DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS C
REPORT C-2011-X

Trust and Mistrust Management in Enterprise Systems

Lea Viljanen

UNIVERSITY OF HELSINKI
FINLAND

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Lea Viljanen

Työn nimi — Arbetets titel — Title

Trust and Mistrust Management in Enterprise Systems

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Licentiate thesis	January 2011	121

Tiivistelmä — Referat — Abstract

In the near future world of dynamic open service ecosystem, where enterprises collaborate with various partners to further their business goals, there are new challenges in managing threats such collaboration brings, especially malicious peers. Traditional security solutions relying on restricting access on individual partner basis are not feasible in this open dynamic environment. Instead, the new community security requirements can be better met with new detective controls utilizing high level information.

This thesis proposes a model where trust is utilized as the community member security solution. The solution consists of trust-guarded authorisation augmented by peer behaviour monitoring and continuous feedback for mistrust management. First, an extensive survey of trust management and an ontology built based on that shows how different types of trust can be utilised in enterprise computer systems and how trust-guarded authorisation is a valid realisation of the theoretical trust formation process. Second, for mistrust management this thesis surveys existing application intrusion detection mechanisms to find suitable mechanisms for peer behaviour monitoring and feedback. Thus this thesis combined two previously well known and separatebly implemented components in a novel way.

The final contribution of this thesis is the TuBE community member security framework design based on the model above. The design shows how i) the trusting decisions are instrumented when an enterprise is engaging in collaboration, ii) the intrusion detection type monitoring is used at each service participant, iii) the collaboration can be terminated or restructured based on an observed intrusion or policy violation, and iv) how the observation affects the reputation and future trust decisions for the violating participant.

Computing Reviews Classification:
C.2.3 COMPUTER-COMMUNICATION NETWORKS: Network Operations: Network Monitoring,
H.3.5 INFORMATION STORAGE AND RETRIEVAL: Online Information Services: Web-based services,
K.6.5 MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS: Security and Protection: Unauthorized access

Avainsanat — Nyckelord — Keywords
enterprise computing, authorization, trust, intrusion detection

Säilytyspaikka — Förvaringsställe — Where deposited
Kumpula Science Library, C–2011–XX

Muita tietoja — Övriga uppgifter — Additional information

Contents

Chapter 1

Introduction

Advances in computer networking during the recent decades have brought unprecedented opportunities for organisations and individuals to connect with other organisations and people first locally and then globally. This has created new ways of interacting and conducting business in the global business world.

It is the view taken in this thesis that the emergence of automated business networks consisting of independent networked organisations more or less automatically soliciting and trading their services is just around the corner. This form of business automation will have a major impact on the global economy because the networks or communities will reduce the impact of physical distance in commerce, utilise economics of scale to drive costs down, and enable organisations to be more agile and thus successful in the modern rapidly changing world. It is assumed that in the future business networks will form automated ecosystems consisting of for example buyers, sellers, brokers and subcontractors acting for the benefit of all participants.

For this world view to fulfil its promise, organisations need open, robust, scalable and secure frameworks on top of which these business network ecosystems will be built. Especially security is of key interest since there have been several recent security breaches where huge number of consumer credit card numbers have been lost to criminal elements. These have caused governments and key business drivers to start setting legal and compliance requirements for networked businesses and services.

From the security point of view, there is a dilemma especially in networked computer systems when interacting with others. On the one hand, the remote party can benefit the community participant by placing orders on goods or services. On the other hand, the remote party might place a fraudulent order or try to break the security of our open system, thus causing the community member harm. So we have two contradictory requirements. If we want to sell our wares or services in business networks or take part in any collaborations, we have to open our services and make them available to others, thus placing trust on third parties. But at the same time, we also must be vigilant to protect our assets from harm, even from peers we trust.

How can trusting and not trusting be managed the same time during collaboration? This question is especially important in environments designed for collaboration between enterprises such as business networks or communities. It is also a challenging goal in current systems, since many collaboration and security technologies have been built on binary principles: if we trust a peer then we always allow access, if we do not trust the peer we do not allow any access ever. These trust relationships, or authentication and authorisation rules derived from them, are currently mostly static and manually administered. However, in a dynamic environment, where business network participants, such as buyers and sellers, can change sometimes even rapidly, static and manually managed systems cause scalability and maintenance problems. Therefore, statically

configured access and authorisation rules are not a suitable solution for business network and communities.

The key question for this thesis is thus, how can a collaboration framework for enterprise co-operation be protected without sacrificing dynamism and scalability required in the business world of tomorrow? There are three distinct challenges within this question. The first challenge is one of dynamism, how can the collaboration be protected in a world where open collaboration and dynamic participation over open networks is the future? The second challenge is the question of protecting members from malicious peers, how can we find out if a previously well behaved community member suddenly turns malicious and what to do after finding out? The third challenge is the cost of management, how can the protective framework be structured so that the daily operations can be automated enough for the solution to be cost effective to deploy in a scalable manner?

The answer to the key question in this thesis lies between the computer science realms of collaborative networks and information security. It is evident that the traditional security models and techniques focused on securing transactions between two known parties need some refinement in the community setting. Therefore this thesis focuses on a new information security area, *community security* and its subfield *community member security*, both of which require novel solutions. The aim of this thesis is to explore and evaluate the concept of trust as a solution for the key community security question above and the three challenges within.

Trust is already one of the key forces in today's society. A good example is the monetary system, where tangible goods are exchanged for pieces of paper or bits of metal, which do not have corresponding material value. The value is created by our trust in a system where the paper or metal can be exchanged again for goods or services of comparable value elsewhere.

Trust is also a major factor in the daily interaction between people. In every encounter we have with other people, we subconsciously assess the trustworthiness of the other persons and dynamically adjust our behaviour and attitude accordingly. This is especially important in situations where the trust evaluation we make results in collaboration where we are subject to harm if our assessment of the trustworthiness of others is wrong.

A key observation in human behaviour is that the trust relationships are not static. We interact with people daily but simultaneously and continuously assess their trustworthiness. If something unexpected happens, we readjust our trust expectations and our behaviour to suit the new situation.

This readjustment policy may also be a good approach for trust in business network scenarios, since enterprise applications must continuously interact with other applications as well. In recent years it has become evident that the current practise of evaluating trust only the first time the parties interact is not sufficient trust management: it is also very important to continuously and dynamically evaluate trust and its related aspects of mistrust, distrust and untrust while collaborating.

How do we then translate the dynamic human trust to computing concepts so that it can be applied as a solution to community security needs?

Several key elements are necessary. First of all, community participants need to be identifiable, i.e. authenticated. Second, we need a trust evaluation process to help us decide if the collaborative action is allowed to proceed. Hence, in a computer system we must make decisions based on peer behaviour and trust evaluation, trust-guarded authorisation. Third, we need to continuously monitor how the collaboration proceeds to detect any problems or anomalies due to misplaced trust. Finally, we need constantly readjust our trust valuations to take account past peer behaviour. Of these, this thesis assumes the first one solved and focuses on the less researched last three elements. These elements need to be bound together in a comprehensive trust management solution for community security.

The contribution in this thesis is to show how a trust management system well grounded in trust theory can be used as a community security solution, especially for community member security needs. The three focus components discussed above have been bound together to a concept framework called TuBE, which consists of trust-guarded authorisation enhanced by semantic information, introspective monitoring and dynamic trust value readjustment. Moreover, this work identifies several interesting avenues of research that should be explored later by other research.

This thesis is organised as follows. First, the next chapter introduces the collaborative enterprise environment and its threats to find and clarify the the research questions in the focus of this thesis, and solution requirements. Next, in Chapter 3 the concept and theory of trust and its management is explored to make sure our understanding and subsequent development of the key components of trust-guarded authorisation and continuous trust valuation is in line with previous research. Chapter 4 surveys different application level intrusion detection methods to find suitable candidates for our third component, the behaviour monitoring mechanism. Finally, in Chapter 5 the TuBE concept framework binding these three components together is introduced. In this chapter it is shown how TuBE can be used to solve the key research question and the three challenges introduced here. Conclusions and pointers to future research are presented in Chapter 6.

Chapter 2

The Collaborative Enterprise Environment

In the modern global business climate, enterprises have a need to form alliances to offer services to business clients across the world. This globalisation of business can manifest itself in many ways. One example of this can be seen in the move of physical production to countries with low labour costs. Another is the rising popularity of various outsourcing and off-shoring services. To successfully do business in this changing world, enterprises need to utilise this trend and start cultivating relationships with other organisations, which may be physically very distant from the enterprise home location. Additionally, if cost cutting is imperative, enterprises must continuously shop the market for best prices for their needs. All this requires the ability to flexibly and effectively manage a multitude of relationships with suppliers, business partners and other collaborators.

Traditionally, relationships have been managed with written contracts, which usually go through several review rounds with the respective legal departments. Additionally, purchasing and ordering processes have been paper oriented and required various signatures. However, if there are hundreds of relationships to manage, some of them potentially for a very short duration, this traditional process is too cumbersome and more cost-effective solutions are required. From there emerges the need for automated enterprise collaboration, where enterprises dynamically and autonomously form collaborative relationships and further their enterprises' business goals. The view taken in this work is that in the future enterprises base their businesses more and more on open service markets, where services are bought, sold and traded flexibly by autonomous agents for the benefit of participating enterprises.

There are of course many more forms of collaboration in the modern networked world. For example, recent years have seen the emergence of many collaborative *social networking* or *Web 2.0* services, such as Facebook, LinkedIn or Second Life. These services have two features in common. First of all, the users of these services are usually individual people[1]. Second, the users take active part in generating or being content in that service, i.e. there exists significant amount of user generated content, UGC. While these services are scientifically interesting from several viewpoints (such as sociological, economics and collaborative technology), this work focuses on collaboration and networking between organisations, not private individuals. However, the proliferation of these services based on social networks between people hints at a strengthening collaborative trend in the general atmosphere of the business world. For this trend to become

[1]There are of course exceptions. For example some companies and even countries have official presence in Second Life and organisations recruit staff in LinkedIn, but generally very few people in these services view themselves as official representatives of some organisation.

reality in business cases, a mature technological platform is required.

Since the field of business networks is relatively new, concepts and vocabulary are still somewhat fluid, but this chapter introduces the collaborative enterprise environment and its challenges and solution requirements.

2.1 Key Concepts

A *community* is a configuration of objects defined to express some common purpose or objective [ISO96b]. A similar concept is a *collaborative network (CN)*, which is constituted by a variety of entities (e.g., organisations and people) that are largely autonomous, geographically distributed, and heterogeneous in terms of their operating environment, culture, social capital, and goals, but collaborate to achieve common or compatible goals using a computer network [CMA05]. Collaborative networks can manifest themselves in many forms. Especially when the focus is on inter-enterprise collaboration, a community can be called a *business network* [KRRM08].

These definitions, while having some differences in vocabulary, have two things in common. First of all, participants are separate entities (objects, persons, organisations) and secondly these entities have some common or at least compatible goals they wish to further. The underlying assumption is that each entity participating in the collaboration benefits by it and thus investing in the business network participation either financially or mentally is worthwhile for all participants.

Since the vocabulary in the discipline of collaborative networking is still developing [CMA05], this work follows the terminology defined the ISO/IEC Open Distributed Processing reference model when available. Thus *community* is chosen as the key concept to describe this concept of general collaboration in this work. However, since the focus of this work is on enterprise collaboration, *business-network* can equally be used.

The community consists of *participants* or *members*, which are computationally represented by *network management agents* (NMA), *agents* for short [KMR06]. Community members can be organisations, companies, governments or sometimes even private individuals. To enable their community participation, a piece of agent software (the NMA) must exist to support them in the community operations. The agent should be independent enough to both reactively and proactively act on behalf of the organisation when necessary. For example, in a business community the agent should be allowed to make a binding sales offer if the tender query falls within acceptable limits. These constraints or limits in an NMA, or anywhere else in the community infrastructure can be expressed in a *policy*. A policy expresses a set of rules related to a particular purpose . A rule can be expressed as an obligation, a permission or a prohibition [ISO96a]. For example, an organisation may have a policy that it takes part only in communities where the initiator is located in a set of pre-defined trusted countries. Or a community level policy may prefer more trusted participants over unknown newcomers when selecting transaction partners.

Each participant fulfils a particular *role* in the community. Role is an identifier for a behaviour, which may appear as a parameter in a template for a composite object, and which is associated with one of the component objects of the composite object [ISO96a]. In a community these roles are shorthand labels for certain expected behaviour and are based on the common purpose of the community. For example, in a business transaction community there can be roles of buyers, sellers, brokers and payment verifiers. Associated with each role is a set of tasks, behaviour, that needs to be performed by an enterprise fulfilling that role. The roles and their interactions are defined in the *business network model* (BNM), which can also define restrictions, such as lists of acceptable peers for each role [KMR06].

The purposes of the communities can vary and therefore the number of roles in communities

can also vary. This variability of purpose also means that to fulfil greater enterprise level objectives, the enterprises must be able to participate in different business networks at the same time. Thus, to produce a good service a travel agent must be able to participate in several roles in several business networks, such as a buyer of holiday flights in one community and a seller of vacation packages in another.

Communities may have an *initiator* or *broker* who defines the purpose of the community and sets the rules of the collaboration [CMA01]. This initiation style requires very little negotiation between the participants, they either agree to the initiator's terms or not. Alternatively, the communities can be formed ad-hoc, where any participant may initiate a community and the collaboration is formed more dynamically. In this ad-hoc style, some negotiation may take place between participants to define the collaboration boundaries and interaction rules. When the community is formed, the participants may agree to a *contract*, which is an agreement specifying obligations, permissions and prohibitions for the objects involved [ISO96a].

Communities can also have a *coordinator*, who will be the regulator component of the community related activities and responsible for the global goal. The coordinator is either a node specialised in coordination and added to the community, or its role can be played by an already existing community member [CMA01].

Community purpose can be described as an activity, a process. There are several challenges in describing the community process, for example the process must be modelled, incorporating correctness of executions with respect to the model, and respecting the constraints of the underlying services and their resources. Second, we must be able to interface a process to underlying functionalities [HS05]. In the case of communities, this interface between a business process and the technical environment would include a suitable model of transactions that take account the distributed nature of the community.

The community process modelling can be made from several perspectives. Coordination of a data flow between community participants can be defined in a *choreography*, which views the community process as being a set of message exchanges between participants. The message exchanges are constrained to occur in various sequences and may be required to be grouped into various transactions [HS05]. An *orchestration* sees the same process as a program or a partial order of operations from the viewpoint of one single member and its orchestration engine [HS05].

How is then an electronic community built from these components and how do the participants interact autonomously? There are at least five different functional layers or levels that are necessary for interoperable electronic communities: technology, service, community, organisation and business levels [RK06]. This structure resembles a traditional network protocol stack, where each layer uses services from the lower layer to provide services for the layer above. These layers in a community setting are illustrated in Figure 2.1.

Within the lowest technology level, the basic communication interoperability is managed. This level includes the technical protocols and standards required for common network connectivity and data flow and could be implemented in a standard TCP/IP protocol stack with modern application level protocols discussed later. On top of that is the service level, which handles the semantics and behaviour of the service or set of software modules. Service level interoperability means capability of inter-operation between electronic services with well-defined, self-descriptive interfaces. Protocols on this level are interested in ensuring syntactic, semantic and behavioural compatibility of interconnected services. The community level functions manage the business network as a whole and may include protocols for negotiating and managing the business network model. Compatible failure-handling properties of the community are also essential. On top of this community level there is the organisational level, by which the business rules and policies for interaction are defined so that the automated community is compliant with applicable laws and business strategies defined

Figure 2.1: Community levels.

in the topmost business level. It is assumed that the higher the level the less automation there is. For example, setting business strategies is a task best defined in company board workshops, not by automated means. These strategies can then be defined for automation in the organisation level policies.

2.2 Types of Communities

As with the multitude of terminology, there are also a multitude of different types of communities. The communities can be taxonomically classified by several factors: duration, dynamism of topology, participation and coordination [CMA01].

- The community duration can vary. There are communities whose existence is for the duration of a single task or business opportunity. These communities are short lived and are dissolved when the business goal, such as the holiday sale or construction of an item, is achieved. On the other hand, there are communities that are long-lived and are more strategic in their nature. These long term alliances may be in existence during several actual business processes.

- Dynamism of topology differentiates the communities on how the members leave or join the community according to the phase of the business process. Some communities have an almost fixed set of members where very little variation is allowed during the lifetime of the community. On the other hand some allow for more dynamic approach, where members fulfilling a particular role can be easily switched. In other words, communities with a dynamic topology are more open to new members.

- Participation in a community can be exclusive or inclusive. While an enterprise may join several communities for different business purposes, there might be exclusivity clauses in

the community contracts that prohibit the members from participating in other similar purpose communities.

- Community coordination structure can be star-like or centralised, where a central authority, typically the community initiator or coordinator, dictates and enforces the rules. This pattern is found in cases where the community exists to serve the need of a single enterprise and the community is created to form a type of extended enterprise.

This classification does not take into account the existence of a formal contract between members. So additionally, the communities can be classified as *implicit* or *explicit*. Implicit communities do not have any high level contractual basis. There is no objective standard of behaviour but each participant evaluates the other participants' compatibilities against its own subjective behaviour standard. Of course this type of community still requires certain level of protocol interoperability and the ability of participants to fulfil the tasks for the roles, but these low level issues can be negotiated ad-hoc.

An example of the implicit case is the Internet electronic mail, where a system contacts another system to send e-mail. The purpose of that community is to facilitate sending electronic mail between the community participants. A system needing a message sent acts in a sending client role and a receiving system acts in an e-mail receiver role. Both have a common application protocol, Simple Mail Transfer protocol, SMTP [Pos82]. There is also a third role, a relay, which is assumed by receiving servers that can relay mail from sending clients to other servers if no direct connectivity exists between the sender and the receiver. Traditionally, most e-mail servers in the Internet have been also open relays to ensure fault-tolerant e-mail delivery. There has not been need for any explicit contracting about Internet e-mail delivery roles and responsibilities.

However, this openness and the informal relationships have their downside, the risk of abuse. These very informal and implicit contracts between e-mail clients and relays have in recent years been a factor in facilitating the unsolicited bulk e-mail (spam) problem by allowing abusive client computers to send millions and millions of spam e-mails via different relays. To counter this, quite many organisations deploying e-mail servers have now restricted the relay role to be assumed only with a particular set of clients, with whose peer they may have a more binding contract, such as an outsourcing or client agreement. In this sense, the Internet e-mail community has been drifting from an implicit community to a partially explicit community.

Depending on the type of the community, the technical requirements for the underlying support infrastructure can be very different. For example, a short-term and very dynamic community needs very flexible mechanisms for community population and renegotiation. And in the non-exclusive participation scheme there must be very strict information confidentiality enforcement across different but similar type of communities.

Especially in the ad-hoc implicit communities, current support infrastructure may not be sufficient, especially considering the misuse possibilities illustrated by the spam case above. There is a clear need for more explicit community definitions which include application behaviour descriptions. The inter-organisational application management is based on explicitly stated, platform independent federation contracts that define the forms of cooperation between performing components [Kut02]. Thus a community is specified via a contract on the members' collective behaviour detailing community roles, enterprise object assignments to roles, policies applying to the roles and behaviour descriptions. The term *e-contracting* covers various aspects of contract forming and use [MJDP02]. Aspects in a general contract management architecture include a contract repository, a contract monitoring facility, a contract notification component and other components such as contract enforcement [MLG+04]. Contract enforcement covers mechanisms for ensuring

that actual behaviour of parties governed by a contract is compliant with their expected behaviour stated in the contract [MJDP02].

2.3 Community Life-cycle

Communities have an operational life-cycle that has four phases: creation, operation, evolution and dissolution [CMA01]. These four phases include at least five management tasks: population, establishment, operations that include monitoring, reacting to change requests, and termination [KM05]. Thus the creation phase has two distinct management tasks, population and establishment.

The population process chooses community participants from a group of *peers* who are uniquely identifiable. Peers can be either people or organisations, i.e. principals, or autonomous agents acting for those principals[2]. The peers add participation offers into the offer repository and the population process selects a suitable offer for each role in the general *community model* or *business network model, BNM*, which is perhaps suggested by an initiator.

In an inter-enterprise scenario, the business network exists to fulfil a business scenario. This scenario is made public by publishing a business network model. The model captures all external business processes that are relevant to the business scenario. The BNM also gives structure to the electronic contract, which governs the community at run time [KRRM08].

In the population phase, if an entity can pick its business partners flexibly, it can select them to optimise any kind of quality-of-service criteria, including performance, availability, reliability, and trustworthiness [HS05].

The participants need to be interoperable and a negotiation process may occur to refine the suggested community properties in the community establishment task. To finish the creation phase means that all roles have been populated and members have both the technical preparedness to accept interaction, and have also completed all required contractual issues.

The second phase is the actual operations phase, where the common goal is realised. This phase may need support functions, such as basic secure data exchange mechanisms, information sharing and visibility rights support, orders management, distributed and dynamic planning and scheduling, distributed task management, high levels of task coordination and collaborative engineering support [CMA01].

During the operations phase there must also be the ability to renegotiate or change the contract if requested by one or more parties. This type of change initiates the evolution phase. Evolutions might be necessary during the operation of a VE when it is necessary to add and or replace a partner, or change roles of partners. This need might be due to some exceptional event, such as (temporary) incapacity of a partner, changes in the business goal, etc [CMA01].

The final phase in any community is the dissolution or termination, either when parties agree that the common purpose has been achieved, or when the contract has been breached either intentionally or unintentionally in such way that there can be no return to normal operations. The third option for community termination is a normal contract expiration, if the contract has an end date.

Within each of these community life-cycle phases there are interesting research questions. For example, what are the best community population strategies and what kind of methods and protocols are best in different kinds of contract negotiations? How can the community transactions be modelled either formally or informally? What kind of expert systems can be built for decision

[2]Despite ODP vocabulary, peer is preferred over principal as the actor noun also because it can not be used as an adjective and thus we avoid ambiguities. Client refers to a piece of software acting on behalf of the peer unless the discussion refers to business.

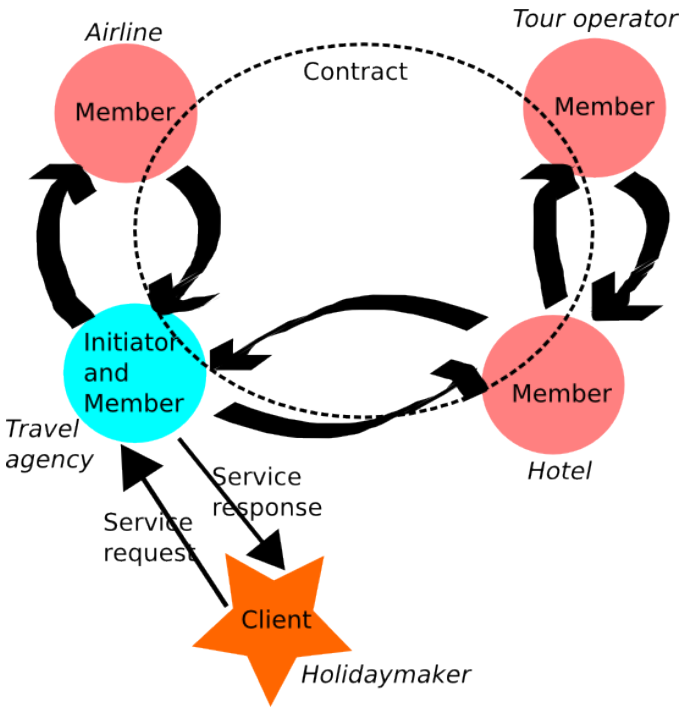

Figure 2.2: An example community with centralised coordination.

making in a community? How can the community be gracefully terminated guaranteeing that transaction data which is not needed anymore (such as credit card numbers), is deleted? However, this work focuses on the community operations phase and especially on the question of how to best secure the participant organisations from the threats arising from collaboration.

