual notions as presented in this thesis, actors can subscribe to information in a distributed environment through filters which transcend organizations and which are alerted by business events of relevant events. The non-conventional way of asking rights for system functionality has the potential to greatly reduce administrative response time, user frustration and improve business efficiency.

The architecture describes a semi-automated solution with a generic design which recognizes the fact that some decisions should be taken by humans after careful consideration of more relevant aspects, while at the same time it allows automated decision processes to be implemented in cases that a human interpretation does not add to the quality of the decision. It does just to the complexity of day-to-day practice, while at the same time offering tools for actors to manage this complexity. As authority in the field of business and information technology Peter Keen\textsuperscript{2} suggests in similar words on his homepage, the best way for optimizing processes does not lie in full automation, but having intelligent people at key-positions take decisions that make a change to business.

The basic architecture, as presented, combines insights from various scientific disciplines which led to a reference implementation in the Java-language. The E-Gas demo is available for testing and further development at \url{www.sk.tbm.tudelft.nl/narad}.

1. **What are the requirements for a solution which gives various actors in E-Gas Station’s distributed business environment the ability to remotely monitor and control devices?** The requirements for a solution for remote monitoring and control have been derived from an interview with one of E-Gas’s senior managers. The requirements are found in section 3.1 on page 13.

2. **Based on these requirements, what would a solution which meets these requirements look like?** The conceptual architecture is presented in chapter 3, and introduced in more detail in chapter 4. Chapters 5 to 13 describe the different components that are part of the architecture. This chapter reflects on the conceptualized architecture and shows it meets its requirements.

3. **What does an implementation of the architecture’s conceptual model in the Java-language look like?** The documented source-code for the reference implementation of the architecture and the E-Gas demonstration is downloadable from \url{www.sk.tbm.tudelft.nl/narad}. Detailed description of the source-code lies beyond the scope of this thesis.

The next chapter which contains the conclusions of this thesis argues the thesis goal is met by the proposed solution.

\textsuperscript{2}\url{www.peterkeen.com}
Chapter 15

Conclusion and Recommendations

This chapter concludes this thesis with an overview of the most important properties of the architecture and its applicability and usefulness to the E-Gas case. After that, it makes recommendations to the various parties involved in this project; E-Gas management, Sun’s iForce Ready Center and the scientific community, more in particular the Systems Engineering group at Delft University of Technology.

15.1 Conclusion

This thesis promised to present an architecture for flexible, real-time monitoring and control of devices in the E-Gas business environment, which would allow various actors involved in primary or support processes to extract information to be used to take decisions and optimize processes. Using non-conventional concepts on distribution of rights and standardized interactions between components within the distributed architecture, it did just that. The solution, if fully configured, supplies the right actor with the right information at the right time and allows him/her to act upon it, if he or she is authorized to do so.

More in detail, the architecture allows different actors to subscribe to information which is relevant for their role and problem perception, allowing them to shape their own view on the over-complex reality of the business chain and information in enterprise environments. Non-relevant information, information which doesn’t require decisive action by the actor, is not presented by default, which gives the actor a sharper focus on relevant issues. The next step is for E-Gas to identify in detail which actors benefit from which information and how the concept that any information (in potential) is available could be best used to help actors optimize processes and raise service levels.

Apart from keeping its promise, the presented architecture is applicable to many business cases with similar properties and challenges as met in the E-Gas case; net-enabling information available within devices at dispersed locations and different organizations and seeing that it can meet actor’s information need. This architecture builds the foundation for further research on a wide
range of tools for optimization of processes which transcend the single organization domain and stretch out to into a complex network. Using the distributed concept as source of information, with an abstraction between real devices and their implementation allows the coupling of simulations and real systems to the architecture with a promising potential.

Where business processes transcend multiple organizations, optimization problems change in nature, from mono-actor to multi-actor optimization and new questions arise. Whereas optimization from a single-organization point of view might be most beneficial for one organization, optimization of the chain which transcends multiple organizations might establish higher overall efficiency. The architecture presents the basic tools to provide actors in many involved organizations with insight in value chains and enable extend to architecture functionality to support model-based tracking of selected parts of simulation- or system-reality.

The architecture, as presented, shows only a tip of the iceberg of possibilities for coordinating information exchange and business in a networked environment. Many exciting issues are still open for further research, discussion and improvement. Recommendations to the various organizations involved in the process during this thesis project, among other things concerning modification of the architecture’s implementation and extension of the conceptual model, as well as research into application of the architecture in organizations are presented in the next section.

15.2 Recommendations

Recommendations for further research and development of the architecture are be split up into different area’s. First, several recommendations are made to E-Gas’s management regarding the deployment of the architecture throughout the organization. Next, recommendations are made to Sun Microsystems and the iForce Ready Center. After that, recommendations are done to the scientific community, focussing on further research of the generic architecture’s conceptual model and its implementation in the Java-language. Recommendations regarding the validation of the architecture’s concepts and its applicability to a real, multi-actor, distributed business environment conclude this chapter.

15.2.1 Recommendations for E-Gas management

It has been shown that the architecture as presented in this thesis meets the requirements to supply the right actor with the right information at the right time in a distributed business environment. The following recommendations help E-Gas management to leverage the solution and supply actors involved in E-Gas’s business processes with the information to optimize these processes with the idea to cut costs and improve of service levels to increase revenue.

- Early estimates suggest that implementation of this architecture will supply actors with information to optimize processes, which will increase profit, but quantitative information to found this assumption is limited
at this moment. Research into the costs of (the needed) further development and implementation of the E-Gas specific solution, as well as the forecasted financial benefits is recommended.

- While this new technology might not be ready for a large scale roll-out at this moment, starting with small-scale pilot-projects as a proof of concept within the E-Gas organization will help actors to work with and develop this new technology and establish its usefulness on a larger scale.

- Currently, hardly and device found any gas station all over the US in connected to a network. It is recommended to contact manufacturers of all the devices and tell them E-Gas would like the next version of their device to be net-enabled and equipped with Java-technology, so wrapping it to a virtual representation can be done quickly with very limited resources, allowing the devices to really be part of the networked organization.

15.2.2 Recommendations for Sun Microsystems and the iForce Ready Center

Throughout its relatively short history, Sun Microsystem’s vision in the field of distributed computing has been exemplary. In many cases Sun led the way where other companies followed later. With Sun’s understanding of the importance of common interfaces between components, open standards and community building throughout solution-developers and solution-users there is a window of opportunity for the support of the technology as described in this thesis.

- The further development and adoption of the idea’s presented in this thesis would be helped by Sun’s support. Sun is recommended to supply the infrastructure to build a community for further work on the concepts as presented in this thesis, focusing on further standardization of the concepts, administration of developer-specific contributions and to help further develop idea’s as presented in this thesis.

- Sun continues to bring together best-of-breed solutions and confront visitors to the iForce Ready Centers with solutions which show how using information and technology used in different ways, has the potential to greatly improve efficiency and open up new business opportunities. The iForce Ready Center is advised to furnish one corner in the iForce lab to show fully distributed solutions, based on the architecture and idea’s as presented in this thesis in a permanent exhibition, to confront visiting business people with sparkling, new ideas which might benefit their organization and way of thinking.

- The iForce Ready Center is invited to use the components described in this thesis work for rapid development of industry-specific demonstrations and help extend the library of components in this way.
15.2.3 Recommendations for further research on the architecture

This section presents recommendations regarding modifications to the architecture implementation and its conceptual model to be part of new research and development, in particular the Systems Engineering-group at Delft University of Technology.

- Update the current implementation of the architecture to meet the latest conceptual model: The current implementation of the architecture in the Java-language differs slightly from the conceptual architecture. It is recommended to update the current implementation with the latest conceptual model. Chapter 14 contains a detailed list of items open for modification.

- Research into the extension of the architecture concepts: The following recommendations sketch directions for further research and extension of the concepts of the architecture. A founded overview of the recommendations is found in chapter 14 which reflects on the architecture.
  - Develop new filters and business rules
  - Implement a revocation list
  - Research, develop and test multi-level filters
  - Integrate the architecture with methodologies for business process modelling
  - Research and develop model-based filter functionality
  - Extend virtual device definition to support complex properties
  - Research automatic relocation of filters
  - Research, implement and test the compound component
  - Research modifications to the Jini-security model

15.2.4 Validation of the architectural concepts and its implementation

This subsection sketches questions for scientific research into the validity of the architecture in a real business case, or even deployment in an enterprise environment. Questions with regard to the validity and applicability of the architecture can be categorized in macro-, meso-, and micro-categories; zooming in on inter-organizational issues, intra-organizational validation and way the architecture changes processes for the individual information-worker:

- Macro: Questions at inter-organizational level:
  - Under what conditions are (which type of) organizations willing to share information using the architecture, embrace its ad-hoc, peer-to-peer way of establishing and maintaining relationships?
What mix of incentives is needed/helpful to get organizations to share information using the paradigms described in this thesis?

How can the process of sharing information be formalized?

To what extent is it possible and relevant for different companies use the same concepts (e.g. virtual devices) to wrap information in their organization?

How can benefits from multi-actor optimization be re-distributed among various organizations in the complex value chain?

- **Meso**: Questions at (intra-)organizational level:

  - To what extent will the introduction of the architecture and use of its non-conventional administrative concepts reduce administrative overhead (categorized for different types of organizations)? Where does the introduction of the architecture shift the bottlenecks to?

  - Will the architecture, when fully deployed, really change the way business managers make their key-decisions in a distributed network or will they keep to rely on their system administrators, hindered by their lack of technical knowledge?

  - What are relevant/interesting workflow design patterns for various business processes where human decision making is to be intertwined with automated processes?

- **Micro**: Questions at the level of the individual actors/users:

  - Is creating a Request For Rights easy, moderate or complex for an average actor? Do actors accept the architecture into their daily way of working?

  - Do the concepts, as presented in this thesis, help information workers to express their information-need and establish common ground for discussion about organizational concepts, or is more needed?

  - *Granularity of rights*: Is the chosen granularity of rights to be granted sufficient for application on most of the organizational domains? Where is more or less fine-grained rights-management needed?

Despite the above questions for further research on the validity and application of the architecture in a real organizational environment, the architecture can already be used by early adaptors and companies which recognize its potential for small pilot-projects. The faster a basic infrastructure of virtual devices is available within organizations and system administrators, developers and users start to experiment with its concepts and technology, the faster the component library of filters and business rules, device-wrappers and virtual actor sub-components will grow. Clearly, the architecture as presented in this thesis becomes increasingly more valuable if it can grow a critical mass of supporters.
Appendix A

Definitions

This chapter provides a definition for the various concepts used in the architecture. Section A.1 describes the definitions specific to the architecture, A.2 describes security-specific concepts, whereas A.3 presents some more general definitions.

A.1 Definitions regarding the architecture

**Actor Panel:** A physical actor interacts with its virtual representation through an actor panel.

**Administrator Panel:** A physical administrator decides which actors have access to which resources by answering requests for rights presented in his/her Administrator panel.

**Business Action:** An (virtual) actor sends a Business Action to a Business Rule to invoke a change in a (sub-)system.

**Business Event:** A Business Event is the message that a filter sends to a virtual actor to notify a certain condition has been met (usually triggered by the fact that a virtual device has been updated).

**Business Rule:** A Business Rule is a (virtual) representation of the functionality one or more virtual devices advertise. A Business Rule is the only component which can directly send messages into a virtual device.

**Control Panel:** A Control Panel is an actor panel that contains both control panel components and display panel components.

**Control Panel Component:** A Control Panel Component is a component on the actor panel through which an actor can actively intervene with the state of the system by sending a business event to a business rule.

**Dead-Reckoning:** The concept of Dead-Reckoning comes from the maritime world, where it refers to the use of a prediction model to minimize the necessary data to communicate about the position, direction and speed of a vessel.
**Display Panel:** A Display Panel is an actor panel that is only used for monitoring certain aspects of the system.

**Display Panel Component:** A Display Panel Component is a component on the actor panel which displays either the state of a virtual device or incoming business events from a virtual device.

**Domain:** Every virtual device, filter or business rule is part of a domain. A domain is uniquely identified on the net by an identifier based on the concept of the Universal Resource Location, also known as URL, as used for the unique identification of a resource on the internet.

**Filter:** A Filter is a component which handles the actor subscription to the occurrence of specific events in a virtual device and the notification of that (virtual) actor. In case a Filter is notified by a virtual device that its state has changed the Filter checks whether the change is of any interest to any subscribed actor and if so, it sends out business event(s).

**FilterSubscription:** A FilterSubscription is the object in the filter which indicates the condition when an actor would like to receive an event and the name of the actor.

**GUI Panel:** The GUI Panel or Graphical User Interface Panel is the interface through which an actor/administrator interacts with this virtual representation and thus the architecture. A GUI Panel can be either an Actor Panel or an Administrator Panel.

**Intrusion Factor:** The Intrusion Factor defines the amount of network or virtual device intrusion an actor is granted rights for.

**Interactive Component:** Interactive components in the architecture are the Filter and the Business Rule, since they interact directly with a virtual actor-component.

**NaradComponent:** The NaradComponent extends functionality directly from the JiniComponent and is contains the basic functionality for all the non-moving components in the architecture.

**Network Intrusion Factor** see Intrusion Factor.

**Organizational Domain:** All components that share the same administrator are part of the same organizational domain.

**Real Device:** Real devices are devices in the architecture that an actor has interest of interacting with in some way. A Real Device can be anything from a legacy system, a fully wrapped ERP-system, an embedded system or even a discrete event simulation.

**Request For Rights:** No actor in the system is allowed to interact with the system without valid rights to do so. An actor can request rights for certain actions by sending the proper administrator a Request For Rights. An
actor requests rights by creating a request for specific rights and sending
that request to an Administrator for approval. If approved, the Adminis-
trator returns digitally signed 'Request For Rights', called a Ticket.

**SEA-model:** The notion *SEA*-models stands for Subscription, Event, Action,
indicating that an actor Subscribes to a Filter by indicating his/her inter-
est in a specific virtual device’ condition; the virtual actor is notified
through a business Event, after which the actor decides and Acts on the
system.

**Subsidiarity:** The term subsidiarity in the context of this thesis refers to
the fact that information should be kept at a place where it is used to
answer questions for actors, and *not* be moved to locations, such as large
data-warehouses, to be stored and never used again.

**Ticket:** A Ticket is a Request For Rights signed by the Administrator who is
authorized to distribute rights for a particular virtual device.

**Virtual Actor:** A physical Actor has a virtual counterpart which contains the
Actor’s acquired rights, his/her panel properties, keypair, etc.

**Virtual Administrator:** An Administrator is responsible for controlling sys-
tem access by granting or denying a request from anyone who wishes to
interact with the system.

**Virtual Device:** A real device is wrapped to a virtual representation called
a Virtual Device. A Virtual Device is a software component with a state
(attribute-values for a certain moment in time) and methods. What a
particular Virtual Device should look like is defined by the need of infor-
mation of the Actors that will potentially be interacting with the device.

### A.2 Security specific definitions

**Confidentiality:** Information exchanged between two parties cannot be viewed
by unauthorized people.

**Data Integrity:** Data cannot be changed have not been changed without the
receiving party knowing this.

**Authentication:** A person or component is who he/she claims he/she is.

### A.3 General definitions

**TimeStamp:** A moment in time, identified by a date and a time.

**TimeFrame:** A period in time, starting with a begin TimeStamp, and ending
with an end TimeStamp.
Appendix B

E-Gas Station

This chapter gives background information on the E-Gas business case. It gives an overview of the different actors involved and covers the means-ends analysis that was made to get a grip on the important issues in the E-Gas business case.

B.1 Actors involved

This section gives an overview of the different actors that are concerned in the business surrounding a gas station which are interesting for this business case.

- gas station service clerk
- different types of suppliers
  - gasoline
  - store commodities
  - other supplies (soap deliverers, etc.)
- maintenance people from different companies
  - carwash maintenance people
  - point of sales maintenance people
  - maintenance for the gas pumps and installation
- different departments at HQ
  - marketing
  - procurement
  - strategy
  - tactics/planning

Each of this actors has need for different information to support his/her task. For example, a gas station service clerk needs to be informed when the soap reservoir of the carwash is running low, so he or she can refill the soap.
Clearly it is not necessary to send this information upstream to the oil company’s headquarters. Some department at headquarters might be interested to know how many carwash installations have been out-of-service over the past month, but that is a different question. The gas station service clerk, however, does not need to be bothered with statistical information regarding the sales of candy bars, which might on the other hand be very useful information for the marketing-department at headquarters to analyze the effect of a newly stated promotion campaign for this candy bar.