2.4 Example Communities

As discussed above, there are several types of communities. In this section two example communities are introduced, one with centralised command structure the other with a more dynamic membership style. The community examples are for illustrative purposes only, the ODP viewpoints used in this thesis are mainly computational and engineering, i.e. what processing components are needed for a community to function, how they should be distributed and what kind of interfaces and interaction there are. The enterprise view, i.e. business requirements, the purpose of the community and enterprise policies, are used only generally since the focus of this work is to create solutions applicable in various applications and business domains.

2.4.1 Centralised Travel Agent Community

A simple example community with centralised command structure is shown in Figure 2.2. The purpose of the community is to provide a service, in this case holiday bookings to holidaymakers. The community initiator is the travel agent and he/she defines the rules in the form of community contract. All participants must agree to this contract to participate in the community. The holidaymaker contacts the travel agent for a scuba diving holiday booking and the travel agency contacts his subcontractors, an airline and a hotel in the target country. There may be several airlines and

hotels that have declared their interest in joining the community, but the travel agent uses some selective criteria, such as pricing and quality, in populating this community of preferred partners.

Additionally, the travel agency does not want to deal with the tour service companies directly so the community choreography specifies that the hotel is responsible in contacting local service operators, such as companies organising sightseeing or scuba diving tours.

The community works in the following way. First, all interested airlines and hotels, i.e. peers, declare their interest in joining a community by placing the description of their offerings into the offer repository. From there the travel agent, i.e. initiator, selects a suitable set of members for the particular holiday community.

The holiday community can be built in two ways. The first type exists to provide the travel agent with a wide network of preferred partners which are later queried for more detailed offers, such as availability on certain dates. In the taxonomy of community types, this example falls in the following categories. First of all, the community of travel agent partners is potentially long lived. If the travel agent continues to receive good service from the community hotels and airlines, the collaboration is fruitful and likely to continue. Second, the community is dynamic. New airlines and hotels may join and old hotels may drop out because of renovations or hotel chain mergers. Additionally, hotels may change tour operator partners. Third, the travel agent community would not probably be exclusive. For hotels it is important to have many sales channels, so any suggested community contract limiting their participation to one travel agent's community only would probably not be received favourably. It is theoretically possible to have one very large travel agent which has a some kind of exclusive deal with one hotel, but as it is unlikely, our example community is not exclusive. Fourth, the coordination structure is centralised. The travel agent dictates the community rules to the airlines and hotels. The hotels in turn dictate their rules to the tour operators. Additionally, this type of community is explicit. Transactions involving money should always be governed by a suitable contract everybody agrees to.

It may not be technically necessary to build a separate *preferred partner* community, but real world alliances and networks have shown that this is the industry way of making business. Hotels form chains, airlines have alliances and these groups have made special deals with travel agents and car rental companies.

However, the purpose of a community is usually more detailed. The second type of community has a very specific goal: in the travel agent case to provide a particular holidaymaker a successful vacation. The key differences to the preferred partner community type above are in duration and dynamism. This particular delivery community exists for the duration of a particular holiday only and is disbanded afterwards. Secondly, once the community members have been selected and reservations booked, the members are not likely to change. However, the needs of the holidaymaker may dictate that the community must accept new and previously unknown members even if the members are usually selected from the network of preferred partners. If, for example the hotel or travel agent has not previously arranged a cave diving tour for a holidaymaker, the cave diving organiser must be accepted even if it is a new and previously unknown member. Alternatively, the travel agent may decline to organise such a tour to the holidaymaker and thus risk losing the customer. Therefore, the community should be open by its nature so that peers can freely communicate with each other according to the needs of the holidaymaker. In practise this means that in this type of communities members have to interact with previously unknown other members as dictated by the business network model.

It is quite possible that these two approaches are not separate communities at all. The community business network model and the choreography can be built to include both long lived preferred partner operations and short-term holiday delivery transactions. This illustrates the multitude of possibilities business communities can offer organisations. However, this work is mainly focused

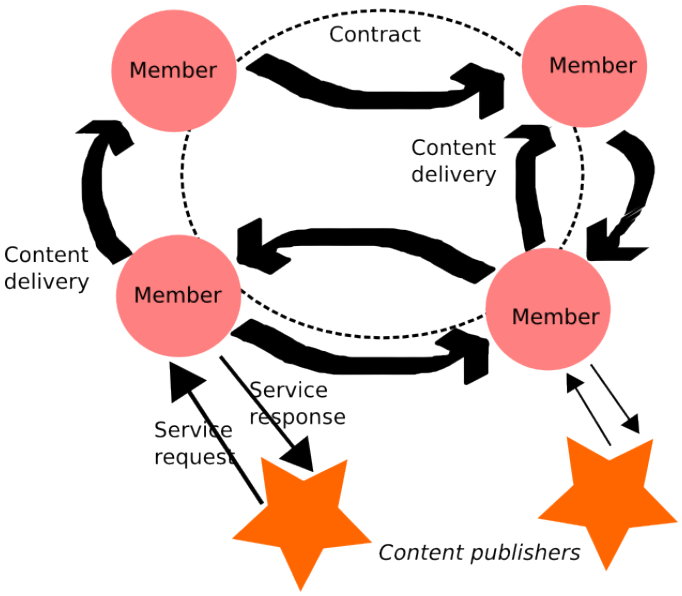

Figure 2.3: An example of a more dynamic community.

on the specific service delivery community with an open membership policy since the challenges are more interesting there.

2.4.2 Distributed Content Delivery Community

Another simple example community is shown in Figure 2.3. The purpose of this community is to provide a distributed content delivery service for digital content, for example movies. The content publisher will submit content to be available in the community and participating members will agree in the contract to host the content or pieces of it. The business benefit in this community is twofold. On other hand the publisher can very simply make the content available without huge requirements for its own network capacity. For the member the benefit is parallellised and thus faster download of the digital content. Additionally, this community could be enhanced by members specialising in selling and distributing different types of licenses to open the access control restrictions specified in the digital rights management (DRM) technology deployed in the content.

This content delivery service is more well known in the context of peer-to-peer (P2P) networking. P2P networks have the basic feature that they are based on direct communication between equal members and lack a centralised coordination server [DDCdVP$^+$02, ATS04]. This basic P2P concept can be used for different applications, such as member-to-member communication (for example Skype) and resource sharing. Resource sharing examples are content distribution with file sharing protocols such as BitTorrent [Coh08] and distributed processing such as SetiHome using BOINC distributed computing platform [And04]. However, depending on the purpose and needs of the P2P application, there may be some centralised services, such as authentication. Actually, applications are labelled *peer-to-peer* not because of their internal operation or architecture, but rather as a result of how they are perceived externally, that is, whether they give the impression of providing direct interaction between computers. Therefore the key features of P2P architectures are direct sharing of resources and the abilities to treat instability and variable connectivity as the norm and to handle transient population [ATS04].

Current Internet anonymous or pseudonymous file-sharing P2P networks can be considered

ad-hoc communities consisting of private persons and their computers. Everyone in the community can publish content and access published content in the community. Since content may be distributed across all or some members of the community, it can be accessed parallel and thus delivery is faster than from a single centralised point. While P2P is currently notorious for enabling illegal sharing of copyrighted material between individuals, P2P networking itself is a neutral technology base which has also been investigated in relation to enterprise applications. For example, some enterprises have needs to rapidly distribute large content files such as movies to a large number of clients or members and the required network capacity may exceed any single network provider's abilities. One current example is the game publisher Valve, which has a content delivery network Steam. The Steam network enables consumers to buy games directly from the publisher and play multilayer games on line. The control of this community is centralised from the content point of view, but the Steam network does incorporate some P2P techniques for direct communication and file transfer between members.

In the Steam example members are individual people, but it is foreseeable that enterprises may have content delivery needs to other businesses as well. For example, movies are made more and more made directly in digital formats. What if they are in the future distributed directly to movie theatres using content delivery networks instead of physical media that can be mislaid? Would the ability to publish the movie directly to theatres bypassing country or continent level distributors generate new revenue and to publishers?

In the standard business mindset of today, the content owner decides exactly to whom the content should be distributed. In this case there may be a populator or some other centralised coordinator deciding who can become member of the business community. However, this cumbersome pre-selection of members is not necessary if the content itself contains measures to control who can utilise it. There are several digital rights management (DRM) systems available for that purpose. So it is feasible to build a dynamic and distributed content delivery business system with no or very little member pre-selection and where there are new members joining the community during the community lifetime. Since the content should be available to new members, old members do not necessarily know in advance with whom they exchange the pieces of content.

While the current way of conducting enterprise business in the net is based on written contracts and pre-selection of community members, the view taken in this thesis is that in the future the way of making business could change to more open and ad-hoc participation if sufficient enabling technology exists. The move towards openness probably starts from specific business needs, such as content delivery, but slowly extends to other business applications as well when the business benefits become clearer.

The move towards more open community participation brings several benefits. First of all, more and more members can participate in business networks even from regions and business environments less familiar to community initiators. This means more equal opportunities for new and less known enterprises. Of course, this has the downside that existing members face stronger competition. Therefore, existing members may not view increasing openness as a benefit. However, from the community purpose point of view the increased competition may reduce the costs of service production or raise the service quality in the community, which are direct monetary benefits.

These are the key reasons open and ad-hoc community structure will be the future. However, for the open model to succeed, there are some problems that must be solved in the technical collaboration environment and support systems. These problems are security and management which will be explored later.

2.5 Technical Environment and Support Infrastructure

A technology often discussed together with electronic communities is service oriented computing. *Service-oriented computing (SOC)* is the computing paradigm that utilises services as fundamental elements for developing applications. SOC involves the service layers, functionality and roles as described by the Service-oriented Architecture (SOA) [PG03]. This paradigm enables applications across different domains to collaborate to provide a unified service.

In this paradigm, *services* are self-describing, open components that support rapid, low-cost composition of distributed applications. Service descriptions are used to advertise the service capabilities, interface, behaviour, and quality. Publication of such information about available services provides the necessary means for discovery, selection, binding, and composition of services. In particular, the service capability description states the conceptual purpose and expected results of the service (by using terms or concepts defined in an application-specific taxonomy). The service interface description publishes the service signature (its input/output/error parameters and message types). The (expected) behaviour of a service during its execution is described by its service behaviour description (for example, as a work-flow process) [PG03].

Services come in two flavors: simple and composite services. The unit of reuse with services is functionality that is in place and readily available and deployable as services that are capable of being managed to achieve the required level of service quality. Composite services involve assembling existing services that access and combine information and functions from possibly multiple service providers [Pap03]. Composite services can be offered by a composite service provider who utilises a community to provide these needed multiple services.

SOC provides the ability for interacting parties to choreograph their behaviours, so that each can apply its local policies autonomously, yet achieve effective and coherent cross-enterprise processes. In addition, it provides support for dynamic selection of partners as well as abstractions through which the state of a business transaction can be captured and flexibly manipulated; in this way, dynamic selection is exploited to yield application level fault tolerance [HS05].

While communities can be built on any paradigm, as the Internet electronic mail case shows, this idea of interacting and co-operating service modules is very useful when building communities whose reason of existence is collaboration. Therefore this work assumes the use of service oriented computing paradigm in the service level of the collaboration stack discussed in 2.1. In practise this means each community member creates a service, simple or composite, and publishes it to the community thus creating the service level from the available technology base.

However, the community may need more support infrastructure than the general service oriented paradigm to function. While enterprises offer their services to the business networks, the community population, establishment and operations phases have tasks that require additional services, such as a service offer repository, a populator, a business network model repository and monitoring components. These or similar common support functions are typically implemented in a business-to-business (B2B) middleware systems which can be called community support systems (for example ADEPT [RD00]) or virtual enterprise support environments (for example CrossFlow [GAHL00] and WISE [LASS00]).

Topologically the middleware components are distributed around the community. Each member usually has a middleware component, the network management agent, enabling community participation. Additionally, there may be several common components, such as the populator, which can be either distributed across several community members, run by a neutral third party, or fully controlled by one member. This division may reflect the community coordination structure.

The community middleware can be built as a framework, which is a reusable design of a program or a part of a program [Deu89]. For the community middleware or parts of it to function

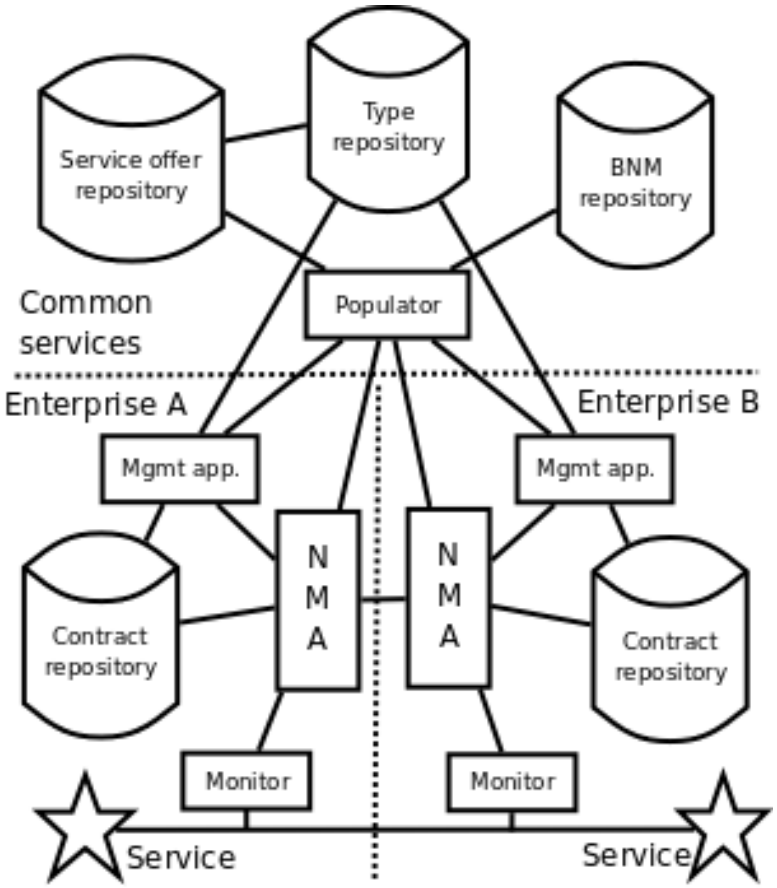

Figure 2.4: Components of the Pilarcos environment [KRRM08]

as a framework, it should by definition be applicable to many business scenarios and community structures.

Generally middleware systems are not usually very flexible from the community coordination point of view. For example the WISE system has a mandatory central coordinator component, the WISE engine, to which the clients and members connect, which distributes tasks, monitors and controls the workflow in the community. As such it has difficulties adapting to a democratic and distributed community structure such as the content delivery community example. In contrast, the CrossFlow system utilises only an external contracting service (a Service Trader in its own language) for matching service offers and requests.

This thesis uses concepts of the Pilarcos middleware system [KRM07, KRRM08], which is depicted in 2.4. In this system, the middleware consists of a populator component, service offer and type repositories and the business network model repository. Each member offering a service has its own business network management agent (NMA), a contract repository, a monitoring component and an administrative management application. This structure is applicable to many types of communities.

The threats present in the future collaborative environment are discussed next to lead the analysis towards possible solutions.

2.6 Security and Threats of the Environment

2.6.1 Security definitions

Security is generally defined as being free from danger or as the condition of safety [KT93]. The danger is presented through *threats* towards *assets*. Assets in a computing setting are typically information, hardware or software. However, they can also be immaterial like reputation or satisfaction [RK10]. A *threat* is a potential occurrence that can have an undesirable effect on the system assets or resources [Bis03].

The concepts used in the field of computing security are not very exact. *Information security* has emerged in the recent years as the umbrella concept for containing all aspects of security in a computing setting despite the name referring to only securing information assets. Concepts like *network security*, where the protected asset is the network, and *computer security*, where the asset is a computer system, are generally thought of as aspects of information security. In this thesis we follow this umbrella status and define *information security* to mean protecting both information and information systems assets from various threats.

In this thesis we define *community security* to be an aspect of general information security, where the protected asset is the community. Here the community is a combination asset consisting of all software, hardware and information assets at all community participants necessary for the community to reach its goals. Correspondingly we define *community member security*, or *member security* in short, as an information security aspect and a subset of community security, which protects the community member assets from the threats community membership brings.

The four general goals of information security are protecting the confidentiality, integrity, availability and non-repudiation of the assets. The assets can be divided into two categories, community framework assets and community specific assets. Framework assets are software and hardware components (such as the NMA) required for participation and the technical interoperability of the community. Additionally, each community has a purpose, such as holiday sales, which brings some community specific assets to be protected. The community specific assets are generally in this thesis called *business data*, since the assets travel through the eCommunity as data representation. However, it is recognised this business data can be at some point of the community functionality converted to physical objects and services.

In community security the characteristics and purpose of the service or community dictate which of the general information security goals are relevant for each asset and what is their order of importance. For example, non-repudiation is crucial for communities having sales transactions, while confidentiality is a key goal for patient records in a healthcare service community. For a generic community, contract integrity is also of importance. As a contrast, availability plays the greatest role for background image rendering system for the movie industry. Problems in rendering one frame causes delays in the rendering of subsequent frames in the pipeline and thus delays the whole movie incurring huge costs.

To analyse the threats affecting our example holiday delivery community it is worthwhile first to examine what are the assets worth protecting in this community. This community does not deal with tangible goods, the transaction targets are services. However, services have value as well and the end result of the community is that a holidaymaker will receive something of value in exchange for money given to the community. This money is eventually distributed to the community members according to the contracts made during the community lifetime. Therefore one class of assets worth protecting are the promises embedded in the community contracts. These promises may be backed up with guarantees required by the members for entering into the contract. In the travel agency world, it is very usual to require holiday-maker's credit card details as a

guarantee for a booking. Protecting the credit card information in the community is very important, since card details are very much a popular crime target for identity theft and credit card fraud purposes.

In our distributed content community the confidentiality of the the movies or other digital content, is protected otherwise with DRM technology. However, the integrity and availability of the content must be protected within the community as well. Members will not be very satisfied if they can not use the downloaded content because one of the pieces of the content has been corrupted in the community.

A threat is related to a *vulnerability*, which is a weakness that makes it possible for a threat to occur [Bis03]. An example of a concrete threat is a hacker[3] breaking into a payment system via an unsecured wireless network access point and stealing credit card information. The vulnerability in this case is weak technical security measures on the wireless access point.

Vulnerabilities that enable the hacker to break into the system can be for example bugs in the application code, architectural flaws, design or system management oversights or algorithmic problems. In the security community the method to utilise the vulnerability, for example a small program, is called an *exploit*. An *attack vector* is the specific route via which the vulnerability can be exploited, for example an e-mail message or a door to access the system console.

For a real security incident to occur and cause loss of one or some of the security goals, several things are thus needed: a threat, a vulnerability and an attack vector to reach that vulnerability with an exploit. If there is no vulnerability, a threat can not be realised. If all attack vectors for a specific vulnerability are known and closed, the vulnerability can not be exploited. However, all potential attack vectors can not always be closed. For example, very few organisations in modern business environment can be without electronic mail to the rest of the world for long and the same can be said for network access to collaborative business applications that generate revenue.

There is an additional concept related to threats, a *risk*. A risk is a qualitative or quantitative measure of how likely a particular threat is from a subjective point of view. A *risk analysis* aims to measure the risk of known threats, whereas a *threat analysis* is interested in enumerating all the possible threats only. In a risk analysis, also the *cost* of the incident caused by the realised threat is often estimated. Based on the occurrence frequency and financial impact of the threats, the most sensible countermeasures can be selected. For example, if a purchase fraud occurs about once a month with average direct and indirect losses of 10 000 euros, it is cost effective to deploy a fraud prevention solution only if it costs less than 120 000 euros yearly in total.

2.6.2 Threat Analysis Method

What is very interesting from the collaborative enterprise point of view, general threats, risks and possible countermeasures have not been thoroughly analysed in a virtual enterprise scenario [DJPJ04]. Some partial efforts in filling the gaps between single system and an enterprise level security exist, for example in the form of message context protection [vdHLP05], but no comprehensive threat, risk or requirement analysis has been done previously. Therefore, such an analysis for community security has been conducted to help focus this thesis.

Therea are all kinds of threats towards adopting an eCommunity collaborative business model. Some of the threats are the high cost while adopting and managing the ecosystem, infrastructure-level threats introduced by the new global support services, and the current lack of legislative

[3]Here the word hacker is used according to its current popular semantics, i.e. a person trying to get unauthorised access into others' networks or systems. If anyone from the MIT Tech Model Railroad Club is offended, I respectfully salute and apologise.

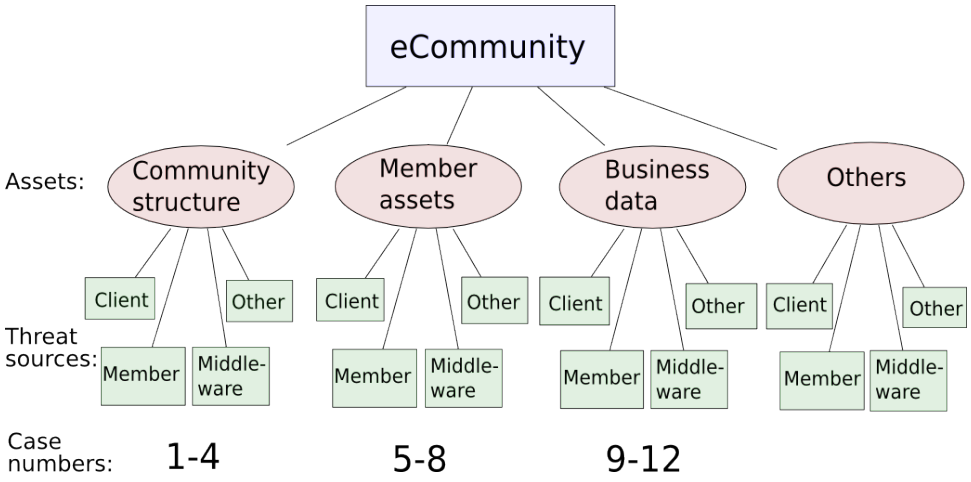

Figure 2.5: The starting point of the eCommunity threat analysis.

support [RK10]. This threat analysis is focused on the threats introduced by the required support services and infrastructure.

There are several methods to use in a threat analysis, for example checklist-based methods or more free-form brainstorming method. A checklist-based method is limited to selecting applicable threats from the pre-compiled list and may miss threats not present in the list. Free-form brainstorming on the other hand is not limited in this way, but still may miss important threats just because some aspect of the target has not been forgotten from the analysis.

Therefore the threat analysis here has been done using a more formal threat tree method [Amo94]. The method divides the target system into parts or views which can then be further divided as many times as necessary to create a tree structure comprising of different cases for consideration. The method produces a conceptually complete target system decomposition if each division includes a part or view called *other*, which includes everything else not listed in the explicitly named cases.

While the threat three method is essentially a form of guided brainstorming this completeness ensures that every possible angle of the system will be considered during the analysis. Of course the quality and depth of the results is dependent on several factors: the actual divisions, the capabilities and knowledge of the person(s) doing the analysis and the time available for the analysis.

In this thesis, the target system is an eCommunity and the division into different views has been done twice as shown in Figure 2.5. The first division is done according to the assets an eCommunity must protect. These assets include first of all the eCommunity framework itself, its functionality, structure and operational data, for example offers, contracts and the business network model. Second, each member itself has assets requiring protection, such as computing resources and private data belonging to that member only. Third, there is the real purpose of the community, the data transferred in the community for business purposes, for example credit card numbers, pricing and quantity information of goods, or personally identifiable information. The rest of the assets are in the "others" section. Assets in this final category can be non-tangible such as the member or community reputation.

The second division is made according to the community actors, i.e. sources of threats. Four types of actors have been identified here, namely clients, members of the community, community middleware components and of course, the others. An example of the other actors are external

hackers. When the community purpose and the community support components are fully known, the actor list can be expanded to specific member roles and named middleware components. This expansion would make the threat analysis more meaningful and complete for software design purposes. However, at this point a general threat analysis suitable for many different community purposes and potential frameworks is more useful.

Thus from these two orthogonal divisions the threat tree yields sixteen cases to be considered. The key question asked in analysing all these cases is: *What threats can the actor X cause to the asset Y in the eCommunity?* The divisions here reflect the issues in the focus of this thesis. To focus this thesis the "Other" asset subtree is ignored and the analysis concentrated on the three primary asset types leaving 12 cases. In other research projects the divisions can and should be made differently and the analysis focused differently. More detailed divisions can also be performed if necessary, but it very easily yields a large number of cases to be considered. This exponential case number explosion is the drawback of the threat tree method.

The threats found are usually somewhat similar between similar cases (for example "client towards the community" vs "client towards member assets") but there are subtle differences warranting separate consideration. For each found threat it is classified into one of six broad types according to the the STRIDE method [HL03]:

1. *S*poofing or masquerading as another entity. Spoofing can occur if the authentication mechanisms are weak or non-existent between the communicating parties (client, web server, application servers etc) especially if data crosses organisational trust boundaries. While spoofing is a legitimate threat itself, it is usually used to cover other attacks, such as denial of service.

2. *T*ampering of data. An attacker can potentially manipulate all data residing on the system or travelling in the data stream between client and server. Sensitive data can be manipulated also in a rich client application. Client applications can be reverse engineered and data embedded in them changed.

3. *R*epudiation, i.e. denying that a transaction occurred or was initiated by someone.

4. *I*nformation disclosure. Data can be obtained from the client machine or when the data travels in the network. However, weak server or application security can also lead to information disclosure from the server or application internal data structures.

5. *D*enial of Service (DoS) prevents a system from performing its duties. Messages with malicious payload can cause server crashes or hangups. It can also be caused by flooding a communicating party with excess data or service requests. DoS are typically server side issues, but sometimes a DoS attack can be directed at a client as well, especially in a rich client situation, where significant response processing happens at client as well.

6. *E*levation of privilege can be thought from two points of view. First, to gain privilege where no privilege was before (i.e. unauthorised access). Second, to gain more privileges in a system where some legitimate access already exists.

2.6.3 Threat Analysis

Since the threat gathering and analysis process is by its nature open-ended the list of threats below is incomplete. Each practical community application should perform its own threat analysis which takes into account the particular business requirements and the technical details of the application

and framework. For example, the threat tree of the travel agent example could be constructed differently and thus some different and more concrete threats in that particular environment found. However, this list may act as a starting point for such a more detailed analysis.

Please note that at this point no judgement is passed on the viability and likelihood of the observed threats, these are discussed later. Similarly, existing solutions to mitigate the found threats are discussed later.

1 - A client towards community

A client requests a service from the community so it has access to at least one community interface.

Spoofing or masquerading as someone else is a possibility in any interaction. In this case the threat is that the client masquerades as someone else and thus creates problems for the community structure. The client can try to masquerade as another client, a community member or a middleware component such as the populator.

Masquerading as another client may cause problems for the community if the business process is not a simple request–response but requires also interaction in between. Out-of-order or semantically unexpected requests and/or responses may cause the community choreography driven by the business network model problems. Moreover, it is possible that if a client can masquerade as the populator it can falsely initiate contract renegotiations, repopulations or even community termination.

A client in his/hers normal role does not really have much tampering possibilities towards the community structure and/or functionality. The client can only affect the data and requests he/she sends to the community. However, a client by definition has access to the community via a member or initiator and thus can try to hack into the member via a community interface which has a security vulnerability and thus gain more tampering opportunities.

A client can also deny sending a service request to the community, i.e. repudiate the transaction. This may cause problems to the community especially in cases where the community charges a fee from its services. Alternatively, the client may claim never having received a particular service or data requested.

A client can also disclose privileged community information if the community reveals any private information to the client. For example, this information may include identification of the community members if the service response includes this information. This structural information may also be available if the client is offered some kind of request tracking interface to see how the request is handled in the community.

The client has also some denial of service capabilities towards the community as a whole. It may flood the community with service requests, valid or invalid, so that other clients can not be served. Alternatively the client can block responses from getting through and thus reserve and eventually exhaust resources in the community.

A standard client can usually only send service requests and receive replies. A client may try to elevate its privileges in the community by hacking into community members or community middleware components, which have more capabilities (or privileges) in the community. By definition the client has access to the community member/initiator it contacts for service, and possibly to others, therefore this is a valid threat.

2 - A member towards community

Threat of a member spoofing or masquerading as another member is valid. Successful spoofing can cause the community choreography problems when interaction patterns break down or become semantically unexpected. However, the threat of a member masquerading as a middleware

component or other community infrastructure actor is more problematic as it can cause disruptions and security breaches in the community functions. For example, masquerading as a populator is especially problematic in communities and situations where periodic repopulation occurs. In these cases the attacker can basically reorganise the community as he/she wishes.

A community member can also tamper with the community metadata or interaction messages that it has access to. This may include contract data and negotiation protocol messages, but one member can only have a limited effect in the contracting and negotiation process.

More interesting threats are possible with a *Sybil attack*, where a single malicious entity can create several false identities and use those false identities to collaboratively harm the system [Dou02]. The validity of the attack here is very much dependent on the identity establishment system the community uses. One malicious member could try to establish several different peer identities and influcence the membership selection and community setup. In an open community where anyone can join or send offers, this is a valid threat, although the population process ultimately selects the members. A separate threat with similar effect is the situation, where several different members act in collusion to shape the community towards their desired goals.

A community member can also deny ever agreeing to a specific contract or community participation.

Members usually have access to some structural and contractual information of the community. The member has a contract with the community and the contracting information may include information about other members of the community as well. The member has some choreography knowledge, for example which parties act in which roles. The knowledge is not necessary exhaustive, but at least it usually knows its up- and downstream partners in the service choreography sequence. This information may be revealed accidentally or deliberately.

If a client can cause havoc with the community interface to the point of denial of service, the community member has even more disruption possibilities. While clients can only access public interfaces, members may utilise other members' private interfaces and some middleware components. By sending massive amounts of valid or invalid messages they can shut them down (if a vulnerability allowing it exists) or exhaust the receivers' resources and thus prevent them from serving others.

A member may try to gain more privileges in the community by hacking into other systems via vulnerable interfaces or systems. Other members do not usually have more privileges than the attacker towards the community, except in cases where the community control is centralised. In that case a member can gain control privileges in the community by attacking the central member.

3 - Middleware towards community

Middleware and community framework components are the most security critical components when the community structure and functionality are in focus. The middleware components taking part in community population and eContracting processes could spoof their identity and masquerade as another component, member, prospective member or client. Components could also tamper with the community setup messages, such as contract offers and population messages.

At minimum such spoofing and tampering means disruption of the normal community communication choreography and will cause at least a temporary slowdown and possibly a denial of service situation while the community tries to gracefully recover from the disruption.

Spoofing can also have more severe consequences, for example in situations where the community middleware components have been distributed to different organisations. For example, if one organisation provides the offer repository and another the populator, the populator could be directed to use a third party offer repository if the offers at the old one are not to the liking of the

organisation running the populator. This in turn may cause the community to provide a different type or degraded quality of service to the client. Changes like this are not visible to the client or maybe even not to the initiator.

There is also the possibility that the contract and type repositories change the offer and type data stored in their databases after it has been entered there. This could cause the community to be set up under false pretences if the members do not check the final contract diligently.

Some repudiation threats are also possible. The offer/contract repository can repudiate ever receiving a particular offer or contract. This may have negative consequences for the community in cases where the rules of population dictate that for example the cheapest offer must always be selected, as is common in public sector tendering. The same end result can also be achieved if the offer/contract is simply lost in transit or dropped due to a software glitch in any middleware component.

The middleware components have naturally access to many community messages and metainformation, especially contracts and the business network model information. Middleware components may reveal some of this information to third parties as part of error messages. It is also possible if some middleware components or component instances are shared between several similar communities, information about one community may end up visible to another community members or administrators. This is an especially relevant threat with shared data storage middleware components such as the offer and business network model repositories even if the communication with these components goes via a centralised populator.

Since the middleware is responsible for technically implementing the community initiation, repopulation and termination tasks, components may very easily cause a denial of service situation either by ceasing to function due to a software bug, deliberately refusing to function or respond at all, or attacking other middleware components with excess or malformed traffic.

In a reverse case there is a possibility of denial of service if the middleware components fails to send the correct community termination messages and the members must keep the community interfaces, components and data structures alive in their systems. This denies the members the ability to use these reserved resources for something else, so this could also be classified as a threat towards the members' assets.

4 - Others towards community

If the community is not open, but requires clients to authenticate themselves, other external parties may try to spoof their identity to gain client status within the community. This is especially desirable from attackers' point of view if the service provided is of material value and the attacker can spoof its identity so that utilising the community does not incur any costs to the attacker. From the community point of view the threat is of using resources to provide a service without benefit or proper compensation, basically a fraud.

External parties may try to conduct a man-in-the middle attack to gain access to the community internal messages. If the attack is successful and gains both read and write access to data, this threat belongs to the tampering category. If the attack is only partially successful and gains the the attacker read-only access to data, it is a case of information disclosure. Tampering is the more dangerous of the two from the community functionality point of view, but information disclosure may also reveal data community members wish to keep community confidential, such as contract terms and safeguards. Similar effects can be achieved if the attacker succeeds in masquerading as a member or a middleware component.

Denial of service is a possible attack as well. Especially distributed denial of service (DDoS) attacks conducted via botnets are a problem in the current Internet and it does not seem to be a

diminishing problem. Problems are greatest in communities offering services to everyone, not just a pre-selected list of approved clients. The DoS attack can happen on the network level (IP/UDP packet or TCP connection flood) or on the application and community message level. The higher level in the protocol stack the DoS attack is using, the more resources it takes at the receiver end to analyse and classify the incoming contact as a DoS attack.

A denial of service situation can also be caused by a non-malicious agent or the threats can be in fact agentless. For example, networking equipment may fail due to human configuration error, because of a software glitch or just because the hardware reached the end of its life. These issues are possible, but from the point of view of this thesis not very interesting and thus these types of threats are not elaborated further.

5 - A client towards a member

Since a client must have network level access to at least one community member (the initiator or other public interface member) the client can try to exploit a vulnerability in that interface to gain unauthorised access to the member network node. Depending on the type of the vulnerability the effect of this attack can vary. It can give the client access to member's private data (for example passwords) or it can enable the client to write to member's system thus allowing tampering with the member's data. Additionally, the vulnerability can give the attacker means to conduct a denial of service attack towards the member. A vulnerability can also provide the client a way to gain elevated privileges in the member system and thus gain complete control of the system. This in turn has severe consequences since this maliciously controlled system can then be used as a stepping stone for more advanced attacks towards the member's internal systems or towards the community.

An example vulnerability in the member system allowing these type of threats is a software problem at the interface, where it fails to properly sanitise or filter client submitted data so that the data is interpreted and executed as code in the member system.

It is interesting to note that the attacking client does not necessarily need direct network access to the victim member. The vulnerability may not be at the first public interface but may actually reside at another member further in the business process interaction chain. The vulnerability could theoretically be exploited if the maliciously crafted input data is passed unfiltered down the business process chain.

6 - A member towards another member

If a member successfully spoofs its identity and can masquerade as another member, it can also receive information private to the spoofed member. This may have severe consequences if the private data is for example part of the contracting process and price negotiations.

To tamper with another member's private data, the attacking member must have access to it. If such private data is sent to the community during for example contracting process it is possible another member listens to it using a man-in-the middle attack or masquerading as a community middleware element. Otherwise access can be gained by breaking into the victim member's systems. The attacker and victim members may have open access to interfaces between themselves if the community choreography so dictates. The attack can be conducted for example with maliciously crafted messages towards a vulnerable interface.

Direct access between the attacking and victim members is not always necessary. A member could also try to hack its way to other member's system by sending maliciously crafted data victims via the middleware mediated offer and contracting messaging exchange. Even if there is no

direct communication between the attacker and victim systems, the populator and offer repository may pass the data unfiltered from the attacker to other members of the community. If these members then have vulnerable systems interpreting the received data as executable code, the attacking member may gain privileges at the victim system or create a denial-of-service situation.

Denying a transaction has taken a place between members is a valid concern. Repudiation issues can affect members directly, for example when another member ever denies sending a service request or receiving a response. While it does mean disruption in the community operations as discussed above, it may have grave consequences to the member who acted based on the now repudiated information. Not only does it mean costs in the form of lost processing power in the victim's systems, but it also may mean real financial losses in the form of shipment of physical or intangible goods.

Even in normal legitimate community transaction between members there may be some private information exchanged. For example the members may have details of each others' network structure, addresses and security policies. This information may be leaked accidentally for example in the form of error messages to clients or community members. Additionally, it can be maliciously revealed if a member is so inclined.

A member can also carry out a denial of service attack towards another member, especially in a case where members have a relationship dictated in the community agreement. The attacking member can thus legitimately send requests or responses to the victim. The messages may be multiplied or otherwise flooded to the victim and thus preventing the victim member systems from carrying out their normal business.

7 - Middleware towards a member

The problems caused by attackers exploiting software vulnerabilities discussed above with client–member threats are of course possible when the attacker is a middleware component. However, here the more interesting additional threats are discussed.

Some middleware components, such as the populator and the business network model storage, are in the privileged position that they have knowledge of the community as a whole and they practically control the community structure and metadata. This knowledge may aid them in masquerading as a community member or client. Spoofing may reveal data the victim member does not wish to distribute to the community middleware.

Data tampering is possible in a scenario where the member sends out some private data with the contracting offer and a middleware component having access to the data changes it. A middleware component may also try to attack the member via a software vulnerability in an accessible interface and gain access and data tampering possibilities that way. There will be accessible interfaces between the member and the populator and offer repository components, so this is quite possible.

The only repudiation type threat towards a member's assets found is the other side of a middleware component denying ever receiving an offer/contract. Above, this threat was discussed from the viewpoint of the community losing by not getting to know all possible offers. However, seen from the perspective of the member whose offer was lost, it means potential loss of revenue.

Disclosing private member data by a middleware component is possible in scenarios where private data is sent to the community middleware. In this case any software bug or design flaw in any middleware component could cause the private data to be displayed to outsiders for example as part of error messages or log files. Especially authentication tokens, such as passwords, are at risk of disclosure if the member–middleware interaction is using password based or other weak authentication schemes.

There is also a threat where the middleware component floods the member with enough requests and/or responses so that either the member node resources or the member communication capacity is exhausted. Especially in the case of communication capacity exhaustion the DoS can affect severely also the normal networked business of the victim member.

8 - Others towards a member

The threats of external parties exploiting vulnerabilities of publicly accessible interfaces at the member's systems apply also here. This is case of the prototypical external hacker attacking a public networked service. If such an attack is successful, the attacker can tamper with the data or disclose it to third parties or even destroy it causing a denial of service situation.

Also previously, the threat of an outside party spoofing its identity and initiating a service request was discussed in the context of the community as a whole losing resources without compensation. However, if this same spoofing problem is looked at from a viewpoint of a member, the threat is about an attacker committing fraud so that the victim has to bear the cost of attacker's actions. Of course the issue of who should bear the costs of a fraudulent transaction is dependent on the legal framework governing the community participation.

9 - A client towards business data

Generally a client is a major source of the business data. The client provides a service request with enough details that the business process can be completed. However, the data originating from the client is not necessarily the only business data in the community. Members can interact with each other using a different set of business data if the process so dictates. For example in the travel agent community the client provides his/her itinerary wishes and a credit card number. But the hotel booking a sightseeing tour for the tourists uses a somewhat different set of data when interacting with the tour operator, which may contain for example bulk rate discounting codes and other assets requiring protection. These both cases need to be considered in a threat analysis.

A client has limited possibilities in affecting the business data that travels the network. First of all, it can always spoof the original service request either coming from someone else or just falsify the business data (for example credit card number, order numbers etc) coming from it.

Generally the client does not have access to the business data moving between the community members apart from the information it gives on the original service request. However, if the business network model and the choreography built based on it requires the members interact with the client during the process, the client may have some more opportunities to tamper with the received data. For example, if in our travel agent case the client is asked during the process which combination of airline, hotel and tour he wishes, it is possible that in the confirmation response the client could change some reference and account numbers.

A client can also deny sending a service request with particular contents, i.e. claiming it sent different data. This is a subtly different case from denying sending the request at all, since here the client can claim the received service was not correct according to the request. Somewhat related is the threat where the client just denies the received result is correct even if the response matches the request.

Clients leaking business data may be a possible threat in the scenarios where the community members interact with the client also during the process. In those middle-of-the-process message exchanges there may be business data that the client can leak to the outsiders. For example, if in our travel agent itinerary confirmation case the confirmation request could contain business data such as discount rates, bulk pricing, account numbers etc.

Here the case of the client accidentally or deliberately leaking its own business data, such as credit card numbers, is considered trivial and not elaborated further.

Normally if the client tampers with the business data, it can only change the quality and nature of the service it receives. However, in some cases tampering with the business data may cause significant activity and reorganisation in the community, which may cause denial of service type of situation. For example, if the original purchase request was for 5 units of ball bearings or movie frame renderings but it is later changed by the client (or by someone else) to 5 000 000, this may exhaust the resources of either the community or individual members.

10 - A member towards business data

In the previous section some possibilities of a client causing threats to the business data were discussed. However, the community members are the biggest source of threats. This is simply because they as a group have the most opportunity since the business data travels through them.

A member could affect the business conducted in the community by spoofing the business data or request origin. Spoofing the request or data origin at the middleware could be used for example delivering services to an organisation with whom the service provider does not want to do business.

The member can change the business data when it travels through it. The data presentation format can change several times during the community process interaction if the members' interfaces are different. This transformation is not necessarily a security threat, but if these transformations are not done carefully the semantics of the data can also change, which then could be seen as a real data tampering threat. Also in our content delivery community a member could change the content pieces it stores and delivers to other members.

In a reverse threat a member does not perform a necessary transformation even if the other member's interface would require it. An example of this type is a failure to do a currency or unit conversion.

A member can also store the business data travelling through it. In some regards it is necessary, since the business network interaction failures need to be analysed and in the absence of any logs it may be impossible. However, if the members retain the business data in the logs, the data storage becomes a desirable attack target for external parties. It is also possible that the business data owner has some business rules concerning the storage of said data (for example credit card numbers or other personally identifiable information) and the member's storage and logging policy may violate those rules.

Information leaks of the business data are also possible if the member systems reveal that information as part of for example error messages. Deliberate information leaks can occur too, if the member decides to turn malicious.

A member can duplicate or multiply received service requests and thus cause problems to both the client and to other members. Business data can also get lost at the member so that the service request is never fulfilled. These could both be classified as denial of service threats.

11 - Middleware towards business data

The middleware components should not theoretically have access to the business data. The middleware components ensure the community is populated according to the requirements, the business process interaction proceeds correctly and the renegotiations and community termination happen nicely. None of these functions should require access to the actual business data, just the community metadata.

However, in practise the situation may be more challenging. If there is a middleware component responsible for the community interaction monitoring or transaction logging, the members may send also business data, such as order numbers, as part of the interaction information. These monitoring and logging components can also be in a man-in-the-middle type position regarding the community interaction and thus gain access to the business data. If this is the case, the middleware can cause similar threats to the business data as discussed above as threats caused by a member.

12 - Others towards business data

As clients can try to exploit vulnerabilities in the community components to gain access to the member data, so can other external parties try to gain access to the business data. Usually the business data is of primary interest, especially if it includes any information that can be directly utilised in criminal intent, such as credit card numbers. In this case any vulnerability in any component through which some of the business data flows or to which it is stored presents a real threat.

One class of threats not discussed previously may arise if one or several members participate in many similar communities and there is infrastructure sharing between the communities at the member. In this case there may be accidental business information leaks between the communities. Other communities' members or middleware components also may try to actively exploit common members' vulnerabilities in order to get access to the other community business data, such as pricing information. Deliberate denial of service attacks are also possible if the communities can be seen in any way competing with each other. Occasional denial of service can also happen if both communities are in a phase requiring a lot of the common member's resources and there is a possibility of resource exhaustion.

2.6.4 From Threats to Risks

A threat analysis gives general guidance to the type and nature of threats needing countermeasures. The next task in assessing the risk would be analysing the impact and frequency of the listed threats in the community. The environment may already have some countermeasures built-in, which could be then taken into account in the analysis.

The analysis would reveal which of the risks are most likely and which have the most impact if the risk is realised in the current environment. The full analysis would help in prioritising and focusing the risk mitigation efforts.

However, from the viewpoint of a thesis interested in drafting a community security framework, taking into account the full prioritised order of risks is not possible. To focus this work to match available resources it is sensible to pick only a few of the listed threats and analyse just their potential impact and frequency. If that analysis reveals these threats require solutions in the framework, solution candidates can be developed and their suitability further analysed. The remaining threats can be explored in other research projects.