In theory each actor should be fed with exactly as much information as is necessary for him or her to perform his job, no more, no less. If an actor has less information than is necessary he/she might make a sub-optimal decision. If an actor has more information than is necessary, the useless information will hinder the process of decision-making. An exact analysis of the information requirements of the various actors was not considered useful for this thesis since it establishes *Proof of Concept* and does not aim to lay out a complete implementable solution.

### B.2 Means-Ends analysis

From the perspective of the oil-company management an analysis\(^1\) has been made of the current challenges the company is facing. From the core goal of the oil-company, increasing profitability, a decomposition is made into different areas’ where cost could be cut and where opportunities can be found for new business. The diagram can be found as B.1 on page 82. At the bottom of the diagram means are presented which could/should be deployed to help different actors involved to reach the company’s ends.

**Description with the diagram**  The main goal of the oil-company is to increase profitability, especially in times of economic downturn. Increasing profit can be done by either reducing cost of business or increasing revenues from business. Reducing cost of business means incorporates reducing cost of operations, cost of maintenance and cost of software updates. Cost of operations are reduced by taking away the need for multiple systems with the same functionality and preventing vendor lock-in. The cost of maintenance can be reduced reducing the number of visits to a site.

Revenue is increased by reducing downtime for services (which also helps to maintain the strong brand-perception the customers have), increasing the number of services offered to the customer and to have the information to cross-sell products and launch marketing campaigns. Downtime for services not only means that the gas pumps are always operational but also that the carwash and all other devices in the gas station are always operational. The number of services can be increased if a flexible solution is used for rapid deployment of these services.

---

\(^1\) This analysis was based on an interview with one the E-Gas’s senior managers at Sun Microsystems’s iForce Ready Center in Menlo Park, CA
From this diagram the following ends can be taken as a functional requirement for a generic tool to help achieve these goals:

- *make devices more visible to the net*
- *use open standards*
- *remote software distribution*
- *remote systems monitoring and administration*
- *progressive alarming through messaging*
- *use of flexible architecture*
- *ability to collect data for mining and trend-analysis*
The official guidelines for drawing a Means-Ends diagram have been interpreted loosely to enhance the communicative value of the diagram.
Appendix C

Theoretical notions

This chapter sketches various theoretical fields as a background with the case and solution presented in this thesis. It covers the following theoretical fields: Organizational Engineering / Business Engineering, Decisionmaking in a multi-actor setting and System Engineering / Software Engineering. This chapter sketches important notions on the various scientific disciplines which formed an important framework of thinking for the project.

C.1 Organizational engineering / Business Engineering

Although the field of Organizational Engineering generally tends to hold a soft view and focus more on non-formalizable information, formalized information\footnote{information available in easy accessible digital form} is becoming more and more important for accelerating and coordinating inter-organizational business processes. The field of Business Engineering in itself holds a more holistic approach towards the way organizations work and change, emphasizing that focus on organizational structure, business process and human and technological support cannot be seen as separate entities, but have an interaction which needs to be answered by an integral approach.

This section describes theory on the fields of organizational engineering, as well as business engineering. In the first subsection notions from the field of organizational engineering are presented, while the second subsection presents background on the field of business engineering.

C.1.1 Organizational engineering

While in the field of Organization & Management the authoritative view on the decisionmaking context used to be based on a hierarchical structure of the actors involved, nowadays the view that decisionmaking takes place in networks of actors better helps to understand actor behavior.

In [de Bruijn, 1999] it is stated that the main characteristics of these networks are pluralism, mutual dependencies of the actors involved, closedness of actors and dynamics.
• **Pluralism** refers to the variety of the actors involved in a typical decision-making network. Actors may differ in several dimensions like size, function, available resources, services & products offered or authority. Pluralism manifests itself between different organizations (inter-organizational pluralism), but also between departments within one organization (intra-organizational pluralism).

• **Mutual dependencies:** In the classical view actors are subordinate to other actors, in an invariable, hierarchical structure. In this view the subordinate actors are completely dependent upon their superiors. In contrast, the starting point of the network view is that the relationship between actors is better described by a relationship of mutual dependencies. One of the factors that cause a mutual rather than one-sided dependency relationship is the fact that actors are involved in multiple processes of decisionmaking. Because different types of decisions are involved, the dependency relationship may also differ from one process to another. In the end this mechanism will dampen the formal power of a formally superior actor.

• **Closedness:** The classical view on decisionmaking processes assumes the availability of all relevant information. In the network view, however, information and resources are thought to be distributed among the actors involved. Moreover, these actors are assumed to treat these resources strategically: they will not provide external access to resources if this harms their interests. This mechanism can be identified as exposure closeness. Actors and organizations can also display reception closeness; e.g. they do not respond to external signals.

• **Dynamics:** The classical view on decisionmaking processes assumes a static decisionmaking context: it is clear which actors are involved, what resources are available, what the current decision is about and what the dependency relationships between the actors are. In contrast, the network view assumes these aspects all to be dynamic in nature.

Requirements that arise form the field of organizational engineering are:

• The architecture must support the richness of actors, their interdependencies and their often dynamic relations. Since transactions in real-life are often based on complex and coincidental networks of actors, the architecture must support these concepts and the corresponding peer to peer relations.

• The actors in the architecture must be self empowered in the sense that they must be able to collect information and (re)act on it on an independent instigation. Empowerment also suggests controlling the amount of service or information an actor provides to others.

84
C.1.2 Business engineering

Business engineering is the field focusing on business process (re)design in conjunction with the organizational structure and technological solutions based on the strategy of an organization. It should be clear to the non-initiated reader that the field of business engineering’s focus is not limited to commercial organizations, but also studies change in business (processes) within governmental and other non-commercial organizations. Global markets and the current development of technology made businesses seek improvements in their business processes, question their efficiency and manageability, as well as question their core business. Current trends focus on decentralizing processes and therefore assume more locale and accurate information on different environments. An example of this trend is the make to order-concept (versus make to stock) which nowadays gains a lot of interest by the European car industry.

The architecture should be able to extract information on business processes within various organizations to be able to keep the pulse on level’s of efficiency and to extract various sorts of steering information at operational and management level.

Since aiding business process (re)design is clearly a goal of the architecture, interdependencies with other systems like simulation and visualization packages should be easily established. Recommended literature on business engineering is for example: [Sol, 2000].

C.2 Decision support in a multi-actor environment

A solution in a multi-actor setting should support many different actors with different requirements with their work and process of decisionmaking. This section extends the mono-actor centered model of decision making presented in appendix D.1 on page 97 to a multi-actor model, catered to cope with the complexities of decisionmaking in a network of organizations. The multi-actor model focuses on the concepts ownership of information and different organizational domains. Actor steering in a distributed chain is less emphasized since symmetric properties for authorization are valid.

The processes of decision making lies at the heart of running any operation and as such vital to businesses. Each actor within an organization needs information to support his or her decision process. For the conceptualization of a multi actor decision support reference model a distinction is made between an actor, an administrator and a developer, each with different requirements. The following roles are defined:

**Actor:** The actor is a physical person in need of information to support his or her decision making. The actor has a role in the organization and some kind of functional description.

**Administrator:** The administrator is the physical person who decides which other actors have access to information in his/her organizational domain and (if so) how information can be used by actors.
Developer: The developer involved in making modifications to the system has his/her particular requirements.

It is conceivable that the actor and administrator roles are fulfilled by one and the same physical person, but this is not mandatory. Due to the shift towards a networked society more and more of the information which is useful for an actor is found within different organizational domains, to which the actor in need of this information to support his/her work might not have access by default. Figure C.1 on page 86 sketches a model for discussion about ownership of information and how other actors are allowed to use this information in a distributed environment. Actors in the model are in need of information which is located in various organizational domains. Access to the information (and in some cases to how the information can be used) is restricted by the organizational domain’s administrator who acts as a gateway to the information which is available in the organization.

In the networked society where every organization is an owner of (potentially) valuable information which could benefit other actors, openness can bring added value to any different actor for which this information can be of added value for optimizing the process of decision making.