The focus of this thesis is on solutions to the threats stemming from peers or members acting maliciously by spoofing their identity and/or tampering with the data. Threats involving malicious peers have been introduced above in sections 1, 5 and 9 while discussions on malicious community members have been discussed in sections 2, 6 and 10. Malicious outsiders have been discussed in sections 4, 8 and 12 above. The fact that threats from malicious peers and members have been found in many categories suggests that this is an inherent issue in enterprise communities and thus requires solutions before the community business models will be more widely adopted.

In estimating likelihood or frequency of malicious peers we assume an open community, which is the worst case from the trust and security perspectives. Assuming open Internet facing interfaces, it is a fact supported by current network security experience that that there will be malicious parties connecting to them, whether the parties are clients or members trying to spoof as someone else or outsider hackers.

If the malicious parties are clients or members, they already have all the tools they need to interface with the community so there is no technological hurdle. In contrast, outsider hackers will need to make an effort to develop tools to semantically connect to the particular community instead of just connecting at the network level. This makes the attack likelihood from outside attackers smaller. However, taking into account the trend of attack sophistication rising from the network level towards the application level in the protocol stack [Eur07], it is very probable that all open community interfaces will eventually be exploit targets by malicious outsiders. Therefore the likelihood of this threat will increase over time.

It is also possible that an old and respected member of the community will suddenly turn malicious either consciously or unconsciously. The unconscious malice is the case of an outsider hacking his or hers way into the member's systems and turning the member's infrastructure against the community. This kind of behaviour has been seen in Internet *botnets*, where simple network-connected personal computers have been turned into remote controlled attack platforms that can be instructed to for example send spam or take part in distributed denial-of-service attacks. It is foreseeable that if a community is interesting enough, this type of attack could be directed towards communities as well although the development cost for the attacker would be quite high and thus the likelihood low. However, the impact of such a successful attack is major, all the business data flowing through the member would be open to tampering and eavesdropping.

There is also the case of conscious malice, i.e. the member changing its way of making business into a malicious or disruptive activity. The likelihood of this depends among other things on the likelihood of detection. If a member can silently and undetected act against the community purpose, and it can get more benefit from the malicious activity than from within the community contract constraints, there may be incentive for disruptive or malicious behaviour. However, this incentive must be weighed against the possible repercussions against the member if and when the malicious activity is discovered.

The likelihood of malice is also dependent on the purpose of community. An outsider may have very little monetary incentive in attacking a procurement community for ball bearings, but if there is a chance of gaining free airline tickets or hotel stays as in our travel agent scenario, or especially credit card information, the community will be an attack target. The likelihood of exploitation is also dependent on other security measures, such as how easy it is to convert the community participation into money or other assets. If for example the travel agent community is built so that the participant members are given credit card details or hotel vouchers on-line, the interest of malicious parties is bigger than in a system where the transaction produces only reservation confirmation number that are bound to a specific passport serial number. Similarly, stealing movies from the content delivery community is not profitable unless the attacker has the technology or licenses to open the DRM protected files.

In our travel agency community example the key business data assets to be protected are credit card numbers of the holidaymakers used to confirm a reservation. These numbers are a prime targets for criminal elements and thus attacks are very likely.

What is the impact then of various malicious threats? Unfortunately it is very difficult to predict the impact in a generic community. The impact depends on the purpose of the community and also on the community structure and the role of the malicious party. In centrally controlled communities, such as our example where the travel agent is very much in control, the impact to

a single hotel participant can be very limited. However, if the malicious party is in the role of a centralised payment processor or license generator, the impact is significant. There are issues in more democratic communities as well. In the content delivery community, one malicious member can cause significant data corruption problems affecting many other members.

Especially communities where any goods of value are traded have possibilities of financial loss, which are at least in the order of a single typical transaction value and may be on the order of sum of all transactions in a day or week or a month, depending on if and how fast the malicious participant is discovered. However, the financial loss may be bigger if the malicious entity is in a financially central role, such as the payment processor.

From this risk analysis can be concluded that the likelihood of malicious insiders and outsiders is somewhere between high and low depending on the purpose of the community. Additionally, it can be concluded that the impact of malicious actors can vary from minor to major. Therefore, the conclusion for the general community security framework requirements is that the risk of malicious parties in open communities must be managed. otherwise the applicability of said framework is severely limited to low-risk closed communities. It is also possible that member organisations refuse to use the community framework if the community member security features do not offer them sufficient protection from malicious parties.

2.6.5 Countermeasures and Solutions

There are several possible measures to counter the threats found above. These measures need to be built into the system as early as possible [And01]. The countermeasures, *controls*, can be divided into two main types: *preventative* and *detective* [KT93]. Preventative controls, such as firewalls, prevent some threats from occurring. Detective controls, such as fire alarms, do not prevent the threats from happening, but detect a security problem and alert when it occurs.

There are are also three supplementing types: corrective, deterrent and recovery [KT93]. Corrective controls, such as security guards locking all doors for the night, make sure errors are corrected. Deterrent controls, such as zero tolerance towards security policy violations are used to discourage people from attacking the system or organization. Recovery controls, such as data restore processes, help the organisation to recover from a security incident.

Quite often solutions from many of these types are necessary when risks are mitigated. What possibilities do we then have to restrict either the impact or likelihood of malicious parties attacking the community or one single member of the community?

From the community topology point of view one solution to counter the threats springs instantly to focus. If we can restrict the communication so that only community members can communicate with each other and the middleware components, and each community member can only affect the members with whom it directly communicates according to the BNM choreography, the attack vectors are few and can be individually managed.

For example, the WISE community support system uses a centralised WISE engine, with whom each member and client communicates [LASS00]. It is not explicitly said but can be inferred from the description that each participant must be pre-authorised to connect to this engine. The previously mentioned CrossFlow system relies on proxy-gateways at each participant to control remote method invocation traffic to and from the organisation [GAHL00]. The security principles and exact function of the said gateway are not explained in more detail.

Similar communications restriction functionality outside the community context exists in the form of generic network level firewalls and application level firewalls. Network firewalls usually limit access based on source and target IP addresses and application port numbers. Application level firewalls, for example web application firewalls (WAF), can analyse the application protocol,

such as HTTP in WAF case, in more detail and filter access based on the protocol messages and message contents.

Analysing the situation further from the point of a single member, for example a hotel in our travel agent example, it is clear that the individual member must be able to participate in different communities organised by multiple travel agents. It is quite possible that the travel agents are not previously known by the hotel, agents may have picked up the hotel's offer from the open offer repository. Additionally, the hotel must be able to connect with multiple tour operators, some previously unknown, based on the needs of the holidaymaker. While the threats from the point of view of the community as a whole may be easily manageable by restricting network level communication, the needs of the individual members make this solution impractical. Similarly, the content delivery community is built on the principle that new members can join any time and download the content by contacting other members.

From the community participant point of view it is difficult to manage the risk by preventing incidents stemming from community participation. There are several preventative options: to close the attack vector interface, to limit the access to authorised peers, and to ensure the exposed interfaces are free of vulnerabilities that can be exploited.

First of all, the enterprise can not close the attack vector at network or application level since it would deny access to the potentially useful community and business. Secondly, static and manually managed access and authorisation rules are not cost-effective protection in the dynamic communities of the future, the management overhead in any way dynamic communities is too large.

Besides, allowing access to authorised peers only does not prevent incidents. One single community member can not prevent other participants from turning malicious, so a threat always exists even with authorised peers. What an enterprise can do in principle, is to ensure that all software interfaces and components it is using or exposing are fully free of vulnerabilities. In practise this is impossible for two reasons. First of all, the enterprise may not have any control over the software quality if the application has been bought as a software package from a third party vendor. Second, if it had developed the applications and agents in house, high level of software quality assurance including source code auditing is time consuming and expensive.

Application level firewalls could be used to delete undesired content from the application protocol, for example HTTP, and thus to reduce threats. However, community participation adds functionality layers above the basic application protocol, and thus application firewalls do not prevent all attacks stemming from the community participation.

Thus, current preventative controls can not be used to manage the member security threats in cost-effective manner. What about other types then? Corrective and recovery controls are used by definition when an attack has already been successful. In a fast paced electronic community the attack may already have done significant damage propagating to other community members before a correction and/or recovery action is instantiated, especially if there are no detective controls and manual action is necessary. So these types of controls are not enough as themselves but useful combined with other controls. Additionally, deterrent controls are not effective if the attacker is an automated non-aware component, for example a wrongly configured NMA. In the case of our key threat, a deliberately malicious party, deterrent controls can be rendered ineffective if the attacker is able to make the attack look like it originated elsewhere so that the retaliation is directed at a wrong party.

Since the availability of suitable and good community security solutions is a requirement for wider adoption of electronic communities, something else must be found. The conclusion is to focus on researching a community security solution which uses detective controls, possibly augmented with corrective or recovery controls.

One well known detective control is system monitoring. System monitoring can be part of

the ODP *security audit function* or a general monitoring function. The security audit function is responsible for monitoring and collection of security related actions [ISO96b]. The types of security problems monitoring can detect depends on the monitoring system and data the monitors have access to. This control mechanism is discussed next.

2.7 Monitoring in Communities

As with any interaction, a community should be continuously observed or monitored to ensure successful collaboration and to detect any errors or security incidents during the processes. Monitoring can be used to achieve many different goals and the selected goal also dictates what kind of monitoring mechanisms must be used.

The basic need for monitoring in any distributed application rises from the general functionality point of view; someone must be capable of observing the status of the distributed system to see where the computation or transaction currently resides especially to detect whether any technical problems, such as deadlocks, have occurred. For example, a business process has a certain choreography, where each peer in a role must obey a certain sequence of messages for the process to succeed.

Another set of monitoring goals are related to community security. When threats or risks are not completely known, designed countermeasures may not have sufficient coverage to protect from all types of problems. In the face of these unknown or very little understood threats, which can not be countered in the framework design, the task of security monitoring is even more important.

Given the challenging threat environment of open networking discussed above and the impossibility of closing all potential attack vectors, two key goals of security monitoring in a community are first to protect one peer from other peers, and second to protect the community as a whole from misbehaving peers. These goals fall into security realms of community member security and community security, respectively.

This division of goals is not the only variable in monitoring. There are various functional parts in the enterprise collaborative system and each part can have its own monitoring mechanism to ensure that the part is functioning well or that it is secure. Thus there are several different monitoring tasks in the collaborative community [Haa05]:

1. Community independent monitoring, which is a general monitoring function in an organisation. Monitoring tasks on this level are not knowledgeable of the community functionality but are focused on for example TCP/IP networking issues.

2. Application monitoring, which is monitoring the inner state of a single application taking part in the community.

3. Choreography monitoring, which follows the state of connections and determines whether messages sent or received follow the choreography.

4. Task level monitoring is responsible for following the task to be performed and deciding whether to change to the next phase or epoch.

5. Role level monitoring. Peers can have several different roles during the business process lifetime and their status and changes can be monitored.

6. Session monitoring. There can be several sessions within one contract or agreement and their state and completion need to be monitored.

Level	*Example monitoring targets*
Business level	Fulfillment of business objectives, business risks
Organisation level	Policy breaches across all communities
Community level	Roles, choreography, tasks, sessions and contract breaches
Service level	Service and network semantic behaviour
Technology level	Community independent network errors, application availability

Figure 2.6: Monitoring tasks in collaborative communities

7. Contract or agreement monitoring is responsible for observing the agreements and their fulfilment in a business network.

Contrasting this list with the five-level general community layer structure discussed earlier in 2.1, several observations can be made. First, the application monitoring mechanism may work on the technology level or the service level depending on the mechanism's access to the service as a collection of software modules and its understanding of the semantics of the service. Therefore, service monitoring must be added to the list above as a new task.

Second, choreography, task, role and session monitoring are all functions of the community level. Agreements or contracts should be also monitored there. However, the community level monitoring function does not have knowledge or organisational policies, so contract breaches during the lifetime of the community should be raised to the organisation level so that the breaches' severity can be investigated and it can be decided whether the breach is against the organisation policies. Therefore, there needs to be a monitored feedback mechanism also at the organisation level.

Thus, following the structure of Figure 2.1 earlier in Chapter 2.1, the enhanced stack of monitoring tasks in collaborative communities is displayed with example monitoring targets in Figure 2.6.

All of these monitoring tasks can be performed by the participant organisation or peer itself for its internal needs (*introspective monitoring* in the realm of community member security), others require a party that monitors elements for the good of the whole community (*inter-enterprise mon-*

itoring in the realm of community security). Even if one party performs both, these tasks should be conceptually separate since the goals, constraints and methods of monitoring are different.

In introspective monitoring, each peer monitors its internal state against its own policies and the community choreography and contracts. On the technical level the monitoring target may be syntax errors in various network protocols and on community level contract violations during the participation in the community. For example, the community contract is usually specified in a high-level language, which specifies community actions and their order and modality, plus value threshold limitations. Contracts do not necessarily specify order syntax and semantics in great detail. Therefore, a peer may fulfil a contractual obligation by sending a purchase order, but its payload may be such that it syntactically or semantically causes security or policy violations on the receiving end. And even if the choreography defined exactly what should be sent and when, a malicious peer can violate the choreography specification. Therefore introspective monitoring is necessary. It should be available at all levels, but especially in technology and business levels.

Inter-enterprise monitoring is focused to ensuring that participants in every role fulfil their obligations on every level (from technical to business level). For example, agreement monitoring can be considered part of the contract enforcement function, which observes the participants' conformance to the contract so that any deviation from the agreed behaviour is noticed. The reasons for deviation need not be malicious, but may be due to for example changed policies on the participant's side. This function should be performed by a neutral third party, a separate monitor function [MJDP02]. The monitor member of the community is interested in the fulfilment of the contractual obligations and the work-flow specified in the community choreography, not in the needs of any single party. If the community coordinator is a neutral third party, the coordinator may also perform these neutral monitoring functions.

From the member security point of view, inter-enterprise monitoring is not sufficient to protect the member enterprise. For example, if a peer deviates from the agreed choreography and sends a malicious payload, inter-enterprise monitoring is not usually able to stop the payload[4]. Another case where inter-enterprise monitoring is not sufficient is with implicit communities. In that case, there is no formal contract agreement and thus there can be no inter-enterprise contract monitoring.

Introspective monitoring function is thus essential for all collaborative enterprises for their member security needs. In implicit communities, the introspective monitoring function can not compare the peer behaviour to any common contract, but it can compare the peer behaviour to its subjective behaviour expectations, syntactical or semantical. For example, in the previous electronic mail relay example, the receiving party has an expectation founded on technical standards on how the SMTP protocol and the community choreography works and thus the conformance can be monitored.

The role and positioning for introspective monitoring is visible in the structure of the Pilarcos middleware discussed earlier in Figure 2.4. However, it is interesting to note that there is currently no neutral component responsible for inter-enterprise monitoring. This is one future research direction. However, it is best tackled when it is clear what is the role and function of introspective monitoring and what (if any) services it can provide to the upper levels of the monitoring stack.

2.8 Challenges and Solutions

The aim of this thesis is to develop a solution for community security challenges. These challenges in community security management have been discussed previously in the description of

[4]This is of course possible if every message in the community goes through a central monitoring service, but this is not always possible or sensible in a modern distributed business network.

the environment and the threat analysis. At this stage it is useful to collect these challenges and analyse what requirements they bring to the community security solution.

2.8.1 Challenges

The challenges can be seen from two different points of view, the community perspective and the member enterprise perspective.

From the collaborative community perspective the key threats and challenges are:

1. Participant organisations may be or turn malicious towards the community or towards one single participant. It is not possible to know or predict which parties will start acting maliciously. Being malicious towards the community as whole can cause disruption in the correct functioning of the community and thus deprive all or some participants of its benefits. If for example in our travel agent community the travel agent turns malicious towards the whole community by sending fraudulent orders, the hotels and tour operators do not benefit at all, on the contrary. The possibilities of malice and fraud vary based on the characteristics of the community. There may also be several malicious parties in a community, even acting in collusion.

2. The participants may have different goals in their community participation. For example, most businesses have only one goal – to maximise revenues. Other organisations may also have more subtle agendas, such as protecting their reputation and other assets while engaging in collaboration. Organisations with malicious management may also have malicious goals, such as collecting credit card numbers and selling them to identity thieves.

3. The participants are separate entities potentially in different jurisdictions. It is quite possible that if there is a conflict or a problem in the collaboration, there is very little practical possibility for judicial recourse.

4. There are several different types of enterprise collaboration communities, from large to small and from highly dynamic to very static. This makes it difficult to design solutions that fit all possible scenarios.

However, from the point of view of a single community member, the problems and concerns are more direct. Key concerns are dynamic collaboration with unknown parties, these parties turning or being malicious, security threats of the open environment and management overhead.

1. A member will interact dynamically with other, potentially previously unknown organisations, to gain benefits brought by the community. This means exposing some assets to harm from these unknown parties.

2. These other organisations may be or turn malicious towards the member. This is the same challenge as discussed above from the community point of view, but the focus here is on the question, what can this malicious party do to the individual member? Here the result may be financial or reputation loss for the participant organisation but otherwise the community can still function and other participants may keep their benefits. For example, if in our travel agent community example the scuba diving operator turns malicious towards the client hotel and sends a confirmation for a requested diving tour with a price it does not intend to honour, the travel agent and the hotel still do get the money and commission from the airfares and the hotel reservations. It is also possible that two or more participants are malicious separately or in collusion against one member in the community.

3. The threat environment is getting more difficult every day, especially in organisations utilising Internet-based communication. Particularly open and dynamic systems must face security threats that the open participation and dynamic community relationships bring. The facts that anyone from anywhere in the world could join the community and that yesterday's community member can be today's outsider, can cause many problems for example from authorisation and fraud prevention point of view. Additionally, in the Internet-based application environment tens of vulnerabilities are found in software components every week. Most of these vulnerabilities are publicised in such a manner that exploit code for utilising these vulnerabilities is either directly given or otherwise obviously deductible. This means that any security problem found in a publicly accessible system is immediately exploitable by thousands of malicious hackers. While good design and coding practises can diminish the probability of vulnerabilities, it is very likely that bugs in the application code cause some security problems during the application lifetime.

4. Community participation may require too much manual management in order to be cost effective. While business community participation is intended to get financial benefits such as increased sales, the price of the required applications, hardware and the management effort may be prohibitive. Especially day-to-day system management and administration manpower is very much a restricted resource in today's enterprises. The cost-benefit analysis can be very simple for example in our holiday eCommunity network. If a hotel sells one extra room per day by participating in the community, participation and its management should not cost more per day than the revenue from that sale.[5]

These challenges mean that community collaboration needs to have specialised community security solutions that are based on more than preventative controls of perimeter protection and statically managed communication restrictions. The community collaborative framework must include technical means to easily and simply manage and secure open collaboration in an dynamically changing environment where previously well behaved parties can turn malicious.

2.8.2 Requirements for Solutions

In the face of these challenges, the community security framework needs to have mechanisms for both securing the community but also mechanisms to protect an individual member of a community. This thesis focuses on community member security, which is a subset of the general community security problems. The protection needs of the community as a whole are left for future research.

In the view of the discussion above the community member security management solution in the overall community framework needs to fulfil the following criteria:

1. It must work with an open community structure, i.e. the principle anyone technically capable could join. This requirement is selected for the maximum applicability and scalability of the framework. What works with open communities works also with closed communities.

2. It must work in a dynamic environment without continuous manual intervention. By dynamic it is here meant that community participants can change during the community lifetime, potentially very rapidly. This requirement is the cost-efficiency criterion to reduce management overhead. Additionally, agility in business decisions can be a financial advantage.

[5]Since the economics of collaboration is not in the focus of this work, this example ignores such business management issues as fixed vs variable costs, profit margins and return of investment calculations.

3. It must include provisions to manage situations where a previously well behaved peer turns malicious, for example by using detective controls such as monitoring.

4. It should be minimally service or application specific. This requirement means that the solution should be deployable in a variety of communities and services, from travel agent systems to manufacturing procurement.

5. It should nor rely on the business application security controls since applications themselves can contain security vulnerabilities. Attacking these business application vulnerabilities can circumvent other security controls in the application.

What kind of a system fulfils all these requirements? For the first and second requirements, the open and dynamic structure rules out any systems where each community participant must be first approved and entered into some kind of database before collaboration is accepted. This means that authentication and authorisation controls with pre-existing static relationships only, can not be be used. The new system must support a way for previously unknown entities to start participating in the community while managing the risks this approach brings.

The third requirement means that the risk management approach requires some kind of monitoring to observe peer behaviour and to detect possible maliciousness. Members must monitor the collaboration and weigh the benefits of each community transaction request against any behavioral hints of maliciousness from the transaction originator. Since the goal is to protect a single member as opposed to the whole community, the solution is tentatively placed into the service or technology level in the community protocol stack discussed previously. There are several interesting challenges in the upper level community security as well, but the focus of this thesis is on these lower levels.

The last two requirements for application generality and independence requirement means that we can not assume the protected applications or services at the community member to include any specific protection mechanisms or interfaces. The protective system must work outside the service or application. However, it should be investigated, whether we can gain any insight into the application semantics using non-invasive means.

Since current systems have deficiencies in fulfilling these criteria, we must look elsewhere. One route is finding suitable "soft computing" approaches for modelling aspects related to human behaviour in collaborative organisations and to handle the issues of decision making and behaviour management in the context of incomplete and imprecise knowledge [CMA05]. This route fits very well with the requirements above, since human decision making is open, dynamic, somewhat scalable and not tied to a particular application domain. The degree of trust to other people is often used as a factor in making these decisions.

The hypothesis presented in this thesis is that a system combining trust in making authorisation decisions, a monitoring component to detect problems and a feedback mechanism from the monitoring component to ensure dynamism in the trust evaluation component is a suitable solution for the community member security management problem and these specific requirements.

With that in mind, in Chapter 3 trust and its transformation to computing systems is explored in more detail to explore whether it is indeed suitable for our needs. Following that, more "hard computing" methods and approaches to introspective peer behaviour monitoring are discussed in Chapter 4 so that both of these can be later combined into a framework solution.

Chapter 3

On Trust

Trust has been a very awkward concept since it is silently embedded in many aspects of human behaviour and, as with quite many other behavioural issues, it and its effects are not very well understood. However, since trust is a part of the basic decision making framework for humans, it is important to try to understand it. It is also important to understand it from a system automation and computing point of view, since computing agents become more and more autonomous also in enterprise communities. Autonomous operation requires the ability to make decisions in the agent without human intervention and thus the human decision making system, including trust, becomes of interest to computer scientists.

There are several interesting questions in and around trust. First of all, the foundation: what is trust, how can it be defined and what kinds of trusts are there. From the utilitarian point of view, there are three separate problem areas around trust. The first one is to define the facts that support trust, the second is how to find the appropriate rules to derive consequences of a set of assumptions about trust, and the third is how to use information about trust to make decisions [Dem04]. It is interesting to note that these problems can be investigated separately, and thus different solutions can be applied to each question.

In this chapter the foundations and theory of trust and mistrust are first explored and discussed in enterprise setting. Secondly, different trust models are explored and new vocabulary and an ontology is developed. A shorter version of these beginning chapters has been separately published [Vil05]. Finally, questions related to trust management in an enterprise setting are discussed.