Currently, not all available information and knowledge available within organizations is shared with other actors for any of the following reasons:

- the organizational domain’s administrator doesn’t explicitly realize he/she has this information
- the organizational domain’s administrator doesn’t realize this information represents an economic value
- the organizational gains competitive advantage from keeping this information secret
- the organizational domain doesn’t have a relationship with the actor in need of this information

A solution for visualization and information extraction should support the actor’s need for flexibility in terms of access to (and use of) information available in various organizational domains in conjunction with the administrator’s
concern which actor’s he/she grants access to the available information. If rights
to access information can be granted without making administrator-intensive
configuration modifications to the IT-systems, the threshold for selling informa-
tion available within the organization is much lower.

There is a natural tension between the actor in need of information in an
organization or system, and the administrator responsible for keeping the sys-
tem operational and restricting access to parties that should not have access to
information. Clearly what is in the administrator’s interest (minimizing system
use and modification) conflicts with the actors desire to query the system and
mould it to his/her personal preference and information need.

C.3 Systems engineering / Software engineering

Software Engineering is, in the context of this thesis, mainly used to develop
systems which are described using concepts from the field of Systems Engi-
neering. The idea’s from the field of Systems Engineering of using an approach
of decomposition of a system to get to know the inner workings and the sub-
components in order to be able to understand and predict the system as a whole
combines very well with the field of Software Engineering where these concepts
are used to build programs for large computer systems.

This section presents concepts from the fields of systems engineering and
software engineering. The first subsection presents idea’s on the field of systems
engineering. The second subsection does into detail on recent insights from the
field of software engineering.

C.3.1 Systems engineering

Systems Engineering has been defined by [Jenkins, 1972] as:

the science of designing complex systems in their totality to ensure
that the component subsystems making up the system are designed,
fitted together, checked and operated in the most efficient way.

As such, Systems Engineering is the field that focusses on the decomposition
of systems into smaller components, describing their behavior assuming that the
combined behavior of the components is representative for the behavior of the
system as a whole. The field of Systems Engineering is an important part of the
Way of Thinking for the project; reducing system complexity and extending
the usability of a system by identification of relevant concepts and introducing
them to the system.

A system is decomposed into smaller components to reduce complexity.
Systems can be decomposed in different ways, among which:

• Phase systems which regard the system only during specific time frames.

• Sub models and aspect systems which only regard specific parts of a sys-
tem.
• Decomposition achieved by separating processes from objects; objects themselves are characterized by the object oriented approach of aggregation and specialization.

A further notion on system engineering is the difference in soft- and hard systems. Hard systems only regards the objects and relations that can be quantified. Soft systems focus as well on soft elements, values, relations and views. Due to the way of support chosen for this architecture\(^2\) a hard approach is chosen. The choice for a hard approach on the system-design-level does not reflect suggestion for the application of the architecture in a decision making cycles. For a more elaborate disclaimer on the usefulness of the architecture see appendix D.1.3 on page 100.

C.3.2 Software engineering

The field of Software Engineering is a rather new scientific discipline. Over the past decade the use of software increased drastically, as well as guidelines how to develop software within groups of developers. This section presents ideas on the development of software in groups, as well as the current paradigms in software engineering.

Engineering software in groups

Software engineering is the field of computer science that deals with the building of software systems which are so large or so complex that they are built by a team or teams of engineers. [Ghezzi, 1991]

[Ghezzi, 1991] introduces a firm basement for software engineering by the waterfall model of software engineering illustrated in figure C.2

![Waterfall model of software life cycle](image)

Figure C.2: Waterfall model of software life cycle

\(^2\)Using software as a solution
The waterfall model of software engineering defines the lifetime of software, focusing on the distinguishable phases.\textsuperscript{3}

Though this model has served most software development for the last decades, currently a shift is seen towards development of software in communities.

In line with the organizational networks, the globalization of the business environments and the strengths of global interaction and communication, companies like Sun Microsystems, IBM, Oracle, etc. have gained a far greater interest in a community process of software engineering and therefore to partner in development. The single actor perspective of the waterfall model has changed into an inter-organizational approach not focusing on phases but on rounds, not focusing on project but on process.

Figure C.3 gives an example of this approach in the \emph{Java community process}. This process describes how expert groups of several companies initially formalize a challenge or goal for the improvement of any Java library/solution. These expert groups are inter-organizational and present their ideas to the public, which can react on their proposals.

Key consequence of this community approach is the need for generic reusable solutions, which are not focused on the expectations of a single actor, but serve communities and hopefully ones to come.

Figure C.3: Java community process

\textbf{New paradigms in software development}

Not only the ‘way of working’ in software development underwent changes. As the science matured, more high-level languages became available which abstract from the bits and bytes and allow developers to think in terms of software objects and components, linked libraries and their interfaces and business-functionality. A full overview of the development of software languages clearly

\textsuperscript{3}Conceptual phases are often referred to as the \emph{high} phases, where implementation phases are referred to as the \emph{low} phases.
lies beyond the scope of this review of theory, although it is important to note just a few developments that shaped, and will continue to shape the industry:

- Re-use of components
- Modular composition of software components
- Object Oriented development in communities
- Compilation of software to interpretable byte-code instead of executables, enabling platform independent code, which use:
  - Virtual Machines as an extra layer of abstraction, introducing flexibility at the cost of processing power.

These developments steer the choice for technology used for implementation on the conceptual architecture. For further reference on Object Oriented modelling a good starting point is: [Fowler, 2000]. More on the Java-language which uses concepts like compilation to byte-code and virtual machines can be found online at java.sun.com.

More on Component based software development can be found in [Szyperski, 1998] and [Jell, 1998].

### C.4 Architecture confrontation with theory

This section combines requirements for a solution as presented in the theoretical background presented in appendix C on page 83 and explores what interesting new insights lie at the intersections of these theoretical fields. After that, the theoretical requirements for a solution are confronted with the requirements from the various theoretical notions.

#### C.4.1 Architecture requirements from theory

The following requirements were learned from the different scientific disciplines covered in the previous chapter:

1. *Decision support in a multi-actor environment*
   - An actor should be able to control the boundaries of his/her observable system
   - An actor should be able to monitor the state of the system
   - System information should be visualizable so it gives the actor insight and relevant information that fits his/her problem perception
   - An actor should receive formalized information, so he/she can use this information directly for further processing
   - An actor should be able to interact with his system

2. *Organizational and Business Engineering*
Support multiple actors and actor-empowerment

- Flexibility to meet dynamics of daily practice

3. Systems/Software Engineering

- Support various ways of sending and receiving messages (push, pull, a-synchronous, synchronous)
- Support subsidiarity\(^4\) with respect to storing and forwarding information
- Support interaction with legacy-devices, different architectures
- Component based, modularity for flexibility, extendibility

C.4.2 Disciplinary intersections

Based on the mentioned theoretical background and the research conducted during the project it was learned that the most interesting concepts are derived from combining various theoretical notions. Figure C.4 on page 91 graphically displays the theoretical insights based on combining ideas from Decision Support, Organizational Sciences and Systems Engineering. These insights are discussed in the following paragraphs:

Decision support and Organizational / Business Engineering

Due to the fact that the boundaries between organizations are changing (business processes transcend single organizations), more and more information which

\(^4\)for the definition see appendix A.1 on page 78
is relevant to an actor to support decisions for optimization of business processes
is found outside the actors own organizational domain.

New tools to support the actor’s decision process should recognize this issue
and allow organizations to easily share particular information and functionality
while restricting actors access to other information and functionality.

Decision support and Systems / Software Engineering

At the intersection of Decision Support and Systems Engineering new insights
lead to a paradigm-shift in the way an information system is questioned to
support an actor with his/her work. Traditionally an actor would (repeatedly)
query a system and get various results. With the new approach actors can not
only query a system and use the results for further processing, but systems will
self-notify actors of conditions which require their attention, based on preset
notification conditions. The functionality to stay up to date on certain condi-
tions should not only be available within an organization (or organizations
which have made difficult, inflexible agreements about sharing information),
but also between organizations that only just learned that sharing information
will benefit the way they do business.