3.1 Trust Characteristics

To fully understand trust, some characteristics of trust need to be first explored. In the real world trust decisions are made by people from their own viewpoint, thus trust is very much subjective. If Bob is considered worth trusting by Alice, Carol does not have to trust Bob. So, to evaluate a trust condition, a role of some subject must be assumed and the trust must be evaluated from that particular perspective. The subject is called *trustor* and the target of trust is called *trustee*. From this subjectivity the lack of global trust is inferred, i.e. in practise there are no entities everybody trusts. From this basic property, other natural characteristics follow:

- Trust is not symmetric. If "Alice trusts Bob", it does not automatically follow that "Bob trusts Alice".
- Trust is not distributive. If “Alice trusts (Bob and Carol)”, it does not follow that “(Alice trusts Bob) and (Alice trusts Carol)”.

- Trust is not inherently transitive. If "Alice trusts Bob" and "Bob trusts Carol", it does not automatically follow that "Alice trusts Carol".

Of course, symmetric and transitive trust are achievable in some special cases, but as a binary relation over a group of peers trust is not symmetric or transitive. Mathematically, the relation "A trusts B" over the group of peers is not a function, since the relation may not be total: some peers may not trust anyone. Additionally, some peers may trust several other peers, so the trust relation is not functional.

There have been many verbal definitions of trust, one of the most widely used definitions of trust comes from Diego Gambetta, a sociologist:

> trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action (or independently of his capacity ever to be able to monitor it) and in a context in which it affects his own action [Gam88].

This has influenced the thought of Mui et al:

> Trust: a subjective expectation an agent has about another's future behaviour based on the history of their encounters [MMH02].

Another earlier example of a trust definition comes from Audun Jøsang:

> Trust in a passionate entity is the belief that it will behave without malicious intent ... Trust in a rational entity is the belief that it will resist malicious manipulation by a passionate entity [Jøs96].

In the field of computing it is not sufficient to just define trust, as above, it is essential for automation that the concept of trust is represented by a trust model, which can be utilised by systems. Here a trust model is defined to be *the formal or computational realisation of the used trust definition, verbal or implicit*. The word computational is used here loosely, meaning a model that can be utilised by computer applications. Several trust models and their characteristics are analysed in depth later in Chapter 3.4.

3.2 Different Types of Trust

McKnight and Chervany suggest after investigating quite many trust definitions that no single definition is able to capture the rich semantic field of meanings of trust. Therefore trust should be characterised as a set of inter-related constructs. They define six different types of trust: trusting belief, trusting intention, trusting behaviour, system trust, dispositional trust and situational decision to trust [MC96]. While these six types are not the only trust concepts, they encompass most of the discussion in the trust research.

Trusting belief is the extent to which we believe another party to be trustworthy in a given situation, i.e. being able and willing to act in our best interest. The trust definition of Audun Jøsang is focused on this type of trust. In people, this depends on their cognitive beliefs about other persons. Trusting belief is both person and situation dependent and is driven by personal belief formation processes.

Trusting intention is the extent to which a party is willing to depend on another party in a given situation and taking into account the potential risk. Trusting intention is, as its name implies, intentional dependence of someone or something and it is also very situation specific. The trust definitions of Gambetta and Mui et al above have elements from this category.

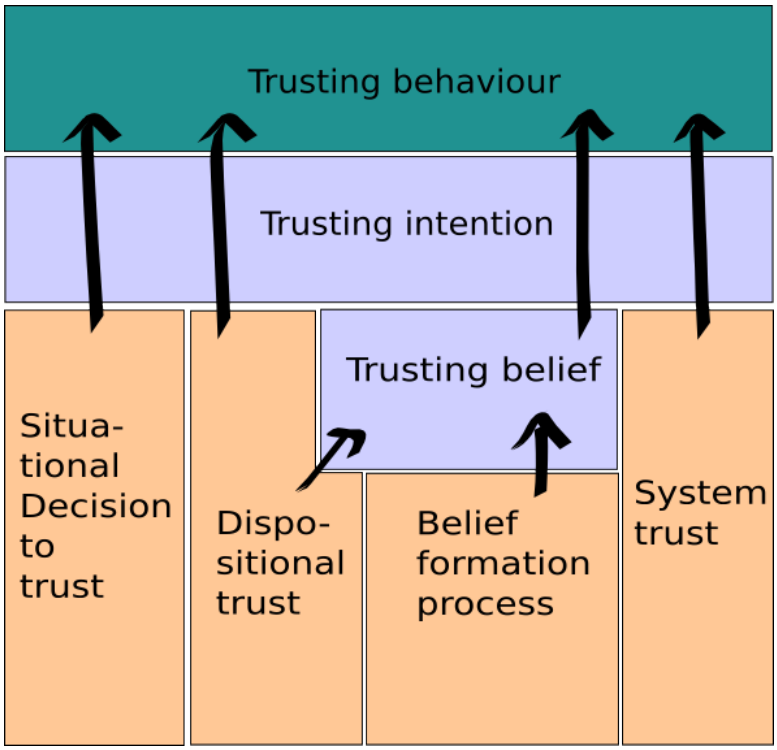

Figure 3.1: Relationships between the trust constructs [MC96]

This intention continues to *trusting behaviour*, which is the above mentioned intention translated to actual behaviour, i.e. actually participating in the action. This is separate from the intention, since intention does not always lead action. Trusting behaviour is situation dependent, i.e. the degree of trust varies based on the action or situation in question.

System trust is the extent to which we believe the proper impersonal structures are in place for us to anticipate success in a future endeavour. These impersonal structures are twofold: on the other hand constructs like laws, regulations and contracts and on the other hand situational normality. Laws, regulations and contracts give us a safety net if the personal transaction has problems, and knowledge about situational normality reduces our uncertainty and thus enables us to perhaps take more risks by trusting. Here the word *system* seems to be semantically closer to a concept of political or regulatory system instead of a computer system.

Dispositional trust is a generalised expectation of trustworthiness of other people or entities. There are people who do not easily trust others as well as those whose basic outlook to life is to trust others. The reasons for the different attitudes may be deep in the early psychological experiences or even in the genetic make up of a person. Corporations can also have dispositions to trust, depending on the experiences of the management and also the competitiveness and the competitive styles in the market segment.

Situational decision to trust does not consider the other party when making the trust decision but only the particular situation is considered. This means that we decide always to trust in a given situation regardless of the trustees involved because the benefits of trusting are bigger than the potential risks. Situational trust decisions can be made either because we can not know the identities or characteristics of the trustees, or we choose to ignore the information.

These constructs are not independent, on the contrary there are influencing relationships be-

tween the constructs. For example beliefs about the trustworthiness of other persons create intentions which are converted to behaviour. On the other hand, system trust is independent of our belief to the trustworthiness of other persons, but influences our trusting intentions directly. See Figure 3.1 for an illustration. This key figure shows how the constructs in the lower levels of the stack have an effect to the constructs in the upper layer. This figure is later used and modified in Chapter 3.5.2 to illustrate trust management in enterprise systems.

3.3 Trust and Mistrust Types in Enterprise Systems

The system developed by McKnight and Chervany seems quite useful. But for the focus of this work, the first question is whether the general approach developed by McKnight and Chervany is applicable in computing and enterprise systems?

Their approach was based on a cross-disciplinary literature based survey. This means that the subject has been looked at from different perspectives, not just from the computer science and automation point of view. Thus their system is a distillate of many research efforts in many different fields. This cross-disciplinary background is important for several reasons.

First of all, by using a system developed in a non-computing environment, the bias of looking at trust only from the sciences and computing perspective is avoided. This has been previously identified as a problem in current computational trust models, which are not sufficiently grounded on understood social characteristics of trust [MMH02].

Of course, this view introduces another bias, looking at trust from only the non-computing point of view. This latter bias is corrected by investigating, how to apply this theory in a computing environment. Additionally, the diversity of the background is important when the aim is to develop a trust model for general enterprise systems, which are not bound to any specific usage scenario. Of course, this strength is also a weakness, sometimes a simpler view of trust and therefore simpler implementation could suffice if the scope of the application domain is narrow enough.

Taking into account that the work of McKnight and Cherveny is relatively recent and especially because of its sufficient background in the real world usage of the word and concept "trust", their work is selected as one starting point for this thesis.

The second question is the more practical issue, how does an impersonal enterprise utilise these human behaviour based constructs in computer systems? Especially interesting is the key question of computer science as applied to trust: which of these trust constructs can and should be automated and to what extent? Therefore, it is useful at this point to discuss these constructs in relation to computers and systems, especially in the enterprise community environment.

System trust for the impersonal structures, such as laws and contracts can be seen in enterprise computing from two different perspectives, outside and inside. First, the outside perspective looks at the structures and safety nets outside of the enterprise computing systems. Issues like the existence and applicability of suitable laws, regulations and general contract frameworks between enterprises are factors that the enterprise management uses in the risk analysis.

Secondly, a case can be made for the existence of inside perspective as well. This perspective would looks at the impersonal structures and safety nets from the computing point of view. Do we, the enterprise or the community, have enough *technical* capabilities and safeguards to make the collaboration successful and recover if there are problems? These capabilities would include technical security frameworks and mechanisms in place for the collaboration. For example, if the transactions in the collaboration should be confidential there is very little system trust to the community if there are no technical safeguards, such as encryption, against eavesdropping. This inside aspect of system trust has not been considered in the original work of McKnight and Cher-

vany since their focus has not been on utilising trust via technical means. However, the realm of technical safeguards fits within the concept of "impersonal structures" described by them in their definition of system trust, although their main point has been to classify into this category everything outside the realm of the trusting person himself/herself.

As an example in the inside aspect of system trust in the Web Services area, the Web Services Trust Language initiative [ABB+05] tries to enhance system trust in the Web Services environment by creating a security model on top of other Web services standards where communicating peers prove various claims, such as identity, by sending and receiving security tokens. WS-Trust defines mechanisms for issuing, exchanging and validating these tokens. WS-Trust is an system or infrastructure service on top of which other trust services may be built.

Both the outside and inside perspectives jointly create system trust to the collaborative process. However, since the factors include outside issues such as laws, the actual decision whether to actually start participating in communities in a specific technical environment seems best to be left to management and not automated. If there is lack of trust to the laws, contracts or technical capabilities around the collaboration, an enterprise seldom enters into the collaborative process.

However, in addition to the outside and inside impersonal structures discussed above, the situational normality needs to be considered as well for system trust. Situational normality or status information can be translated in the computing arena as *context*: any information that describes our current situation. It is also possible to view the inside perspective of system trust, i.e. our tehcnical abilities for successful co-operation, as context information, although the "situational normality" then covers very different time frames from years to minutes and seconds. Gathering this context information can be automated, but decisions about the limits of situational normality may need guidance from management.

A similar automation problem can be seen in the *situational decision to trust*, which does not utilise personal trust relationships but requires information about the attempted action or service. In computing systems this type of trust is utilised in open Internet services, such as the e-mail relays or public web sites, which are open to all potential peers, regardless of their identity. When an organisation does not have control of the connecting peers and their characteristics in the situation, the decision to trust in a particular situation becomes a policy issue and thus must be made by the management after careful deliberation of the risks and the rewards.

Especially enterprises using situational trust are quite vulnerable to malicious parties, so they must have the appropriate monitoring mechanisms in place to detect any misuse. Since some measure of situational trust must be used when participating in the current Internet environment, it is not a great surprise that many enterprises have deployed intrusion detection systems to monitor the inevitable misuse of the situational trust.

Dispositional trust, i.e. trustor's general attitude towards people, is best translated to computing environments as some kind of default or initial value for trust to other parties. An example is a default purchasing limit for new peers or default trust score, if no other value can be deduced from past experiences or recommendations. This is not a situation dependent construct but a basic attitude or value and thus should not be automated but again best determined in enterprises by management after a comprehensive risk analysis.

Trusting belief reflects how trustworthy we think the peer is in this particular situation. As seen from Figure 3.1, trusting belief in humans is created by a personal belief formation process. In computer systems this process can be simulated by automatically gathering trust decision support facts and creating a trust score based on that information. The variety of these support facts are discussed in Chapter 3.4. In a simple trust use scenario, where the peer is trusted based on their identity information only, the highest score can be given to authenticated peers and the lowest score to unauthenticated users.

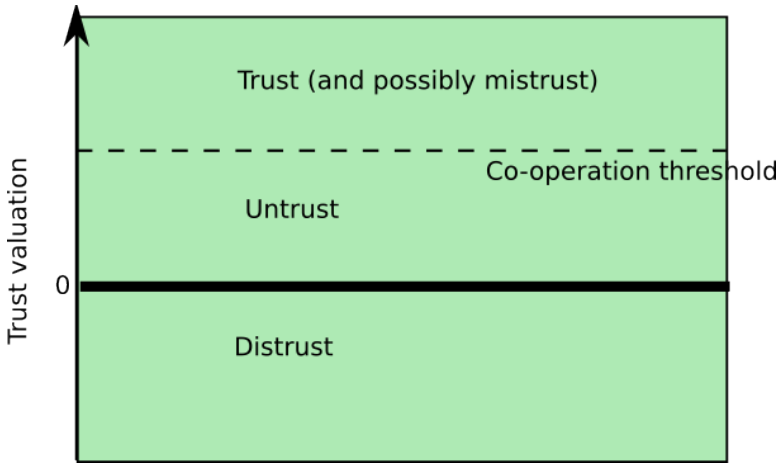

Figure 3.2: Relationships between variants of not trusting.

Since peers seldom have absolute belief to the other peers' trustworthiness, and the past behaviour of a peer is no guarantee to the future behaviour, all aspects of not-trusting must be managed. This concept of not-trusting includes concepts like mistrust, distrust and untrust. As with any developing area, there is some confusion over definitions of mistrust and distrust. For clarity, this work uses the definition of mistrust as *misplaced trust* and distrust as *negative expectation regarding another's conduct* [MD05]. Mistrust happens when a peer we actually trust misuses our trust by doing something harmful to us. This must be detected and managed by having the introspective monitoring system in place. Distrust means lack of trust and untrust is the uncertain area of trusting somewhat but not enough to engage in collaboration and co-operation. The relationships between distrust, untrust and trust/mistrust are displayed in Figure 3.2.

Sometimes only the peer trust score is used in the decision making process. In the simple trusting belief example above, authenticated users were decided to be trusted. However, there are two additional trust constructs that can utilise this trusting belief: trusting intention and trusting behaviour.

In the field of enterprise computing, *trusting intention* dictates how much the enterprise wants to depend on someone in the current situation to achieve something. In traditional single services this can be thought as behaviour to which some limitations are placed. However, this has more meaning in communities, where this form of trust is the key. In communities trusting intention may dictate how tight partnerships an enterprise is willing to form and what kind of contracts it is willing to negotiate.

Trusting intention declares the enterprise's will in community interactions and how deep dependence relationships to form. Since this is a decision based on the corporate strategy and vision, it also is best left to be decided by the enterprise management, i.e. people. The strategic decisions can then be transformed to contract templates or bounds to the automated contract negotiating process. It will define the type of contracts or communities the enterprise is entering.

The five trust constructs above are such that in enterprise settings the best source of the trust is a passionate entity, a human being in the management chain of the enterprise. However, trusting behaviour which draws input from these lower level constructs via trusting intention is something that can be automated. In the enterprise field, this type of trust manifests itself as actual participation in a community and fulfilment of the contractual obligations if there are any.

The conclusion at this point is that the approach of McKnight and Chervany can be applied in

theory to computing and enterprise systems. Therefore one of the premises of this work is that once the trusting intention is known, the translation of that intention to trusting behaviour, i.e. to trust-guarded interaction can be automated. Thus the question how to achieve trusting behaviour using automation becomes the most interesting followup question for computer scientists. It should eventually be evaluated, whether this approach works in practice.

These questions has been researched in the past in conjunction with different underlying trust utilising systems. The next chapter surveys different trust models and trust systems in computer science and classifies those based on the information they use to make the translation from intention to behaviour.

3.4 Trust Models

Since there is no universal definition of trust, the developed trust models and systems relying on those models have been very different in both verbal and formal definitions of trust, and also regarding the vocabulary used in the trust models. This has been noted and explored by McKnight and Chervany previously in 1996 [MC96]. However, at this date it is beneficial to revisit this area by summarising recent work, classifying various models according to their emphasised trust feature sets and describing the examined trust models with a common language. This section presents such a language and gives examples on how various research models can be positioned within the descriptive system.

This section focuses on only one trust research question, defining or gathering the facts that support trust. Because different trust models emphasise different trust support fact and ignore or abstract away others, it is assumed that a union of these emphasised facts across all models provides us with a maximal set of trust support facts. This maximal set is then utilised to provide us with an extensible ontology.

3.4.1 Trust Model Research

It is important to note that since trust is a complex concept, all the models are by necessity simplifications of that complexity and different models streamline the concept differently. The key simplification of the models is that they try to facilitate some kind of decision making, i.e. trusting behaviour by McKnight and Chervany's categorisation, even if the actual trust definition has elements from other trust categories.

Early attempts to formalise trust for computer use have been in the context of authentication. Yahalom et al. developed a trust model to be used in authentication scenarios [YKB93]. One of the earliest attempts to define trust from the general and computational point of view was Stephen Marsh's thesis [Mar94], which drew input from the sociological trust research. After that several interesting trust models and also systems, such as PolicyMaker [BFL96], KeyNote [BFK99] and REFEREE [CFL$^+$97] have emerged. Around that time also Abdul-Rahman et al. [ARH97], Daniel Essin [Ess98] and Audun Jøsang [Jøs96, Jøs98] described their trust concepts and models.

In recent years the focus has been on more comprehensive and concrete systems having wider trust management elements, such as Poblano [CY00], Free Haven [Din00], SULTAN [GS02], TERM [BZ02] and SECURE [CGS$^+$03, CNS03]. But since the applicability of trust has been widened to cover more than authentication and authorisation, they build on the research done earlier, but do not necessarily use the same terminology or basic components. Additionally, their focus is not necessarily in general usage trust management, but they have tried to explore trust in various application domains, such as peer-to-peer networking or web applications.

MODEL	Identity	Action	Business value	Capability	Competence	Confidence	Context	History	3rd party
Abdul-Rahman	X	X							X
Essin	X	X	X	X	X		X	X	X
Free Haven	X				X	X			X
Jøsang	X					X			
KeyNote	X	X							X
Marsh	X	X	X		X			X	
Poblano	X		X		X	X			
PolicyMaker	X	X					X		X
REFEREE	X	X							X
Sultan	X	X							X
TERM	X	X				X		X	X
Secure	X	X						X	X
Yahalom et al	X	X							X

Figure 3.3: Trust model input factors

These thirteen models discussed span a time frame of ten years of trust model research. They include all key trust models and form a comprehensive set of research models. Thus this set is used for the actual feature analysis. All models take some kind of input to calculate some kind of trust score or valuation. Therefore the classification is done using two orthogonal axes: input factors and output type.

3.4.2 Trust Model Input Factors

The key differentiating element in trust models is the list of factors or facts that are required or used for evaluating trust or building the trust score. This list is usually given as an n-tuple. This tuple can contain several elements, some of which can themselves be tuples.

Because of this factor diversity, it is impossible to give a simple taxonomy where each model sits squarely only in one classification box along this axis. Instead, we have defined a classification of factors and each model can be described by what set of factors it is using or, in our terminology, is aware of. A summary is shown in Figure 3.3.

Identity-aware Models

As a basic tenet we have found that all reviewed systems are identity-aware, i.e. they assume to have some identifying information on the target of the trust evaluation, trustee. This identity awareness does not mean that the models must know the real name or other global information of the communicating peer. In some models it is quite sufficient that the identity is only locally unique and temporally sufficiently stable so that it aids in recognising the same entity in this system over time. Therefore we can find models that use globally unique identities or locally unique identities. The vocabulary for the trustee varies according to the model, so in the context of this chapter trustee, peer and principal are interchangeable.

The Poblano system uses *peerID*, which needs to be unique across the universe of peers [CY00]. KeyNote [BFL96] can use any string as principal identifier, in practise it can also be a string representation of the principal public key. An example of local identities can be found in the work of Abdul-Rahman and Hailes [ARH97], where the trust relationships are local and there is no such thing as a global map of trust.

Action-aware Models

Most trust models have noticed that there is an action component to trust. That is, the actual trust evaluation and decision depends on what the target of our trust is trying to do in or with our system or for what purpose we are trusting the target. This is very intuitive, as we may trust one party to relay our messages but not to transfer any money. This is also reflected in the definition by Gambetta in Chapter 3.1 above. On the other hand, the action component is not always necessary, since there is a concept called *general trust* which is an unqualified trust towards a principal [Mar94].

If we have a trust model that takes the action into account, the set of actions can be closed or open. Closed means that there is a set of pre-defined actions the model supports and no others can be defined by users or organisations using the trust model. An open set of actions means that the model offers a way of defining at least some of the actions or does not restrict them to a particular set.

This concept has many names in the actual models. PolicyMaker [BFL96], REFEREE [CFL+97] and KeyNote [BFK99] call this *action* and Abdul-Rahman et al. [ARH97] have a *trust category*. Yahalom et al. use a pre-defined set of *trust classes* for which we trust a certain principal [YKB93]. SULTAN [GS02] uses the name *context*, but in its core the the definition is about action and action sets. Similarly, the SECURE project uses context in the meaning of action [CNS03]. Essin has the concept of *activity* [Ess98], although it is not used in the trust evaluation directly, but as a sub-component in determining the capability of trust subject and subject reputation.

However, not all models use the action factor explicitly. Some specialised systems, for example Poblano [CY00] and Free Haven [Din00], use the trust valuation in relation to data content received from a network peer. There the actions are implicit in the system definition, but since there is only the one basic action type and the action system is closed, we do not consider Poblano or Free Haven systems action-aware.

Marsh describes *situation*, which is a point of time relative to a specific agent [Mar94], i.e. the principal evaluating trust. This definition includes the actions the other principals are attempting, therefore Marsh's model is action-aware.

The TERM system [BZ02] has no action concept as such, but trust calculations are related to *roles* in role-based access control. There are two roles, an access role and a testifying role, so the system has two predefined actions. Hence it is action-aware, although the action set is closed.

Business Value Awareness

Between people trust implies potential loss and also potential benefit. In several models there are concepts called *risk*, *benefit* or *value*. These are all associated with a particular action and try to give impression on how the action can help us or how the misplaced trust can hurt us. Risk is the most common modelled business value element. However, to understand the full impact of the attempted action, this needs to be balanced against the potential benefits or value of the action. We combine these under the common concept of *business value*, since all these concepts try to model the potential impact, positive or negative, of the attempted operation.

Marsh is using three separate business value concepts: utility, importance and risk. Utility is a measurable benefit, importance a subjective valuation of the importance of the action. These both are used in evaluating situational trust, i.e. trust in a specific situation or action. Risk is used in determining the co-operation threshold, i.e. whether to actually engage in the action in a particular situation [Mar94].

The model by Essin also uses several business value factors. He uses the concept of valuation as the cost of the resources or assets affected by the action. He also uses *stake* which is the degree

to which the entity(s) proposing to engage in the activity has a vested interest in the outcome [Ess98]. Stake tries to measure the level of commitment for this action and thus it is quite close to the concept of importance. He also uses the concepts of risk and benefit, although his model combines these to a single risk/benefit set.

In the Poblano P2P system [CY00], risk is a statistically computed metric of peer accessibility and performance, both concepts relating to network metrics. Here the considered risk is of the type "risk of not getting the information" instead of any loss of data or money. This is however, well within the defined use of "risk", so we consider the model business value aware. Poblano also uses *importance*, which is the importance in engaging in the activity, in calculating the co-operation threshold.

The SECURE model does not have risk as part of the formal trust model, although the resulting system has an added risk and cost/benefit analysis as part of the trust evaluation. Therefore the model itself is not business-value aware.

Competence-aware models

One type of trust decision factor is information on the competence of the subject with regards to performing a particular action. When human clients are considered, this is an important decision factor, but with automated clients at least some degree of technical competence in following the specification should be assumed. Therefore this factor is not very common in the reviewed models.

Competence is considered by Marsh in calculating the co-operation threshold based on trust [Mar94]. Grandison and Sloman use a verbal definition of trust where competence is a factor, but their computational trust model in the SULTAN system does not include competence as such [GS02].

One interesting variant of competence is lifting the competence evaluation from the technical to the semantic level. The Poblano model has a concept of *CodatConfidence*, which is a measure of semantic experience, i.e. the system's competence in providing us with relevant information [CY00]. Similarly, the Free Haven system [Din00] has a concept of *metatrust*, which signals that the data received from a node is indeed valuable information. For example, if the Free Haven system agrees with a recommendation from a third party, the third party metatrust is incremented.

This concept of ability to perform a task is also called *capability* by Essin. He defines capability as measurable expertise that the entity possesses about the activity [Ess98]. In this common vocabulary this idea of expertise is called competence.

Capability-aware Models

Capability has a dual meaning in the security and trust research. On one hand, capability is considered synonymous with competence, i.e. evaluating the peer ability to perform a certain task. On the other hand, a capability has a very specialised meaning in the field of security as a token given to a peer to access a resource. We differentiate between these two concepts and here define capability as a form of an access granting token.

This latter form is also used by Essin. In addition to the capability as competence definition, he defines capability also as demonstrable access and authority necessary to act [Ess98]. This is capability in this second sense.

Confidence-aware Models

As input for the trust calculation can be received from multiple sources, sometimes also as external recommendations or reputation, we may also have uncertainty associated with trust or the trust

input factors. The concept of *confidence* reflects this uncertainty, although in the actual models this concept has many names.

The TERM model [BZ02] uses the concept of an *opinion*, meaning how much the calculating TERM server believes the trust statement. The Poblano system also uses a confidence value, *PeerConfidence*, in determining whether the trust subject is able to co-operate and thus being trustworthy [CY00].

Confidence can also relate to only one of the trust input factors. Essin uses the term *certainty* that the true identity of the trust subject is known [Ess98]. The Free Haven system [Din00] uses the term *confidence rating*, which is used as a measure of how fast or slow external recommendations change our trust value in regards of that particular entity.

Context-aware Models

The trust evaluation may depend on the evaluating system internal or external status at that particular point of time, i.e. context. If, for example, the organisation firewall is experiencing heavy port scanning activity or is otherwise deemed being under an internet based attack, it may be sensible to lower the trust valuation on all or some principals to limit exposure to potentially malicious activity.

Interestingly enough, this is not a very widely used factor, Essin has it as a sub-factor in determining the action valuation and subject stake in the action [Ess98]. In PolicyMaker, the policy is defined in an interpreted programming language and it thus can obtain some context information. The set of local policies are considered the context under which the trust is evaluated.

The SECURE model also defines trust via a policy construct. Theoretically it may be possible to include context information in the policy, but since this option is not explored in their work, SECURE is not classified here as context-aware.

The SULTAN model [GS02] uses the term context to mean the action to be performed (see Chapter 3.4.2), not the system state or circumstances. Therefore SULTAN is not context-aware by this definition.

History Awareness

When people interact, one of the key components in our trust evaluation is the trustee past behaviour or track record. Some models try to capture this idea as well. It can be modelled using vocabulary like *experience* or *evidence* or *local reputation*.

Reputation is opinion or view of one about something [SS00]. There can be two types of reputation, subjective reputation is reputation calculated directly from the trustor direct experiences [SS00] and external reputation is reputation received from third parties. The former type of subjective reputation is considered in this history awareness category.

Essin uses the word *reputation* but does not differentiate between subjective and external reputation [Ess98]. The Marsh model also includes past history data by including the trust values in all previous similar situations in the situational trust evaluation [Mar94].

The TERM model [BZ02] includes a concept called *direct experience*. It is used by the system to evaluate trust opinions, although it is not formally defined. The SECURE model is very flexible so that it can also use history information if required by making history part of local trust policy [CNS03].

Third Party Awareness

A trust model can be open or closed. A closed model does not take into account any input from outside the actors involved in the trust evaluation (i.e. the trustor and the trustee). Open trust models accept information from third parties. This information can be in the form of external reputation, recommendations or even delegated decision making.

In addition to subjective reputation discussed above with regards to history-awareness, we also have external reputation. For example, we may belong to a community, which has a common reputation service with shared ontology and receive reputation information from that external source. This external type of reputation is also considered in this third party awareness category. This is not a widely used feature. In the Essin model there is reputation, either generally or bound to a specific action, as a component in the trust evaluation [Ess98]. In his model, reputation can be either subjective or from an external source.

Recommendations are conceptually somewhat difficult. Since trust is not transitive, recommendations should not be taken directly as trust. However, third party recommendations can be an influencing factor when deciding about trust. Recommendations are considered by many models. For example the models by Yahalom et al. and Abdul-Rahman et al. use them. The SULTAN model and system also uses recommendations as a basis for new trust relationships. The Free Haven system uses the word of *referral* instead of a recommendation. The TERM model also uses recommendations, although it is said that they are used indirectly because of the transitivity issue.

Delegated decision making is not very common. The SECURE model is third party aware because it can handle *delegation* where a principal can refer to another principal's trust information.

This external information can be carried in the form of a credential. A credential is simply a statement, purportedly made by some speaker [CFL+97]. These statements, such as a name or proof of membership, can be used when deciding about trust. A credential is not a semantic information category, but a technical one. A credential can carry identity information, subject properties, reputation or even capabilities. Therefore we do not categorise models as credential-aware, but the classifying factor is the semantic category such as identity or a particular property. The TERM model calls these credentials *evidence*, and these evidence statements can carry many types of information. For example, a X.509 certificate signed by a certificate authority is a credential testifying a particular name and whose validity must be checked using a third party.

However, in some systems credentials can be executable programs. This is true in the Keynote and PolicyMaker systems, where these programs are called *assertions*. In the REFEREE system a credential is also a program that examines the initial statements passed to it and derives additional statements.

3.4.3 Trust Model Output Type

The second differentiating axis for trust models is what does the trust model produce.

Discrete Value

In many systems the computational model produces a discrete numerical value in some predefined range. This value is then passed to a separate decision component to be compared with a threshold value to make the actual trust decision.

The SULTAN model [GS02] uses a range between 0 and 100 for trust and -100 to 0 for distrust. Similarly, the Poblano system [CY00], peer trust values are integers ranging between -1 and 4. However, the final trust value for the information chain is computed by a formula and the action is allowed if the formula result is above a set co-operation threshold. The same value range is used

by Abdul-Rahman and Hailes [ARH97]. The Free Haven system [Din00] defines the used trust values as integers.

The SECURE system [CNS03] approaches this from a different angle. They utilise a set of trust values, whose elements are not predefined by the model, but can be simple binary results, trust vectors or action-trust pairs.

Logic Value

The key task for a comprehensive trust system is aid in deciding whether the trust subject is trusted enough to allow the requested operation. Therefore it would be natural for a computational trust model directly return binary yes or no answers. While this type of output can be also classified as a discrete value set, we separate this because this choice implies that an authorisation or trust decision has already been made. However, this is not very common, usually the trust value is passed to a separate authorisation system for decision.

One example of binary yes/no answers is in the early work of Yahalom et al. [YKB93]. The PolicyMaker evaluation is simple fail/pass [BFL96]. In the KeyNote system [BFL96] the assertions return also yes/no answers, but additionally they can return annotations, i.e. restrictions, with the yes-answer. The model by Essin is not explicit in its outcome specification, but notes that trust evaluations are always reduced to yes/no answers [Ess98].

REFEREE has tri-values: true, false and unknown [CFL+97]. The unknown answer is given if the trust or policy compliance can not be determined. While this is the basic return value, the system also gives out additional statements about the reason for the value.

Probabilistic Value

As seen from the definitions above in Chapter 3.1, probability is used in some definitions in describing trust and thus in many real world models as the result of trust evaluation. This probability value is then passed to some decision engine for the actual trust decision.

Even though probabilities are used in definitions, they are not used in the actual policies that much. Marsh's thesis [Mar94] and the TERM system [BZ02] use probabilities, as does the Jøsang model.

3.4.4 Towards an Ontology of Trust

The trust model analysis above gives us a wide view on how trust has been modelled in previous research. This information makes it possible to create a domain ontology, i.e. a description of the concepts and relationships in the trust domain. Trust ontologies have been made before [GPH03], but not based on a comprehensive analysis. This background analysis makes this new ontology widely usable and compatible with previous research models.

Because different trust models emphasise different features in the concept and abstract away others, it is assumed that a union of these emphasised features across all models provides us with a maximal list of trust input factors. Based on this maximal list we formulate the following ontological structure.

First of all, trust is a relationship between two principals, the subject, trustor, and the target, trustee. The trust between trustor and trustee may depend on the the action trustor is attempting. The action may have a score of business value properties attached to it. The trustor also may use context information or history data to help in the trust evaluation. The trust can also depend on the peer competence. Additionally, there can be an element of confidence attached to the trust

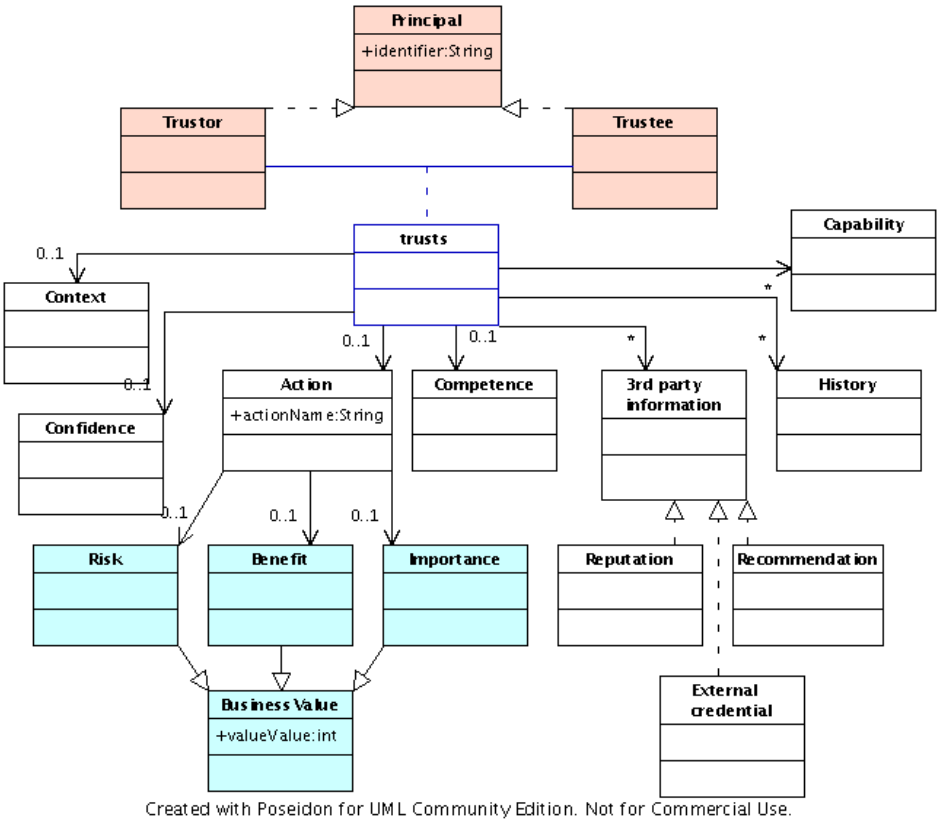

Figure 3.4: Trust as a UML diagram.

relationship. There can also be a set of third party opinions in the form of reputation information, recommendations or credentials that influence the trust evaluation.

The trust value itself is represented as an integer. All discrete and logical values can naturally be represented as integers and probability constructs could be transformed to integers within a large range (for example 0-10000) without a major loss of expressiveness.

A UML diagram modelling these relationships in trust is presented in Figure 3.4. The model in the picture is a level 1 meta-model, i.e. it models a particular application domain, trust, which is represented in a UML modelling system [OMG04]. An actual population of this model, such as defining trust relationships between concrete principals, is a level 0 meta-model and can be created utilising this level 1 model.

The conceptualisation can also be described in the OWL Web Ontology language [W3C04a] to facilitate trust model and trust relationship sharing. The UML model in Figure 3.4 can be translated to an ontology expressed in the OWL language according to ontology definition meta-model guidelines [OMG04] [1].

Putting this ontology and preceding discussion into the context of the wider goal of this thesis, we have now developed a definition for trust based on human perceptions and broken it to the constituent components people use to evaluate trust. This analysis enables us to try to transform this concept into the realm of computing and towards our goal of securing electronic communities.

It is also interesting to note that none of the trust definitions investigated use all input factors.

[1] The actual OWL description is not included in this thesis but is referred to in the published paper [Vil05]

Daniel Essin's model is the most complete and he describes an idea of a policy engine built on that model [Ess98]. His work is directed towards implementing policies and identifying conflicts within single enterprise, and facilitating sociological debate about them. While policies are an integral part of an automated system, they are not enough for securing the dynamic communities.

3.5 Trust Management

In addition to the basic trust research questions discussed above, managing trust is an interesting research question in itself. It is not sufficient to just understand trust, it is necessary to utilise and manage trust relationships in the collaborative enterprise environment. Communities and the technical agents need tools and mechanisms to manage and use the available trust information. As with the concept of trust itself, the view on trust management varies with the used trust definition and model.

3.5.1 Background and Challenges

Trust management as separate research topic began in 1996 with Blaze et al formulating trust management as *the study of security policies, security credentials and trust relationships* and the introduction of the PolicyMaker trust management system [BFL96]. The principles and goals of PolicyMaker were:

- An unified approach for all applications.
- Flexibility to support complex trust relationships.
- Locality of control to support subjective trust evaluation.
- Separation of mechanism from policy, which enables different applications to use the same verification infrastructure.

In today's vocabulary PolicyMaker solves trust management by discarding identity certification and subsequent authorisation and replacing it with a system where a peer introduces a set of credentials and the local trust management engine decides whether a peer with these credentials complies with the local security policy. If it does, it is authorised to perform the requested action. The later Keynote [BFK99] and REFEREE [CFL$^+$97] systems were in this regard conceptually similar to PolicyMaker only differing in the mechanics of credential evaluation. The four goals above are still desirable requirements for a trust management system today. However, the list is not sufficient in today's collaborative enterprise environment.

These credentials can theoretically be anything, from an airline customer reward system membership to a reputation token from another online community. In a purely credential based system, authentication is not necessary. This means the lack of a a persistent peer identity in the system, which in turn makes it impossible to attach any history or reputation information with peers. In other words, there is no knowledge whether the peer presenting certain credentials has abused the enterprise's trust previously. This lack of persistent identity is a problem in systems which must by necessity be open to remote collaboration. Theoretically this problem can be solved in the context of credential based systems by requiring that one of the credentials presents some kind of persistent and unique identification and its presence is mandatory. In practise this is equivalent with a system requiring authentication. Therefore some kind of persistent identity is a requirement in enterprise trust management.

Since trust relationships between people are not static, some measure of dynamism, i.e. the effect of trustee behaviour on trust evaluation, should be present. This requirement was not identified nor addressed by the PolicyMaker and similar systems. The first systems introducing dynamism and monitoring into trust management were TERM and SULTAN.

TERM [BZ02], took into account experience and recommendations and included evidence and evaluation into the model. The SULTAN system [GS02] defined trust management as *the activity of collecting, codifying, analysing and evaluating evidence relating to competence, honesty, security or dependability with the purpose of making assessments and decisions regarding trust relationships for Internet applications*. It is interesting to note that this management definition includes their trust model and thus mixes the two.

In addition to the trust establishment and analysis, SULTAN introduced a monitoring system, which feeds experience and other state information to the trust evaluation. In SULTAN the monitoring service updates state information related to scenarios being evaluated or from actual systems, as well as experience information, such as number of successful interactions, which can be used to increase trust level ratings directly [GS02, GS03]. However, the main focus of SULTAN has been on the logic of trust relationship analysis and the monitoring function is somewhat underdeveloped and sketch-like.

If trust management systems are viewed from the functional point of view, they consist of a number of functions and subsystems. One way of functionally dividing a trust management system is to three parts: the authorisation decision, certificate storage and retrieval, and trust negotiation [CSW06]. However, this particular division to three is not sufficient, since it does not take account the dynamic nature of trust. Therefore a feedback mechanism from observed peer behaviour to trust authorisation decision needs to be made a key component of the management system. This ensures that if a peer abuses our trust it is immediately reflected in future authorisation decisions. This thesis is focusing on binding this new feedback mechanism to the authorisation decision.

Therefore, in the TuBE project trust lifecycle has been described as the combination of initialization, usage and update, which takes into account the dynamic nature of trust [RK05].

3.5.2 Enterprise Trust Management

To learn from the definitions discussed above, enterprise trust management is in this thesis defined to include *issues relating to the initialisation, usage, monitoring and update of the trust relationships for an enterprise.*

This definition has several positive sides. First of all, it does not define trust, and thus is independent of the trust model used. For example, it can be utilised in an organisation using a third party aware trust model that delegates all decisions to third party recommendations or by an organisation utilising a context and action-aware model. Secondly, the wording "for an enterprise" includes both intra-enterprise and inter-enterprise trust relationship management. This is an important distinction, since there are two levels of trust issues in a modern community environment: the single system level and the community level. Thirdly, the monitoring and update factors take into account the dynamic nature of trust.

Thus, trust management is in this work is divided into four functional parts: trust initialisation, trust use, behaviour monitoring and reaction to monitoring observations. While this division is applicable to both the intra-enterprise and inter-enterprise scenarios, this parallels the community lifecycle phases discussed previously in Chapter 2.3. The relationship between these aspects can be seen in Figure 3.5.

This new division to four functional parts include both earlier divisions to three. The tuple (authorisation, certificate storage and retrieval, trust negotiation) is compatible because these three

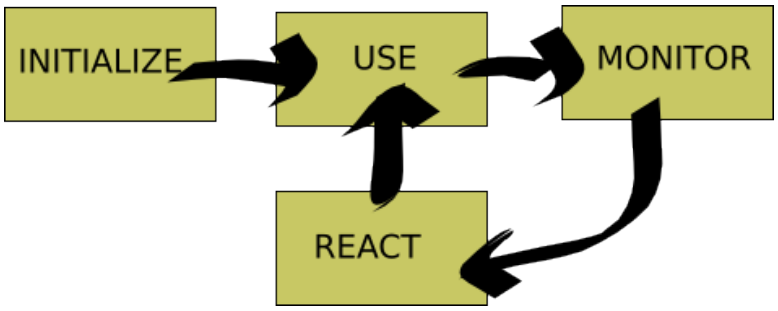

Figure 3.5: The new four part view of trust management.

components are included the "trust use" part. The tuple (initialize, use, update) is also compatible, but in the model of this thesis the update lifecycle phase has been divided into two more specific tasks, monitoring and reaction. The functions in this new four part view are explored in more detail in the following chapters.

Trust Initialisation

Trust relationship initialisation dictates how trust relationships are formed with new peers. This phase includes finding and choosing the peers and deciding how much to trust them. In a community setting these tasks have direct links to the community lifecycle phase of community population.

Although partner discovery can be done in multitude of ways, it is here assumed that in a collaborative community new peers are found from the service offer repository, where potential participants declare their capabilities to participate in various roles. The partner selection itself can be trust-guarded by accepting only trusted partners to the community roles. This assumes that there has been some kind of encounter with this partner before so an existing trust relationship can be utilised.

If the initialisation phase bootstraps the trust relationship, the key trust type in this phase is the dispositional trust. If there is no established trust relationships with the peers wishing to fill a role in the community, the trust relationship must be initialised with them. The dispositional trust, the default value for previously unknown entities, is usually the starting point for the trust valuation. The dispositional trust score should be reasonably low to prevent new peers from abusing our system, but still positive enough to enable some form of collaboration.

Previously the requirement for persistent identity was discussed. One counterargument for this requirement is that peers could switch identities after abuse and thus render the behaviour monitoring and trust feedback cycle useless. Depending on the nature of the collaboration and community populating process, this identity switch may not be feasible. But assuming it is, one of the control mechanisms against this risk is the effective use of dispositional trust. If a previously well behaved peer suddenly abuses the service, the abuse is detected and the trust evaluation is changed and causes the authorisation to be cancelled, this cancellation has consequences to the abusing peer. Forming a new identity and starting to use the service as a previously unknown peer should incur a operational penalty in the community. This penalty can take the form of low dispositional trust meaning in practise a very restricted set of allowed operations in the service for new peer. Effective types of penalties or operational restrictions could be an interesting research questions for the future.

Another refinement for this initialisation process is the use of third party information, such as

reputation or recommendations. For new peers, or old peers in new roles, recommendations or reputation information may be used to help in the trust relationship initialisation. Since trust is not by nature transitive, care should be taken to decide how much third party opinions affect the trust relationships. In one end of the spectrum there is the total delegation of trust initialisation to a third party and in the other end the total disregard of all third party opinions. For the goals of this work, the existence and effect of reputation systems is acknowledged, but their implementation or actual mechanics is outside of the focus of this study.

Trust Use

How is this now existing trust relationship used? At the top of the trust construct hierarchy there was the trusting behaviour, actions performed based on trust evaluation. So, how do the other trust constructs influence this trusting behaviour and how can these construct used in enterprise collaborative computing?

There can be very many types of actions in communities, for example signing contracts or seeking to form communities, but the focus of this thesis is on allowing or disallowing participants to use our resources. Since the decision process of giving someone a permission to perform an action is called *authorisation*, the concept of *trust-guarded authorisation* is formed. This concept is the translation of trusting behaviour to enterprise setting and means the authorisation decision utilises trust information.

To get this trusting behaviour or trust-guarded authorisation, the underlying trust construct data must be filtered through trusting intention as can be seen from Figure 3.1. The intention decides whether the trustor is willing to trust this role occupant in this community doing this particular action taking account the contractual obligations negotiated before.