Organizational / Business Engineering and Systems / Software En-
gineering

The ever-changing constellation of actors in the business environment asks for
support systems with the flexibility to meet these environmental and inter-
organizational business changes. Current information systems for supporting
business are often monolithic and not catered to meet the swift changes in
business which are required to maintain the competitive edge. The required
changes lie both in the area of (re)shaping the information systems as well as
their configuration. For that reason flexibility is required, as well as the need to
minimize administrative overhead. Another important requirement is the ability
for for different systems to be able to exchange information. In this regard,
open standards are an invaluable driver for inter-organizational integration of
processes and process control. Open standards are an important requirement.

Decision support, Organizational / Business Engineering and Sys-
tems / Software Engineering

At the heart of the Venn-diagram where all the theoretical fields meet lies the
concept of Event generation based on model exceptions. This concepts consists
of the notion that an actor should have the possibility to compare a part of
the relevant system with a (mathematical) prediction model. As long as the
(sub-)system behaves according to the agreed model no events are sent, while
the moment the system deviates from the behavior predicted by the model
events are sent. More often than not this model will be accompanied by a
business transaction (or Service Level Agreement) in which one organization
makes a promise to deliver some kind of business result. Depending on the
inter-organizational separation of responsibilities either the requesting organization or the tracked organization or both should be notified as soon as an exception of a predicted situation occurs. Depending on whether the model is descriptive or prescriptive in nature, either the model would be modified to meet with the changed environment or the model is a guideline for re-evaluating operational management-procedures. The concept of using a prediction model to keep track of a sub-system was first used in the maritime world to limit communication about the position, direction and speed of a vessel without limiting the amount of information. In the maritime world this concept is known as 'Dead-Reckoning'.

The insights presented in the past subsections are added to the requirements already known as an extra requirement, leading to the combined requirements presented in the next section.

C.4.3 Combined requirements for the architecture

This section concludes the architecture requirements by combining the insights from the different scientific disciplines. The following list summarizes theoretical requirements for the solution, which will be confronted with the architectural concept in the next section:

1. Decision Support in a multi-actor environment
   - An actor should be able to move the boundaries of his/her observable system but only as far as the relevant (system-) administrator allows
   - An actor should be able to monitor the state of the system and be notified by a system of relevant events but only as far as the relevant (system-) administrator allows
   - System information should be visualizable so it gives the actor insight and relevant information that fits his/her problem perception
   - An actor should receive formalized information, so he/she can use this information directly for further processing
   - An(y) actor should be able to act on the system if granted the proper rights by an administrative authority

2. Organizational / Business Engineering
   - Support multiple actors, actor-empowerment
   - Flexibility to meet dynamics of daily practice
   - Minimize both communicative as administrative overhead
   - Architecture-extendibility to 'Model-based generation of exceptions'
   - Security, Authorization
   - Open standards

3. Systems/Software Engineering
   - Support interaction with legacy-devices, different architectures
- Support subsidiarity with respect to storing and forwarding information
- Support event-based paradigm next to actor initiated requests
- Extendibility, flexibility
- Architecture scalability

C.4.4 Confronting the architecture with theoretical requirements

This section confronts the architecture as described in this thesis with the theoretical notions and requirements as presented in the previous section.

1. Decision Support

- **An actor should be able to move the boundaries of his/her observable system, but only as far as the relevant (system-) administrator allows:** Actors have the possibility to indicate their interest in particular virtual devices and query their state if the valid rights are available.

- **An actor should be able to monitor the state of the system and be notified by a system of relevant events, but only as far as the system administrator allows:** An actor with proper rights can have a filter instantiated to monitor the particular virtual devices in the system and be notified of relevant (filtered) state changes.

- **System information should be visualizable so it gives the actor insight and relevant information that fits his/her problem perception:** The architecture supports various means of display for business events and visualization in the architecture.

- **An actor should be able to use/process the information he/she receives in a formalized way:** The concept of the virtual actor allows specific the virtual actor to be ‘loaded’ with specific functionality which can interpret incoming business events in an actor-specific and flexible way.

- **An actor should be able to act on the system:** An actor with the appropriate rights can issue business actions to change the system’s state.

2. Organizational and Business Engineering

- **Support multiple actors, actor-empowerment:** The architecture allows multiple instances of a virtual actor to be initiated, each with fully independent rights to interact with the system. The architecture allows actors to take the initiative in both querying and acting on the different virtual devices as well as placing listeners for information that is particularly relevant to them.
• **Flexibility to meet dynamics of daily practice:** The modularity of the components, as well as the limited adjustments required to add functionality enable the architecture to change rapidly to new situations. System modifications deployed within minutes throughout the entire network after the proper rights have been granted by an administrator.

• **Minimize communicative and administrative overhead:** The concept of requesting rights in a formal way has the potential to drastically shift workload from the administrator to the requesting actor, while at the same time maintaining the flexibility for an administrator to "fine-tune" the rights he/she distributes. Since parts of the administrative work is shifted from administrator to actor there is a dampening effect on any administrative bottleneck.

• **Architecture-extendibility to 'Model-based generation of exceptions':** The way filters in the architecture have been conceptualized as separate objects which can be reside in different organizational domains and the fact that their functionality is extendible, allows them to be specialized to contain prediction models for any preferred part of the value-chain.

• **Security, Authorization:** The use of a public key cryptography not only allows messages to be sent securely from one component to another, it also facilitates the introduction of formalized requests for rights and their advantages, allowing rights to be distributed to authorized actors within and between networked organizations.

• **Support open standards:** The requirement of using open standards is met by the use of the Java programming language, open, documented interfaces for the architecture’s components and use of the XML-standard in the reference implementation.

3. Systems/Software Engineering

• **Support interaction with legacy-devices, different architectures:** The flexibility of the component-structure allows various communication-sub-components which can support various messaging-paradigms to be inserted at will. The separation between a real-device and its virtual representation allows for numerous of different architectures to be connected to the architecture. If a connection between a real-device and its Java-based virtual representation can be made, the device can be a part of an architecture instantiation. The support for various ways of sending and receiving messages (push, pull, a-synchronous, synchronous) makes the architecture able to interact with legacy-devices and other architectures.

• **Support subsidiarity**\(^5\) with respect to storing and forwarding information: Setting the right (limited) amount of filters keeps information

\(^5\)See the definition in appendix A.1 on page 78
as low in the network as needed to answer questions of different actors, preventing it to be sent to large data warehouses from where it is never used.

- **Support event-based paradigm next to actor initiated requests (see also Decision Support):** The introduction of filters which listen to virtual devices for relevant property changes allow actors to be notified without them having to query the system every other time.

- **Extendibility, Flexibility:** The component-based, Object Oriented architecture allows functional sub-components to be replaced during operations, as well as new components to be extended from components with existing functionality.

- **Architecture scalability:** The possibility for the introduction of redundancy in the network allows for scalability beyond the capability of single components. Scalability is supported by the use of a distributed solution which, is much easier expanded by additional computing power. Its scalability allows the architecture to be used both in a demonstration setting, as well as in a real time business environment.

As a result of the confrontation of the architectural concepts it turns out the architecture also meets the theoretical requirements.
Appendix D

Concepts on Decision Support, Systems Engineering

This chapter gives background information on the field of decision support and systems engineering. It sketches a mono-actor model used as a way of thinking about what actors need to influence a system. The multi-actor generalization of this conceptual model lies at the basis of the conceptual architecture which is presented in this thesis. The second section elaborates on the way of asking questions using an event-based concept for receiving answers and gives some background on the theory of subsidiarity.

D.1 Decisionmaking theory

Since information systems must be able to support decision making this section theoretical notions on decisionmaking and their implications for an information infrastructure. A model of decisionmaking is introduced that focuses on decisionmaking as a (continuous) process driven by a certain problem perception. The section concludes with some distilled requirements for an information system providing support for the process of decisionmaking.

D.1.1 Problem driven approach

In [Bots, 2001] decisions are thought to emerge out of an underlying problem perception of an actor, the problem owner. It therefore introduces an analytical model explaining the relationship between a single, rational actor, a problem that is perceived by this actor and the relationship this actor has with regard to the observable system in which the problem occurs. A graphical view of this model is presented in figure D.1 [Bots, 2001]. This model assumes that the world can be regarded as a system, a collection of elements and relations surrounded by an environment. The model illustrates that the problem owner should:

- Be aware of his problem perception. The problem perception may for example be formalized by means of an objective hierarchy or means-ends model.
• Observe a *specific* part of the world, which is bounded by and related to the problem perception. This part of the world will be identified as the *relevant* system. The whole system that can be observed by the actor will be identified as the *observable* system.

• Be aware of the way instruments and environment influence the relevant system. This implies that a *model* of the relevant system is available for the problem owner to enable him to *predict* in advance the consequences of a certain course of action for a given environment behavior.

• Have enough *instruments* to act on the system. The problem owner should at least have enough instruments at his disposal to counterbalance the environmental impacts on the relevant system.

The aforementioned consequences correspond with the five necessary conditions for effective control:\(^1\):

• Clear objectives of the control process.

• Availability of a model of the controlled system.

• Available information on the state and environment of the controlled system.

• Availability of enough steering instruments.

• Availability of sufficient information and communication handling capacity.

\(^1\)see [Leeuw, 1990]
The last condition follows from the need to process state changes of the relevant system fast enough in order to be able to apply suitable control mechanisms in time.

D.1.2 Requirements for an information architecture

The illustrated decisionmaking model implies the following consequences for an information architecture providing decision support:

- The actor should be able to put *boundaries* on the observable system in order to limit the amount of information received by the actor. The need for such a limitation of the received information is shown figure D.2. The actor is only interested in the information in the relevant system and therefore needs to filter the superfluous information. The theory of systems suggests several dimensions for bounding systems, among which the following:

  - Definition of a subsystem, consisting of a sub-collection of elements of the original system (including the attached relations). The elements of the subsystem are usually characterized by a certain condition. All elements may for example be located in a specific geographical area.

  - Definition of an aspect system, consisting of a sub-collection of *relations* of the original system (including the attached elements). The relations of the aspect system are usually characterized by a certain condition. All relations having to do with information flows may for example be included.

  - Definition of a phase system, in which a system is observed during a certain *time-interval*. The chosen time-interval is usually defined by a certain condition, for example maximum load on a system.

- The actor should be able to *monitor* the state of the system and its environment in order to be able to react accordingly, given the problem perception.
• The system information should be visualizable in such a way that it gives the actor the most insight in the relevance (and significance) of the information for the problem perception. In this way, the information and communication handling capacity of the actor may be leveraged. Since problem perceptions as well as information types differ, various ways of information visualization should be available for the actor.

• The actor should be able to use the information available of the observable system as a basis for a descriptive or predictive model of the system. If this model is to be a computer model, this implies that information is available in a general computer-readable form.

• The actor should be able to act on the system by use of his instruments. The information system could support this for some instruments by sending control information to parts of the relevant system.

D.1.3 Usefulness of formal models for decision making

The theory on decision making as described in section D.1 on page 97 holds a rational/hard view on the process of decision making. In practice no actor can rely solely on automated decision support systems, but will also use all kinds of 'soft' or non-formalized information before making a decision. Soft information can very well be any of the following:

• a rumor which the actor overhead down the hallway

• the declining economic situation as presented on yesterday evening’s news

• any event beyond the scope of what has been formalized in the system

This thesis does not want to make claims about:

1. the quality a decision as a function of the amount of information used

2. the quality a decision as a function of the percentage of formalized information used

D.2 Event-based way of asking questions

Actors in every function should be provided with exactly as much information as is necessary for their task or functional description. Both too much information as too little information can negatively influence the quality of a decision. If we assume for a moment that information on the system is fully formalized and fully available than each actor has specific work-related questions for the observable system. Questions can be translated in formalized questions to be answered by an business information system.

Some examples of questions relevant for different actors are:

1. logistics manager - Of which products is the amount in stock less than 25?
2. *general manager* - Which business unit was most productive last year in terms of throughput?

3. *financial manager* - Which 4 products contribute most to our revenue?

Nowadays the previous questions are generally answered by firing queries at a large enterprise database which can take many system resources to process and come up with an answer. The previous questions can also be formulated a little bit differently:

1. *logistics manager* - Please notify me when a certain product reaches a stock level of less than 25.

2. *general manager* - Notify me when productivity for any business unit drops below 85

3. *financial manager* - Notify me when one of the top 4 products (revenue-wise) undergoes any changes.

The second series of questions is of a different nature. Instead of putting the initiative with an actor who has to query the system, the initiative is with certain components in the system to to listen for pre-defined conditions and notify an actor of action is required. This event-based approach allows actors to concentrate on relevant issues and be notified the moment that their attention is necessary. A figure depicting a situation where an actor takes action after being notified of a situation which requires his/her attention can be found in figure D.3.

### D.3 Theory of Subsidiarity

This section theorizes on the subsidiarity when it comes to storing and forwarding information. The theory of subsidiarity is defined as the following:

**Subsidiarity:** Subsidiarity is the principle that decisions and responsibilities should lie as low down in the system as possible. [Handy, 1998]
The theory of subsidiarity says that what can be done at a particular hierarchical level should be done there and not be brought up to a higher level. Although this might sound like stating the obvious, it is an important principle for thinking about responsibilities and the question where to store information in an enterprise information system. There is an analogy with the theory of subsidiarity and the way a modern networked organization should collect and handle information. Whether a hierarchical model or a networked approach is used, information should not be moved to places where it is not needed. Information which is not used by an actor for his/her process of decision making should not be presented to this actor. It is humorous to note that even the clergy has an opinion about the concept of subsidiarity:

It is an injustice and at the same time a grave evil and disturbance of right order to assign to a greater and higher association what lesser and subordinate organizations can do.

Pope Pius XI, 1931
Appendix E

Concepts on Administration, messages between systems

This chapter presents a new paradigm in administration of systems in a distributed environment and explains the difference between synchronous and an a-synchronous communication.

E.1 A new paradigms in administration

This section presents a new paradigm in systems administration. It proposes that instead of requesting rights through a traditional, non-formalized means like e-mail the person who would like to have the rights should submit a request to an administrator in a formalized way to minimize administrative and communicative overhead in the network. First some background on the traditional way of distribution of rights is presented and continues with describing the concept of requesting rights in a formalized way. Various benefits of this new approach are presented.

E.1.1 Traditional distribution of rights

Currently, when a user wants rights to use a system in a particular way he or she would (mostly) send an e-mail to ask the system administrator to grant these rights. The administrator reads and validates the request and if he decides to grant the request, he/she updates the database tables where the user rights are stored. The administrator has to spend a considerable amount of time mapping or parsing the written (un-formalized) request to a (database) record of some sort, which reflects the granted rights. Consequence of this approach is that more often than not it takes quite some time before an administrator has handled the actor’s request due to difference in perceived importance between the requesting actor and the administrator.

\[\text{(1) cases are known where it took over two months before the proper rights for a new project member were available}\]
E.1.2 Requests for Rights in a formalized way

To ease system administration a concept is presented were requests for certain rights will reach the administrator in a formalized way (see figure E.1 on page 104). The user who wants to request rights accesses a tool in which he/she indicates the rights that he/she requests, by choosing them from various pull-down menu’s. Next, the user clicks a 'submit for approval'-button which sends the request to the administrator’s requests-inqueue. After that, the administrator logs on to his system and evaluates the submitted request in his inqueue. To grant or deny he/she can respond with a single mouseclick instead of spending several minutes translating the un-formalized request into a tuple in the administrative database. Needless to say the use of this paradigm for the distribution of rights requires an administrator that knows what he/she is doing and makes no mistakes.

E.1.3 Automatic generation of requests

In the event that an actor tries to issue an action for which he/she is not authorized, the component on which the action was to be executed, the filter or business rule will notice and automatically generate an appropriate Request For Rights which allows the particular action. The Request For Rights will then be sent to the administrator for approval, after which the actor receives the ticket back from the administrator. The actor will then automatically forward the signed request to the interactive component which he/she initially sent the action to. From the administrator the ticket is sent via the actor to give the (virtual) actor the possibility to store the rights which have been granted to him/her.

This new way of administration has the potential of dissolving and shifting the boundaries of between different actor’s functional descriptions, since one actor without proper rights can use this paradigm to propose actions in a formal way to the actor/administrator who is entitled to execute/grant these actions.

E.1.4 Granting rights only once

Once a user has been granted rights to perform some action on a system, there is no need for administrator interference the second time the user wishes to perform a certain action for which rights are required. If the user still has a
valid ticket, this can be used without administrative interaction. Unless the actor has lost his ticket, it is not necessary to contact the system administrator again. The virtual administrator might also contain a component\(^2\) which stores granted rights from which requests can be re-issued without human administrative interference, if a virtual actor for some reason lost granted tickets.