This trusting intention specifically depends on the trusting belief, our view on the trustworthiness of another party in this situation. What kind of data is there available in an enterprise system that can be used to support this? The most widely used input factors for the belief formation process in the field of computing were collected and classified previously in Chapter 3.4. From that list the peer identity and the attempted action are crucial in a collaborative community . For example, an enterprise may generally trust someone to make purchases from their web shop but if he or she wants to buy items using a credit card, the enterprise's willingness to depend on the peer in that case may be significantly lower. The formation process gives data that may be used to calculate the trusting belief score for a particular peer in this situation.

System trust is also very important in this phase, both for a community and for the single system. System trust dictates whether we believe that the situation and outlying impersonal structures support success in our actions. From the outside perspective we get the legal and contractual framework, but from the computing systems inside perspective the system trust issues are focused on the trust to the correct functioning of the security frameworks etc.

In the the community we should have system trust to networking and community contracting processes. In the community level trust is also about the trustor's expectation of a certain behaviour within the community, or at least the lack of malicious behaviour. Therefore it is natural that contracts can encourage trust by making implicit expectations explicit [GMT04] and this affects also system trust. There is also another positive effect of contracts, since they usually specify penalties to non-compliance and contract breaches. These penalties can range from the refusal to deal with the trustee in future transactions to monetary sanctions. Thus e-contracts also offer incentives to good behaviour and could be considered a system trust factor. However, in this work system trust is handled as just context for simplicity, but a study of the effects of a dynamic and automated contracting process to system trust and the whole trust calculation could be interesting.

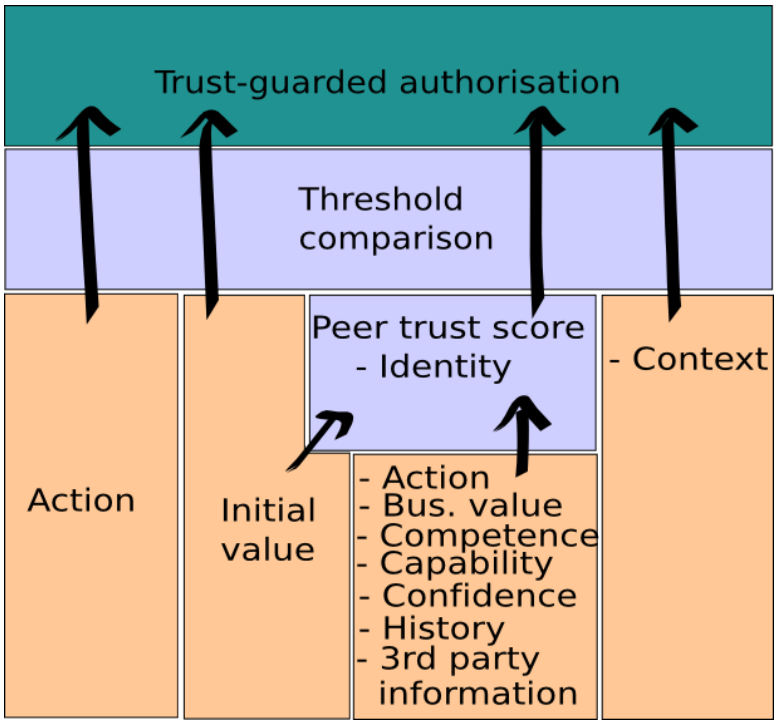

Figure 3.6: Trust constructs in a trust-guarded authorisation scenario.

It should be noted that the attempted action can be used in the trust evaluation from two separate points of view. Trusting belief uses the peer identity and the attempted action together to form an understanding on the trustworthiness of the peer, while situational decision to trust considers only the current situation, attempted action. It is possible to use either or both of these trust constructs depending on the system. For example, if the system has no access to peer identity information situational decision to trust should be used instead. If the peer identity and attempted actions are used in the belief formation process, situational decision to trust can augment the trusting intention calculation. In an enterprise scenario this may be used to give all peers access to some actions such as service listings independent of their general or specific trustworthiness.

A picture, parallelling the Figure 3.1 of McKnight and Chervany, is shown in Figure 3.6. Here the general trust goal, trusting behaviour, is defined as a successful action by the peer. The authorisation for this attempted action needs to be trust-guarded, i.e. depend on whether the enterprise trusts the peer or not at this particular moment for this attempted action. The intention to trust, i.e. authorise the action, is ultimately decided by comparing a trust score to a threshold value. The input values for the trust score comparison are received from trusting belief and dispositional trust. Trusting belief here is a collection of peer specific factors in the used trust model. The dispositional trust in turn is modelled by an initial value. Additionally, since the trusting intention is very situation specific, the comparison is enhanced by the trust input factor of context that describes the enterprise situational normality in system trust.

This is a new general model for trust use in an enterprise computing scenario. It is well grounded in trust theory and independent of the used trust model. This model can be applied to different concrete applications, one such trust-guarded framework, TuBE, is presented later in Chapter 5.

Compared to previous research, this new model makes explicitly clear the differences between

various aspects of trust and enables better structure in the trust management system. For example, in the previous SULTAN system the monitoring component provides not only peer experience information but also system context and risk information [GS03]. Since the SULTAN monitoring mechanism has so widely separate tasks its implementation is not simple. The division of trust management system components according to trust constructs brings two key benefits. First of all, the modular structure enables easier changes in the framework if the implementation and experiments show the structure does not work. Moreover, each component becomes simpler to implement and easier to deploy.

Behaviour Monitoring

When a positive authorisation decision has been done taking to account all trust information, it is prudent to observe the behaviour of the peer in the system or community to detect any misuse, which in turn must cause our trust to be translated to mistrust, i.e. misplaced trust.

As discussed previously in Chapter 2.7 monitoring in communities can be done in various ways and for various reasons. From the community point of view, a neutral third party can use inter-enterprise monitoring to detect any problems in the functioning of the community. However, for the needs of a single community participant, introspective monitoring is also required.

The reasons for this monitoring are twofold. First of all, an enterprise may want to collect experience of peer behaviour to enhance the trust evaluation in the future, both on the community and single system level. Secondly, observation can give warning if any anomalies or misbehaviour occur so that damage to our systems can be prevented or limited. Afterwards, the misbehaving peer can be classified as someone to distrust, i.e. not trust at all.

In introspective monitoring participants monitor their internal state and compare it to the community rules and expectations. With the goal of detecting security violations, this type of observation moves conceptually close to the security realm of intrusion detection systems (IDS). Intrusion detection monitors systems, such as applications or networks, to detect and identify potential security breaches. Application level intrusion detection is discussed in more detail in Chapter 4.

Trust Update

It is not enough to just observe or monitor the collaboration. Mechanisms for reacting to the observations in some way are needed, for example by passing the observation information back to the trust evaluation and updating the trust values. There are several types of reactions: updating the local trust information, modifying the system to correspond to the changed trust valuation and propagating the information to third parties. Dynamic trust re-evaluation has not been very common in recent trust management systems [GS02].

The key trust concept here is updating the trusting belief; how much we believe a peer is trustworthy in a given situation. People have an underlying belief formation process that affect the value and update of this trust construct, systems can use past experiences gathered by an observation component to adjust this belief. This is a belief of a future behaviour formed by past experiences of a peer in a particular situation.

The update or feedback from the monitoring component to the trust valuation is either positive, i.e. everything went as expected and our trusting belief is increased, or negative, i.e. there were problems in the collaboration that cause us to trust the peer less [2].

In the extreme case, updating a trust relationship can also mean severing the existing trust relationship. In our view, a severed relationship is still a trust relationship, however its meaning is “do

[2]The “no change” case is ignored as uninteresting.

not trust". Therefore complete distrust is not handled any different from other trust relationships in this work and trust management thus includes also distrust management.

An update to the trust score of a peer causes the next evaluation in the trust-guarded interaction for that peer to use that updated value. However, that evaluation can be a long time in the future depending on the granularity of the trust guards. If, for example the trust evaluation is done only at peer authentication time, the current activity causing a negative trust update can progress uninterrupted and thus cause the enterprise harm. Therefore a good trust-guarded system should also have a mechanism for modifying the system via an immediate functional reaction, such as blocking all network communication or rollback of all actions in this session if the trust valuation drops too low. This immediate functional reaction feature is missing from the SULTAN system monitor, since it only adds state information to the trust state database or sends an alert to the administrator GUI tool [GS03].

The update to the observer's internal trust value storage can also be propagated to the whole community as reputation or a recommendation. It could also be propagated to parties external to the community where the actions take place, but the common semantics of that reputation or recommendation should be agreed beforehand by creating a reputation or recommendation sharing community. Reputation and recommendation mechanisms and their intricacies are outside of the scope of this work. Reputation and recommendations are useful especially when trying to populate the community roles, but from the TuBE perspective the key update issue is to get trust updates from the monitoring function for the local trust decisions.

3.6 Management System Evaluation Criteria

Previously enterprise communities, trust and finally trust management have been introduced and discussed. From the reviewed research it can be seen that many separate systems have been developed in these areas, but very few bind all the components together.

Therefore it is the aim of this work to develop a trust management system which fulfils the requirements below. The first in the list is a conceptual requirement that ties trust management in computing systems to the theoretical foundations of trust type categorisation, the next four are general trust management requirements [BFL96], and the last two add the support for the new fourfold trust management model.

1. Conceptually, it automates the translation of trusting intention to trusting behaviour.
2. It presents a unified approach to trust management across applications.
3. There should be locality of control in the authorisation decisions.
4. It is flexible in supporting complex trust relationships and multiple decision input factors that influence the trusting belief to a particular trustee.
5. The mechanisms for evaluating trust should be separated from the policies thus enabling reuse of the mechanisms across applications requiring different policies.
6. It should introspectively monitor the peer behaviour to detect ongoing mistrust situations.
7. It must produce behavioural feedback to the trusting belief calculations to prevent further mistrust.

A platform or framework fulfilling these requirements has several benefits over current state of the art authorisation systems. First of all, it is well founded by trust theory. It supports dynamic trust evaluation thus removing the need to manually configure each and every static trust relationship for each application. Secondly, it is capable of utilising a multitude of trust models and is not strictly bound to any single one. This is a benefit from the flexibility point of view. In a community setting there may be situations where different trust support facts are used. For example in a community population scenario peer competence may be much more useful support fact than any business value factor. Thirdly, it is aware of the misuse possibilities such dynamic trust relationships bring and has ways of detecting and managing the mistrust events. Potential mechanisms for obtaining this third benefit are explored next.

Chapter 4

Behaviour Monitoring

Enterprises today rely increasingly on the Internet and the partnering and data exchange opportunities the network connectivity brings. Collaborative communities are a new and emerging trend of this network based business. However, the current Internet and networking environment is full of threats, both external and internal. Physical security around the organisation premises are not enough since network based automated attacks and network-borne viruses have caused organisations and their published services or applications to be potential attack targets every minute of every day. Solutions to this problems today are various restrictions. First of all, the enterprise can restrict the application use to a set of well known partners. Second, they can restrict the network connectivity to these partner organisations to limits their exposure. However, neither of these solutions do not remove the general exposure threat, since a simple misjudgement from the partner organisation personnel may cause a security incident that has ramifications for all other connected partners as well. Besides, these restrictions must be actively managed and in a large enterprise, such as a grocery shop chain, the list of suppliers and other partners can be huge. Thus restriction based security incurs a heavy management overhead.

Additionally, applications are nowadays so complex that practically every piece of software has some amount of faults, or bugs. Some of the bugs are security related, i.e. their existence creates a security vulnerability that can be exploited to create a real security problem or incident.

When these two facts are combined, it can be concluded that tools and mechanisms helping enterprises manage the threats collaboration brings are needed. It is especially important to create solutions that have a very light management overhead to enable dynamic partnering and collaboration. Therefore it is useful to explore ideas that help enterprises manage the exposure threat not with static restrictions but using dynamically changing trust. This implies the trust management system must detect the actions of potentially malicious people and automated applications. In the language of trust explored earlier in Chapter 3 there must exist mechanisms for managing mistrust, untrust and distrust.

Mistrust management is relevant in cases where there is a set of trusted and thus situationally authorised partners, but who may try to defraud us or exploit our service. Untrust must be managed in cases where there is some, but not enough trust to a connecting peer. In distrust management we have a negative trust opinion on the peer or not enough information to form a trust opinion, and must guard against also their malicious actions. Peers may try to exploit known and unknown software faults, system features or existing trust relationships to gain access or privileges against the published or implicit system security policy. This risk must be managed with an introspective monitoring system. Traditionally in the Internet environment the solution has been an intrusion detection system (IDS). In this chapter we take a brief survey what kind of intrusion detection systems there are and how they may help us manage all these variants of trust problems especially

in a collaborative enterprise setting.

4.1 Intrusion Detection Concepts

An intrusion detection system (IDS) tries to detect when a malicious attempt against a system which may lead to an intrusion is made. It usually consists of a data collection *agent* that collects information about the system being observed. The collection agents are often also called *sensors*. This collected data is then either stored or processed directly by the detector proper. The *detector* analyses the data and presents conclusions to the responsible person in some usable format. In some cases the detector can also initiate countermeasures to block the detected intrusion.

Intrusion detection systems can be classified in many ways into several categories depending on their characteristics. One way of doing this division is according to the data collection method and the data analysis mechanism. Furthermore, this thesis add a new category to the classification, namely the reaction mechanism, since it is extremely relevant to any system trying to manage trust relationships, not just detect intrusion attempts.

4.1.1 Data Collection Model

The following aspects can be identified in the data collection models of various intrusion detection systems:

- Where do the agents or sensors reside? Classically they have been either host based agents or network based agents. Systems with host based agents are called host based intrusion detection systems (HIDS) and systems with network based agents network intrusion detection systems, (NIDS).
- Is the collecting centralised or distributed, i.e. is the data collected from a single point, such as from a centralised log file (for example `syslog`) or from various places, such as different network segments.
- Is the monitoring direct or indirect? Direct monitoring means that the IDS can directly observe the target, including its internal state. Indirect monitoring relies on target external interfaces (such as the log files) for information.

Network based IDS (NIDS) sensor a listening device in the network, where it typically captures all network traffic and tries to detect protocol packets with malicious contents. It has a set of usually lexical rules detailing the definitions of malicious traffic. Examples of malicious traffic are possibly port scanning, too large ICMP packets (i.e ping-of-death) and known web server CGI exploits. Network based IDS also has directives what to do when such activity is detected.

Host based IDS (HIDS) works within a computer server and tries to detect misuse or anomalies that occur in the server but are not necessarily network based usually by examining various log files and system level audit trails. Examples of such problems are brute-force password guessing, changing key system configuration files and accessing protected files.

The data collection can be done centralised or distributed. Centralised collection reads the information from a single point, such as central log file or from main network switch or the internet firewall. However, this approach has both scaling and coverage problems, therefore most current systems preferably deploy distributed collection, i.e. there is a sensor in nearly every host and network segment.

The monitoring can be done indirectly or directly. Indirect monitoring is defined as "The observation of the monitored component through a separate mechanism or tool" while direct monitoring can access the component and its internal state directly [Zam01]. Examples of indirect monitoring are application log file reading, while direct monitoring is performed within an application.

4.1.2 Data Analysis Model

Whether one deploys a HIDS or NIDS system, centralised or distributed collection, an analysis system is required to provide a sensible summary on the observed activities. One feature of the analysis model is the location of the analysis, whether the analysis is done in many places simultaneously) or centralised.

But the most differentiating feature of this is the analysis philosophy; misuse detection, anomaly detection or specification based analysis. This has been one of the basic factors according to which IDSes have been usually classified.

Misuse detection relies on patterns (usually called signatures) of known attacks and effective handling of large amounts of network traffic or log files from which to detect these patterns. While the signatures are attack or attack-class specific, and thus their applicability to unforeseen situations is limited, some new attack types can be detected if they fall in the similar class than some previous attacks and the signature has been written generically enough.

Anomaly detection tries to differentiate between anomalous and normal behaviour of the system and alert on all unexpected behaviour since it may be an attack. The problem of this method lies on the question how to effectively define system normal behaviour, how to compare the current situation to this normalised ideal and how to minimise the false positive rate (i.e. normal behaviour flagged as intrusions) while keeping the false negative rate (i.e. missed intrusions) close to zero. On the positive side, this method is not tied to any specific attack or attack type and can potentially alert to new and previously unseen attacks. This technique is especially valuable in new situations where attacks and attack types are not known.

Specification based analysis approaches the problem by more or less formally by defining the correct behaviour of the program [KRL97] or network protocol [SGF+02] and reacting to any deviations. The key problem in this approach is the initial generation of such a formal model, which needs to be done for each security critical system.

Each of these philosophies have different techniques suited to the task such as statistical analysis, syntactical pattern matching or expert systems. The basic detection principles have been very well classified by Axelsson in his survey of intrusion detection systems [Axe00].

4.1.3 Reaction Models

One newer aspect in the IDS classification is the IDS reaction model, i.e. what the system does when it finds out a potential problem. The reaction model can be immediate or delayed. Depending on the organisation and its capabilities sometimes it is quite enough to know a penetration has occurred some time in the past, while in some cases it is quite imperative to be able to instantly do something to the suspicious activity.

Traditionally IDS components have been interested in just generating and sending alarms immediately so that people can deal with the intrusion. This is a sensible approach in situations where people are available 24x7x365. But in some organisations it can take hours or even days for a system administrator to be able to react to the alarm. Therefore, some organisations would actually be better off with a delayed reaction IDS.

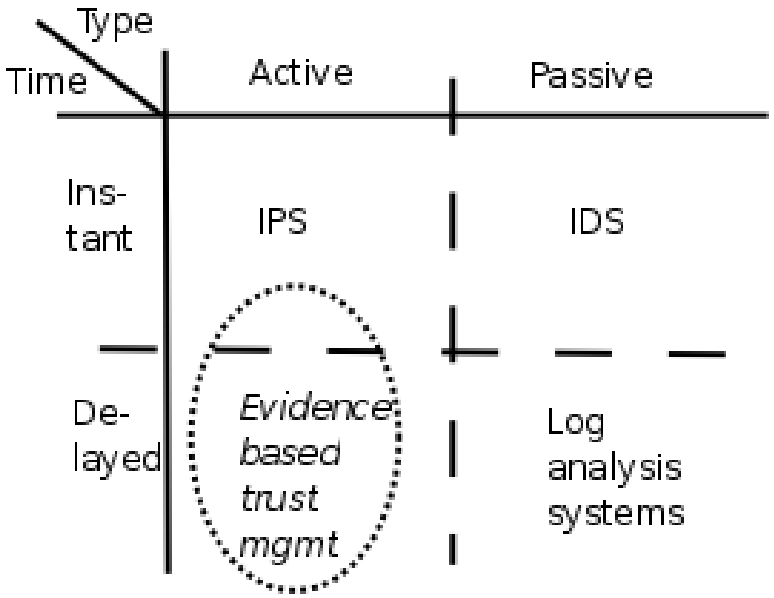

Figure 4.1: The IDS reaction matrix

The delayed reaction has given rise to various expert systems that analyse massive amounts of network traffic and system logs with for example data-mining [LS98] or expert system techniques [LSC97]. These enables them to create better overviews and search for larger patterns in the attacks. This can be done for example during the night and when people show up to work in the morning, they are given a report on the night's activities. Various types of delayed analysis systems have also been utilised in commercial products.

Another aspect is the mechanism of the reaction, active vs. passive. Passive reactions, such as alarms, do not affect the observed system in any way. Active reactions, such as closing the attacking network connection, actively protect the system, thus gaining the name Intrusion Protection Systems or Intrusion Prevention Systems (IPS).

The different variants are illustrated in a simple matrix form in Figure 4.1. The lower left hand quadrant of the matrix, active but delayed measures, is currently somewhat open. Into this quadrant would fit measures that for example adjust peer authorisation, reputation or trust values after a security incident or any kind of deviation from the norm has occurred. This is somewhat different from immediate reactions, since the effect of these measures is usually visible some time after the incident has occurred. For example simple trust score adjustments take effect when the peer contacts the service the next time. Thus evidence based trust management is placed mainly in this quadrant with potential to reach into the active and immediate quadrant depending on the actual mechanism of the reaction.

4.2 Problems in the Current IDS Approaches

Consider a scenario where a local workstation user is sending all outgoing e-mail to a company main mail server, which then re-sends these e-mails to the real destination. This is a common setup in organisations where the outgoing connections of a simple workstation are restricted.

What happens if the workstation is infected with a virus, worm or trojan that is either propagating by sending out a mass mailing of itself or sending out some other undesired content (i.e

spam) to all addresses in the address book. These e-mail borne viruses have been very common in recent years and the trojan horse programs installed by the viruses have been used to send spam.

Note that the infection can occur even if an antivirus program is deployed at the workstation. The time between the detection of a new virus and the saturation point of a network enabled virus is diminishing so fast that the antivirus companies have a hard time developing and especially globally distributing the recognition signatures and countermeasures in time. The same is true for network IDS systems that use pattern matching so that the signature patterns need to be distributed.

So, there is an infected machine on the network that starts spewing out massive amounts of e-mail (either with viral or spam payload). This is not detected by the NIDS, since sending e-mail is by its rules quite acceptable, nor is it detected by the HIDS at the mail server, since it also considers receiving e-mail from a trusted host in the network quite normal. So the mail server happily accepts and starts redistributing the harmful content around the world.

Another scenario could be an extranet server that is offering some application via a defined Web Services interface. It probably has some form of access control and authorisation to determine which systems are allowed to connect and what various contacting peers can do. But what happens if the peer system is penetrated by a malicious hacker? The peer penetration may not be at all visible to the extranet server, since the authentication and the authorisation information is usually stored at the server and possibly even unencrypted on the system disk. Thus the attacker can send syntactically correct but semantically malicious information to the extranet server undetected by access control and authorisation mechanisms, NIDS or HIDS.

Some additional detection approaches are required if the damage of this type of situations is to be kept to a minimum. Especially some level of application awareness is required of the intrusion detection systems.

Application level intrusion detection (AppIDS) detects intrusions by collecting information about the monitored application system, analysing this information by evaluating relations, and taking some action if a relation result is judged to be anomalous [Sie99].

The difference between an application IDS and a host based IDS is the fact that AppIDS is application specific; it is able to work with one application or application class only. HIDS usually detects problems within one host system, possibly running several applications. HIDS works especially on operating system level but it can include application specific components as well. Similarly, the difference between AppIDS and NIDS is the fact that NIDS uses only network based collection methods. While a NIDS can also include application specific rules, it is concentrating on the network traffic in general, not one specific application.

Defined like this, application IDS is not a new idea. Research on various forms of application execution monitoring has been done in the 1990's [FHSL96]. The name *application intrusion detection* was coined later by Sielken [Sie99] and its benefits were enumerated more closely by Zamboni [Zam01]. The application level intrusion detection has several good characteristics over NIDS and HIDS that make it a desirable enhancement to the IDS palette:

- Depth of defence. This is a basic tenet in the security field, if one system (such as a firewall) develops a problem, there must be other mechanisms in place to detect and protect against intrusions.