### E.1.5 Right decisions with the right actors

Administration of business IT-systems is currently done by people trained for systems administration. Many of the issues system administrators are confronted with however, have an impact on business aspects. The following exemplary questions are in some situations answered by (systems) administrators but should arguably be answered by an employee with management authority:

- Who has access to what kind of information within the organizational domain?
- Based on what thresholds or conditions more in general are purchase orders for new inventory placed?

It is argued that systems administrators should not take decisions that influence the way a company does business. The new way of distributing rights covered in this section allows the system to be configured in such a way that the right people in an organization take decisions.

### E.2 Synchronous vs. Asynchronous communication

This section highlights the difference between of synchronously and asynchronously communicating systems. With synchronously communicating systems the interface defines the functionality a component offers to the outside world. Systems using asynchronous communication interpret incoming messages by some special handling component.

#### E.2.1 Synchronous communication

Components that communicate synchronously offer their functionality through their interface, separating specification and implementation. When designing a system of components which communicate synchronously the developer decides upon the functionality of components (their interface) and how and when they call each other’s functionality.

#### E.2.2 Asynchronous communication

With asynchronous communication a component’s interface to the outside-world is much less important that the way it handles and processes incoming messages. The only method that is available will usually be IncomingMessage(Message message). Much more interesting is to see how a component pro-

\(^2\)linked to a authorization database
cesses an incoming message and handles its requests. Here not only the structure of the message is important, but also of the component which processes the message. Diagram E.2 on page 107 shows the inner workings of the messaging component which handles incoming JMS-messages from the reference implementation.

E.2.3 Messaging component reference implementation

Where in a system with synchronous communication functionality is defined by the components’ interfaces, in an asynchronous system functionality is defined by the structure of the message and the structure of the component which parses the message. A reference implementation of a message parsing component is added as figure E.2 on page 107.

Figure E.2 on page 107 should be read top-down, starting at the incoming-message-box. If a message is encrypted it is first de-encrypted after which it is evaluated again. If the message is digitally signed the signature will be evaluated to see path has to be taken next. An admin-signed object goes left, an actor or virtual device-signed object takes the path to the right. The path is followed until a one a box which indicates which functionality is called has been reached.

The description of the messaging component in this chapter should be seen as a reference implementation, which since the definitive conceptual architecture was changed in a minor way. See [Richard Monson-Haefel, 2001] for more information on how to implement asynchronous messaging in the Java-language using a Java Message Server.
Schematics for Message Handling by a NaradComponent
"Virtual Device instantiates Filter" reference implementation

Figure E.2: Message Handling Component, reference implementation
Appendix F

Concepts on filters

This chapter goes into more detail on the Filter-component. It describes how a filter should be initialized, what information should be sent by a filter and how one filter could transcend multiple organizational domains.

F.1 Filter instantiation; various conceptual alternatives

This section analyzes the consequences of different design-alternatives of initialization and ownership of a filter. When answering the question which component should instantiate and take responsibility for a filter there are typically three possibilities:

- a Virtual Actor instantiates a Filter
- a Virtual Device instantiates a Filter
- a Virtual Administrator instantiates a Filter

Each of these alternatives has its positive points and its drawbacks, which are covered in the following paragraphs. This chapter concludes that the only component which should be able to instantiate a filter (or a business rule), is a virtual component with administrative authority; the virtual administrator. If rights have been granted before, this can be done without any human interference.

F.1.1 Virtual Actor Instantiates Filter

The virtual actor is granted rights, after which he instantiates a filter. The virtual actor sends the ticket to the filter and the filter uses these rights to place a listener on a virtual device.
The virtual actor instantiating a filter has the advantage that the virtual actor does not depend on the administrator for any task. He or she can instantiate a filter the moment he needs without having to contact the virtual administrator. Drawback of this approach is that the virtual administrator would have to allow a filter which he doesn’t know to be instantiated in the virtual administrator’s organizational domain or the particular filter could not live close to the virtual device it is listening to. It is clearly not acceptable for an administrator to have a unknown filter living within his organizational domain, nor is it a good idea to have property change events be sent over multiple organizational domains to the location where the actor instantiated the filter.

F.1.2 Virtual Device instantiates Filter

In theory a virtual device could instantiate a Filter if it receives a request for subscription to information. Having a filter instantiated by a virtual device has the advantage of having the functionality available right at the moment it is needed. Disadvantage is, that in cases where one filter is supposed to listen to more virtual devices in different organizational domains, it remains very much the issue which virtual device is responsible for instantiating the particular filter and whether that filter has been reviewed by the respective administrators. Apart from this issue, if a virtual device would instantiate a filter, actors would have to be able to interact directly with a virtual device.

F.1.3 Administrator Instantiates Filter

In figure F.1.3 the virtual administrator who is asked for rights instantiates the filter, after which the virtual actor sends the ticket to the filter. The filter
places a listener on the virtual device which notifies the filter of state changes. In this situation, the filter is available after the administrator has instantiated the particular filter. If an actor decides that he needs particular filter functionality he/she cannot use this filter without contacting the virtual administrator, which is a drawback.

F.2 Objective vs. Normative information in Business Events

When sending out a business event, it is important that a filter doesn’t just send normative information, but the full (objective) context as well. Just a normative message does not give the receiving actor enough information as a foundation for decision making. Suppose a (virtual) carwash should needs to notify the gas station clerk it is running out of soap. To do so, it can send a normative message, or a message containing objective information together with a normative interpretation. Let’s look at the two possibilities:

Example F.1 carwash1 at GasStation 1 reports soap level low

or

Example F.2 carwash1 at GasStation 1 reports soap level is 2 liter (low)

In the first case the actor which receives the business event must know what is meant by the normative interpretation ‘low’. Does it mean the carwash has enough soap left for 10 cars, for 5 hours, 3 days, or is it nearly empty?

When the carwash reports its soap-Level is 2 liter the actor who receives this message can interpret and decide\(^1\) for himself, whether it is necessary to take care of this right away, or whether there is still enough soap left to finish some other business first and refill the soap in a couple of hours; which really puts the decision with the actor instead of the automated system.

The benefit of implementing the system with objective messages versus normative messages is a two-edged sword. An actor must, in case of objective

\(^1\)of course, this assumes knowledge about situation and business process with the actor
messages, be able to make an interpretation based on the (current) soap-Level by him/herself. Is there still enough left, is it running low? To be able to put the objective business event in context knowledge about the carwash and what is a normal soap-Level is assumed with the actor. Of course, if an organization decides it doesn’t want certain actors to take responsibility themselves (by interpreting the objective business events), a choice can be made for sending normative events.

While sending objective messages has a the potential for process optimization by human interference, it might also have the opposite effect.

F.3 Data, information or knowledge?

Figure F.1 on page 111 describes a conceptual model about the relation between data, information and knowledge. It shows data formatted, filtered or summarized into information. Information in itself is interpreted, decided or acted upon. The result if these acts or decisions are accumulated into new knowledge.

An important design question is whether the information a filter sends upstream should be qualified, as data, information or knowledge (see F.1 on page 111 [Alter, 1999]). Since data doesn’t contain any context, it is not very useful to send just data. A virtual actor would not know what to do with bare numbers. A filter sends knowledge if the contents of the message a filter sends doesn’t need to be interpreted, but direct (human) action can be taken. If there is no need for human interpretation the process can be fully automated, instead of offering an actor the information in a half-automated solution.

Almost always a filter will send information after which the information is interpreted by a human (maybe helped by an expert system) to make a better decision than the filter can suggest based on its settings. See for an elaboration on normative vs. objective events appendix F.2 on page 110.

F.4 Cooperative Interactive Components in a Trusted Environment

If an actor has a question which requires processing of information from multiple organizational domains and the administrators from these domains do not allow
each other, or the actor to see the actual information a processing filter can (for security reasons) not be placed with one of the organizational domains. This problem cannot be solved without the introduction of a third party trusted by the involved administrators (see figure F.2 on page 112).

The protocol for filter-initialization (and the RequestForRights-object) will be slightly more complex than for a single filter as depicted in figure F.1.3, since rights have to be requested for all the different virtual devices which are involved. The protocol for a one of these intra-organizational filters is depicted in figure F.3 on page 112.
F.4.1 Example of components in a trusted environment

One example of such a situation is where a real estate agent would like to know whether its client is financially credible for a certain amount of money, for which the answer is either true or false. To get an answer to this question, let’s assume information (the account balance) from the different financial institutions must be combined (added) and the total answer is true if the total sum is larger than $x$ and false if the total sum is smaller than $x$.

None of the financial institutions are likely to be willing to share their client’s balance with anyone, although they might be willing to answer (return yes or no) a credibility-question from a trusted partner. Problem in this particular case is how information from all the different financial institutions is combined without revealing their substance and yet still the real estate broker is given an answer to his question regarding the credibility of this client.

A relevant question is: How are different organizations with each their own responsibility and wish to restrict access to sensitive information going to cooperate to answer questions? In the next subsection a solution to this problem is proposed by the introduction of a trusted third party where the reviewed filter will reside.

F.4.2 Introduction of a Third party

In case the administrators of the respective organizational domains can be persuaded to share their information but they want to have a saying how this information can be used a reviewed filter or business rule in a trusted environment can be used. The interaction component would be engineered by developers from all the different organizational domains. Each of the developers would ensure that the only information that is sent to an actor who asks a question to which the answer yes or no. Sensitive information like the balance is not shared beyond the server of this trusted organization.

When engineering such a cooperation between multiple business partners it is of considerable importance to create incentives for all the parties to participate.

It is assumed that there is less of a threshold for organizations to share their sensitive information if they can restrict how this information can be used, and if they get either money or information in return.
Appendix G

Concepts on Security and Cryptography

This chapter gives background information on various types of cryptography, mentions models for establishing trust in a network and looks into the various relevant aspects of security.

G.1 Concepts of Cryptography

This chapter explains the three types of secret writing on a conceptual level. It starts with cryptography without a key, is followed with secret key cryptography and ends with public key cryptography. Public key encryption is used in the architecture to authenticate users and grant actors various rights to interact with the system. For more information on the concepts of cryptography, see [Phil Zimmermann, 1998]. For specific cryptographic implementations in the Java-language, see [Knudsen, 1998].

G.1.1 Introduction

Cryptography or the art of secret writing has intrigued many during the course of history. Its concept is depicted in figure G.1. A readable plaintext is encrypted using some kind of algorithm to an unreadable ciphertext. It is not possible to
recover the original text from the ciphertext (at least in theory) other than through decryption.

G.1.2 Cryptography without a key

Cryptography without a key or 'security through obscurity' is based on not-telling how the conversion from plaintext into ciphertext is done. Probably the most famous example of keyless cryptography was developed by Julius Caesar’s who, when sending messages to his generals, used a shifted alphabet. When one shifts the alphabet by 3 characters, A becomes D, B becomes E, etc. Whoever knows the method of encryption/decryption can use this method to decrypt the ciphertext; no additional information is necessary. Security through obscurity is quite susceptible to simple analytic attacks and is therefore not regarded a good solution for securing sensitive information.

G.1.3 Secret key cryptography

Secret key cryptography or symmetric cryptography uses the same key for encrypting the plaintext to the ciphertext as for decryption of the ciphertext to the plaintext. For decryption, apart from knowing the method/ciper used, the decryptor needs the key which was used for the encryption of the plaintext. See figure G.2 where key A is used both for encryption as for decryption. This method of encryption leaves the problem of key management when two parties who have never met want to exchange information secretly.

G.1.4 Public key cryptography

With public key or asymmetric cryptography a different key is used for encrypting a plaintext to it’s ciphertext as for decryption of a ciphertext to the
plaintext (see figure G.3).

A sender encrypts a document to the public key of a receiver after which no-one else not even the sender but the receiver (because he/she is the only one who holds the private-key) can decrypt the message. A public key and it's private key are mathematically related. Details about the mathematics of asymmetric cryptography go beyond the scope of this document.

**Digital signatures** A useful characteristic of public key cryptography is that it cannot only be used for encrypting a document for a particular receiver, but also for digitally signing information. Instead of encrypting to another actor’s public key, the signer signs with his own private key. Since every actor has access to every other actor’s private keys everybody can verify that the information was signed by the signer and was not tampered with. A digitally signed document cannot be changed by a third party without detection by the receiver.

**Authorized formalized request** The concept of digital signatures is used in the architecture to indicate that an administrator grants rights for a particular formalized request. The signature can be seen as the fact that the signed authorizes the signed object. The signed object can be used in automatic processes to show proof of authorization. The receiving component can verify instantly that the request if valid.

### G.2 Trust models

Using cryptographic techniques three different models of trust between parties can be constructed:
• Direct Trust: Certificates based on which one party trusts another are handed over directly.

• Hierarchical Trust: Trust is derived in a hierarchical, top-down way.

• Web of Trust: Trust is asserted in a bi-directional way.

The hierarchical method of distributing trust is generally used between organizations. This model as a couple of disadvantages:

1. Certificates from Certificate Authorities are not free of charge

2. The process of establishing trust through a centralized Certificate Authority generally takes more time than using a Web of Trust certification.

Although trust is not implemented in the architecture at this moment the concepts behind the architecture suggests the use of the Web of Trust-paradigm for establishing trust in the distributed environment. This paradigm is more flexible than the hierarchical paradigm and can easily be used in a strictly hierarchical way if needed. For a more extensive description the reader is pointed to [Phil Zimmermann, 1998]

G.3 Security

When designing a secure system different aspects should be to taken into account, especially when a system is to be deployed on a public network. For more information on implementing secure Java-applications in distributed networks, see [Oaks, 2001].

G.3.1 Different dimensions of security

When discussing security of a communication-system there are three aspects that together make a solution secure. For the three relevant aspects of security several design questions will be answered to dimension the required level of security in the architecture.

1. Confidentiality

• Is the fact that an actor subscribes to a filter considered secret information? If the actual contents of the message should remain a secret, encryption of the message is a solution.

• Is the fact that an actor sends a message to a filter considered secret information? Although some might regard this as quite ridiculous, it is conceivable that the fact that an actor is interested in particular information is considered secret information. Implementing this kind of security is not possible within the boundaries of the architecture. A solution where bogus-messages are sent every couple of seconds is not feasible, considering the large number of objects in the chain.
• Is the business event that a filter sends back to its subscribers considered secret information? If the actual contents of the message should remain a secret, encryption of the message is a solution.

• Is the mere fact that an actor receives a business event from a filter considered secret information? Again, although some might regard this as quite ridiculous, it is conceivable that the fact that an actor is interested in particular information is considered secret information. Implementing this kind of security is not possible within the boundaries of the architecture. A solution where bogus-messages are sent every couple of seconds is not feasible, considering the large number of objects in the chain.

2. Data Integrity

• Is there a need to protect business actions or business events against modification along the line? In the past false messages and information have brought companies to bankruptcy, so there is a definite need to protect messages from tampering when they are underway between sender and receiver. In the architecture, messages are protected against tampering by digital signatures.

3. Authentication

• Is there a need for actor authentication when communicating with the administrator, a virtual device or filter? Considering the open structure of the architecture, authentication is of major importance. Authentication is taken care of through the used of digital signatures.

• Is there a need for virtual device authentication when communicating with an actor? The general answer regarding to authentication is yes. If a message is not authenticated a malicious actor might send bogus-business events to the virtual actor.

Since authentication makes up a such vital part of the concepts of the architecture and how incoming messages are handled by the messaging component it is standard in the communication. Dependant on particular use of the architecture encryption of messages can be added if needed; for instance when used for communication over an open network. Encryption and decryption of messages comes at a cost of computational resources. Whether or not this is acceptable is specific for any given implementation and needs to be evaluated on a per business case basis.

G.3.2 Security at connection level vs. Security at message level

When implementing a security solution there are different levels at which the necessary cryptography can be implemented. Two options are at the connection level (through a VPN-connection for instance) or the individual message level. Encrypting messages at the connection level does supply confidentiality and data integrity, but not authentication. This could be solved using a
username/password combination, but authentication at connection level does allow differentiation in access rights within a domain. Using a domainmanager/gatekeeper with complex functionality is a less feasible solution to solve this problem.

G.4 Architecture security

The implemented amount of security should depend on the particular architecture implementation. For instance, when deployed on an company-intranet, encryption of messages might not be required and can be refrained from for reasons of system performance. Based on answers to specific design questions a business case-specific choice for a level of security can be made.
Bibliography