- The application awareness in NIDS and HIDS can be restricted. This means that the network and host based systems do not necessarily need to be enhanced to recognise application level problems, this can be delegated to an application level system.

- It can technically monitor applications directly, i.e. there can be sensors observing the application internal state directly. However, all AppIDS methods do not utilise this possibility.

- The application can see data which a network or even host IDS sensor can not. The key example is traffic that is encrypted at the application level (for example using SSH, SSL). This NIDS blindness to encrypted traffic is a concern in some organisations. Terminating the encrypted connection in the perimeter may not be an option for all environments.

- The IDS reaction can be more fine-tuned according to the service capabilities. Depending on the intrusion type the enterprise might not want to break the connection altogether, but for example restrict the connection or data rate (i.e. tar-pitting). Other application type countermeasures can be changing the source authorisation parameters (i.e what actions are permitted and what data is available).

There are also problems at the semantic level. Most intrusions today exploit known application vulnerabilities, which are in many cases weaknesses in mechanical input handling. Unchecked buffer bounds, misplaced nulls and format string problems etc are syntactical in their nature, i.e. the received value differs from the expected value in length, type or allowed characters. But current systems are not interested in the actual semantic sensibility of the received input, i.e. if the peer suddenly places an order for 10 000 000 flight tickets. While inputting this should not cause immediate access to the operating system level (unless there is a buffer overflow and attached exploit code) or denial of service attack, it may still be a cause of concern because of fraud possibilities. Application level mechanisms can potentially analyse this type of problems.

The survey below in this chapter has been made to canvas the area of application intrusion detection methods to pinpoint possible technical methods and tools for managing mistrust, distrust and untrust at the application level.

4.3 Data Collection Methods in Application Level Intrusion Detection

Before it can actually be analysed whether an intrusion or misuse attempt has actually happened in an application, there must be access to the application behaviour at a suitable level. This chapter discusses various options for monitoring the application and collecting the application behaviour data for analysis.

However, not all reviewed work in the application IDS research domain were explicit in their data collection methods (for example [JL01]) and some were just developing new analysis methods to old collection mechanisms (for example [WD01, GJM02, WFP99]). The works detailing new analysis methods only are discussed later in Chapter 4.4.

The reviewed research papers were generally not discussing the distributed vs. centralised aspect of their collection method. However, since application level data collection can add information to the overall intrusion detection system, these methods can be seen as extra sensors providing depth of defence in a larger integrated intrusion detection system.

4.3.1 Host Based Monitoring

There are several mechanisms within a host, typically at the operating system level, that can be used to monitor an application. Some of these could be seen as host-based intrusion detection data collection but since they have insight to the application behaviour, they have been included in this survey.

Process Auditing Mechanisms

The process auditing subsystem of the operating system is of use to application level intrusion detection especially when it comes to extracting system call or other process behaviour data for execution monitoring analysis. Process auditing subsystems do not have insight into the application internal behaviour, but are able to monitor how the application acts towards the operating system.

Early work used auditing subsystems to extract security events from the operating system logs. For example VAX/VMS had a very advanced auditing facility that has been used for user activity tracing in an intrusion detection setting [TCL90]. In this particular case the audit events analysed were user commands, so this is not really in the application intrusion detection domain.

A more widely used system is Solaris SunSHIELD Basic Security Module (BSM), which is a part of the Sun Microsystems' Solaris operating system. It provides the security features defined as C2 in the Trusted Computer System Evaluation Criteria (TCSEC). The features provided by the BSM are the security auditing subsystem and a device allocation mechanism that provides the required object reuse characteristics for removable or assignable devices [Sun94]. BSM has been used in several AppIDS systems to be the source of system call information [KFL94, KRL97, End98, GSS99, GMS00, MG02].

Wespi et al also used an unnamed audit subsystem to collect audit events [WDD00]. The received audit records were not system calls, but on a more abstract level where the basic operation information was available (such as "file open"), but not the particular parameters.

Other operating system tools have also been used. Some Unix systems contain the user-level program `strace`, which will run a program given as an argument and list all the system calls it makes. Other such Unix programs also exist, such as `truss`. The `strace` program has been used to gather information in several application IDS systems [FHSL96, KH97, Mar00]. Additionally, the data analysed by Lee et al ([LS98],[LSC97]) had been obtained with `strace`.

Since these user level programs exist, the Unix kernel offers primitives for system call trace data collection. Some research also used these underlying operating system constructs directly from user level for better performance [SBDB01], [FKF+03].

Kernel Mechanisms

The process auditing mechanisms above used operating system services for reporting system call or audit data but worked at operating system user space. However, this reporting function can also reside at kernel level. For example, Sekar et al used a kernel level module to intercept system calls to support policy enforcement. Also the analysis mechanism resided in the kernel space [SBS99].

A kernel-level approach has also been used in behaviour data collection from software execution environments. In the work by Ko et al, the execution environment i.e. a software wrapper, functions in kernel space [KFBK00].

One application level mechanism utilising kernel level possibilities is the waypoint technique. Waypoints are kernel-supported trustworthy control flow checkpoints in the code. Waypoints can actively report program control flow information in real time. Also security attributes can be attached to each waypoint or to a sequence of waypoints. Waypoints are set at each application function entrance and exit, also middle waypoints that reside in the function can be created. The waypoints are generated via static analysis [XDC04].

From the data collection point of view, waypoints have one key benefit over plain system call collection. Waypoints can give the intrusion detection system a better granularity than traditional system call based solution by the ability to create more waypoints than system calls [XDC04].

The key problem with this type of approach is the extensive kernel support required. The waypoint structures and policy enforcement systems must be located in the kernel to be secure from manipulation. While they are more secure from the system users, it is also riskier to augment stock kernels with extra modules. From the kernel point of view, any security or performance problem in the added IDS functionality may be reflected to the whole system. This analysis applies to monolithic kernel structure, exhibited in systems like Linux. The situation is different in micro-kernel systems, such as Windows NT and derivatives, where only minimal message passing and scheduling systems are present in the kernel proper. Other functionality is running in user space or between the outer user ring and inner kernel ring. The effects of embedding IDS functionality into a micro-kernel depend on the positioning of the intrusion detection modules in the micro-kernel protection system. This has not been a very well researched area.

Kernel-level approaches have also other problems. First of all, they are kernel specific and thus not portable. Secondly, the kernel systems can also cause performance interference, i.e. generic overhead to operations that are not in need of an IDS mechanism. The system management is a problem as well, since reconfiguring a kernel-space system may require a system reboot. Also the kernel services have a very low level of abstraction, which causes the analysis phase having also low level of abstraction and deduction.

Library and System Call Interposition

One possibility of collecting information about application calls is library or system call interposition, which is the process of inserting a modified call between the application and the real target of the call.

The interposition can be done at library level, for example in Linux and Solaris environments the LD_PRELOAD environment variable is able to insert libraries that are searched before original libraries. This technique can be used to track the library calls an application makes.

The system built by Kuperman and Spafford used this library interposition to detect unsafe Unix programming practises, such as overruns of static buffers in `sprintf` calls. The penalty of interposition was about 3% for non-interposed calls and about 57% for interposed and logged system calls. Most of that overhead was due to the logging mechanism [KS99].

Call interposition can also be used at kernel-level. Bowen et al used a loadable Linux kernel module to insert an interposition layer into the kernel and rewrote the kernel system call table to execute first an interceptor and then the call. If any function besides interception is required, it can be specified in a module compiled from a behavioural monitoring specification language (BMSL) description [BCS+00].

The kernel-level method described above is able to track the application behaviour towards the kernel only, whereas library interposition is able to collect data on the library call level. Static analysis can achieve similar goals to library interposition, but it can not give us information on what happens in an application at run-time.

4.3.2 Network Based Monitoring

Some application IDS systems do use network based data collection, since they use the data available from the network for intrusion analysis. This assumes that the application is distributed in a way that network data is meaningful for analysis. This approach is not discussed in detail, since this is more in the domain of network based IDS.

Some interesting research exists, though. For example Raz et al captured data in a dynamic data feed for semantic anomaly analysis [RKS02]. Also CORBA remote calls have been inter-

cepted so that the remote call execution patterns could be analysed similarly to system call patterns [SMS99]. These approaches show that network traffic interception can be utilised in the application layer as well.

4.3.3 Direct Monitoring

The most straightforward way of collecting data for misuse or anomaly analysis in an application is to write detection and collection hooks into the application code itself. This is direct monitoring, i.e. the method can access the component and its internal state directly.

Diego Zamboni did code internal misuse detection in his research by modifying existing software to detect well known attacks against it. The solution described added a series of detection code agents to the `sendmail` application code against attacks collected from Common Vulnerabilities and Exposures (CVE) database [Zam01]. He also added some generic agents, for example against port scanning. Different techniques, such as character matching heuristics or stateful/stateless detection, were used in the agent implementation.

Similar approach was also used by Sielken, who first did a risk analysis of potential application fraud points and then wrote additional code to detect anomalies in client behaviour [Sie99].

Elbaum and Munson instrumented an application, a Linux kernel in their case, with code hooks that provide information on how the application modules call each other [EM99]. This module profile information was then used for anomaly detection.

The hard part of the code-internal method in general is the extensive programming and programming related activities it requires. While the actual number of code lines generated is not significant, modifying an existing system means in practise that a significant effort is spent in understanding the code before being able to make meaningful modifications [Zam01].

The actual agent or sensor can also be in a separate module, which has been linked into an application via a suitable interface [AL01]. This method enhances the flexibility of the agent implementation and diminishes the time needed to understand the code base. However, the used interface limits the agent's view of the application and also places restrictions on how the agent can affect the application. Therefore, the degree of directness in the agent model is also diminished depending on the interface characteristics.

4.3.4 Monitoring the Execution Environment

A very distinctive class of IDS data collection methods includes systems where the application execution happens in whole or in part in a pre-configured container or other safe execution environment. The executed application is monitored by this container and the container is able to collect behaviour data and potentially also analyse unexpected behaviour. There are several variants of this basic idea.

OS-level Environments

There are various methods of restricting the application visible environment for security reasons. One of the most successful is Unix kernel `chroot` system call and the associated control program, where a subdirectory of the file system hierarchy is created to resemble the full Unix file system, and the application execution is restricted to this subdirectory and the services and components visible from there. However, `chroot` can not be considered a real application IDS, since it just prevents applications accessing the real OS file system and it has not been designed to alert on any undesired activity. There are other such execution restriction systems as well, such as the BSD

`jail` and Linux AppArmor but since they are somewhat off-focus this thesis does not explore them in more detail.

Since no widely used operating system offers a ready made system focused on application level intrusion detection, there are ways to augment the systems with monitored execution environments or application security wrappers. One such effort is the Generic Software Wrapper system, which is a loadable Unix kernel level module (LKM) that associates wrapper instances with application processes [FBF99]. Generic software wrappers have also been used by Ko et al in their work [KFBK00].

Wrappers are defined in the Wrapper Definition Language (WDL) that masks the operating system details from the wrappers and also defines the functionality the wrappers have. One key point of this system is the fact that the WDL characterisation of the low level kernel API contains some semantic information on the system calls, for example a tag labelling a call parameter or return value as a file descriptor. This enables both some level of OS independence and also simple semantic reasoning in the wrapper action rules.

Wrappers intercept some or all system calls the applications make and take action based on the WDL definition. The wrapper can either deny the system call or augment or transform the call. Augmenting means adding functionality to the call, such as encrypting or performing intrusion detection analysis on the call data. Transforming, on the other hand, is replacing the call with one or more other system calls or other events [FBF99].

Another example of this kind is *Systrace*, an environment which supports fine grained process confinement, intrusion detection, auditing and privilege elevation [Pro03]. There the system calls are recorded by a kernel-level mechanism and the system call is checked against the recorded normal policy in a user space policy daemon. The approach takes also the system call arguments into account. *Systrace* is currently available for Linux, Mac OS X, NetBSD, and OpenBSD.

One positive point of this OS level approach is the fact that it works is practically application independent. It is also easy to deploy even for older or third-party applications. However, since it is an OS level tool, it does not necessarily have any deep insight into the actual workings of the application. Therefore, these systems are doing indirect monitoring. A suitable system may not be available to the preferred operating system either.

Dynamic Execution Environments

Dynamic execution environments (DEE) are systems where compilation, profiling and optimisation are dynamic, such as Sun Java and Microsoft .NET Common Language Infrastructure [IF02]. Dynamic execution environments are a subgroup of virtual execution environments (VEE), more often called virtual machines.

As an example of a dynamic execution environment, programs written in the Java programming language are run in Java Virtual Machine (Java VM, JVM). Rather than running directly on the native operating system, the program is interpreted by the JVM and the resulting bytecode is run within the JVM on top of the native operating system. The JVM restricts the application by creating a sandbox, where a security policy dictates what the application is allowed to do.

One IDS technique utilising the Java virtual machine is dynamic sandboxing. It consists of two activities: sandbox generation and sandbox execution. In the first, a dynamic sandbox profile is constructed by running the program with an instrumented JVM and all observed method calls are written to the profile. Later, in the execution phase proper, observed method calls are compared to this profile and anomalies cause an error [IF02].

The difference between JVM's own sandboxing and this dynamic sandboxing is similar to the difference between Unix `chroot` jail and a real IDS. Both the Java sandbox and the Unix

`chroot` are able to just deny the activity outside their limits, but dynamic sandboxing can be classified as a real intrusion detection system, since it is able to detect malicious activity.

There are also systems where only the intrusion detection data collection agent is built into a JVM [SKV03]. To monitor any suspicious activity performed by applications running in the JVM (JikesRVM in this case) the virtual machine was extended with an event logging system. The auditing facility consists of an event driver, an event queue, and an event logger. The event driver adds thread-level execution events to the event queue. The logger processes events that are contained in the queue and writes them to an external log . This external log can then be processed by a suitable intrusion data analysis system (STAT in this case). In the analysis model this system is more related to host based intrusion detection systems than other application level systems.

Component Level Wrappers

Component based application development has rapidly become popular in recent years. There are sound reasons for it, especially regarding cutting development costs with code reuse and isolating faults with modularisation.

Application wrapping for intrusion detection can also be done on the application component level instead of the level of application as a whole. Herrmann and Krumm created an architecture where an adapter is generated for each Java Bean and the bean is wrapped by the adapter. An adapter generator introspects the bean for its interface actions and creates the related adapter based on this examination. This approach means that each component is protected by the adapter and all component interaction via the component interface goes through the adapter so that the interaction can be observed [HK01]. This is direct monitoring in the sense that the application interaction between the beans can be observed, but on the other hand this method does not have visibility in the inner behaviour of the beans.

4.4 Analysis Methods in Application Level Intrusion Detection

Whether one deploys a HIDS or NIDS system, or detects problems on application level, an analysis system is required to provide a sensible summary on the observed activities. This analysis can be done according to three distinct philosophies: misuse detection, anomaly detection or specification based analysis.

Since application level IDS is a relatively new concept and there are several data collection methods described above, there are also several algorithms and methods for analysing the collected data. The application data analysis methods have been classified into the three main philosophies with the difference that in this document, survey of the anomaly detection techniques has been divided into two sub-chapters: anomaly detection in execution monitoring and anomaly detection in input semantics. This division is purely a device for managing the readability and structure of this document, conceptually these two are in the same anomaly detection domain.

4.4.1 Misuse Detection

Misuse detection has not been a very widely researched topic in the application IDS arena. The methodology map is shown in Figure 4.2.

A trivial approach to misuse detection is to make modifications in the code to detect known vulnerabilities [Zam01]. More refined method is to use known attacks to train a neural network, a multi-level perceptron (MLP), to recognise them and also similar attacks from a stream of audit

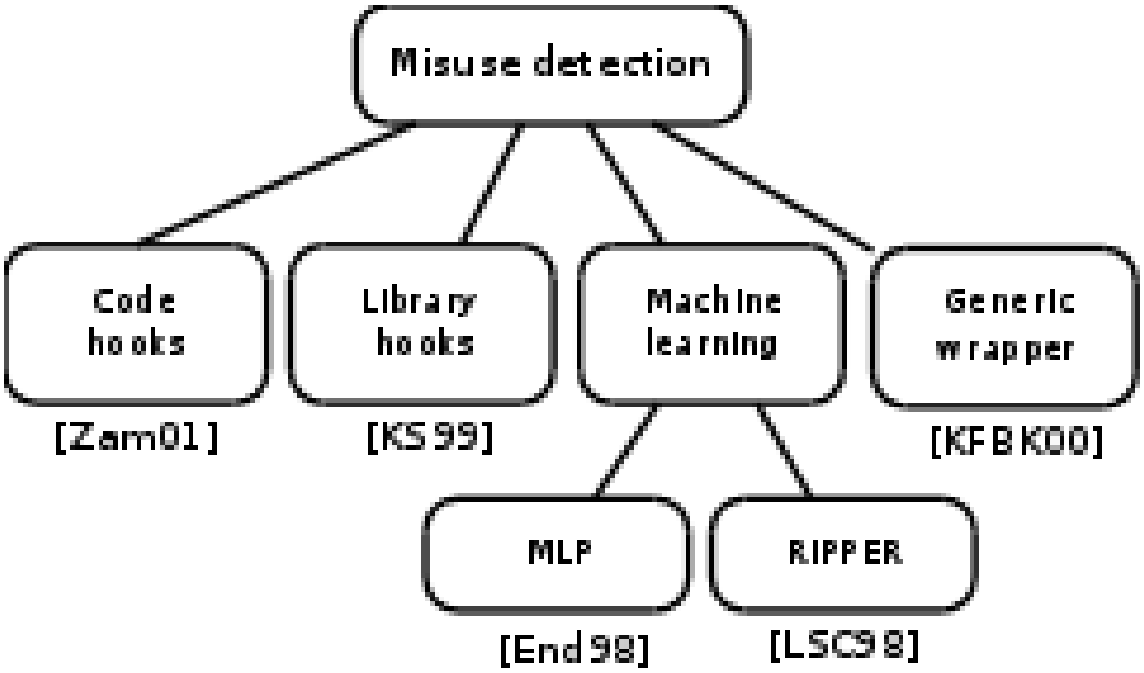

Figure 4.2: The misuse analysis methodology map

events [End98]. A software wrapper technique can also be used for misuse detection if the attack signatures are encoded into the wrapper data structures [KFBK00].

Kuperman and Spafford used library interposition to detect unsafe Unix programming practises, such as unbounded data insertion into static buffers, race conditions and unsafe system calls (such as `system()` and `execvp()`) [KS99]. Since this technique searches for vulnerabilities facilitated by the used platform and tools instead of the application characteristics, it is application independent.

This lack of research in the misuse domain is not very surprising, since misuse detection uses known vulnerabilities either for searching or training. This means each new discovered vulnerability must be encoded in the application intrusion system. If the detection method is source code based, as in the trivial case, this would mean recompiling and reinstalling the application every time a new type of vulnerability is found. Also wrapper based techniques suffer from this update problem, although to a lesser extent, since the update must be done on the wrapper side, the application can remain as it is.

Machine learning could make life with new vulnerabilities much easier, if we could be sure that a new attack would be covered by the learning generated by the old training data. At minimum, an IDS system employing machine learning would need to be tested every time a new vulnerability is observed, and if the old training does not cover the new problem, retraining must be done.

4.4.2 Anomaly Detection in Execution Monitoring

One key form of application execution monitoring is application call analysis. Here, the word *call* means both system calls and application internal procedure or method calls. Also "calls" to software modules are considered. While the techniques of extracting information for different call

Figure 4.3: The anomaly analysis methodology map

types are different, algorithms for analysing this type of information are similar. Thus these types are not separated here but discussed jointly. The analysis method map for anomaly detection is shown in Figure 4.3. The picture also includes research done in the area of anomaly detection in input semantics, which is discussed in more detail in Chapter 4.4.3.

System calls seem to be at a suitable granularity level for detecting malicious behaviour in applications. This suitability is based on the assumption that the exploit must interact with the operating system to create a real intrusion. While this is true in the case of buffer overflows and shell code exploits, it leaves problems caused by the application data semantics unexplored.

The seminal paper describing this idea came from Forrest et al in 1996 [FHSL96],[FHS97], and it has generated a multitude of different techniques and variants for either obtaining or analysing the system call information.

Their basic idea is to parallel computer defence with human immunology. This analogy needs the definition of system "self", which by their original idea is obtained from the program system call profile. This idea of self can be used for anomaly detection by collecting traces of system calls from programs in a secure training phase and creating a normal profile of how the process runs. This normal database is then used to monitor the running processes and flag anomalies when system calls occur in abnormal positions.

This overall method is called *stide*, sequence time-delay embedding, and it contains four parts: selecting the feature set (system calls in this case), collecting normal data, extracting features from the data and detecting anomalies by comparing the features of current situation to the normal data.

There are two general problems in the system-call based anomaly detection; mimicry attacks and impossible paths [WD01].

Mimicry attack is a form of attack where the exploit is coded in a way that it resembles the normal run of application as much as possible, for example using only normally occurring system calls. However, not all attacks can be coded this way. This attack method would require extensive knowledge on what is normal for this application and also what kind of application model or detection threshold target is using [KH97], [WD01]. There are also two types of mimicry attacks: local and global. A global mimicry attack combines the legal system calls of multiple functions, while a local mimicry attack uses the legal system calls of only the running function [XDC04].

The impossible path problem manifests in techniques where the model includes all possible control paths, i.e. it has also those that no valid run-time behaviour uses. An example of this type is the return-into-others impossible path, where a function is called from location A, but the control is returned to point B [XDC04]. The main concern with impossible paths is that the model can classify such an occurring impossible path as normal and thus miss an attack. Not all system call methods exhibit this problem, typical examples of this problem are in the state transition research [WD01],[SBDB01].

Sequence Similarity Techniques

The original research by Forrest et al used simple sequence enumeration in data collection and feature extraction from the system calls. The system call sequences were of fixed size, *N*-grams, and they used a technique called look-ahead pairs for determining mismatches. The actual measure of anomaly in the original research was a *mismatch count* as a percentage of total number of possible mismatches in the look-ahead sequence. When tested with real exploits against `sendmail` application, the mismatch counts were between 0.3-5.3%. The described algorithm can be implemented in *O(N)* time, where *N* is the system call sequence length [FHSL96].

The key finding was that this was a good method of differentiating between processes and for detecting problems where an unusual system call sequence in an attack is used. However, this can

not be used to detect all exploits, for example race conditions or resource stealing.

An observation made by Kosoresow et al around the same time was that all the anomalies observed exhibited behaviour where the system call mismatches between normal and intrusion traces are executed in bursts, i.e. they were temporally close. This allows for an anomaly detection technique where just the number of mismatched system calls within a certain system call sequence length is used as a detection rule [KH97].

This type of a simple numeric measure for an anomaly gives rise to a new form of a mimicry attack. In this evasion method the idea would be not to clump the exploit system calls together, but to pad the exploit code with normal sequences of system calls, thus fooling the detection threshold. As with basic mimicry attacks, this would require knowing specific details of the IDS analysis method, in this case the detection threshold.

Forrest, Hofmeyr et al refined their work later. The key modification was in the definition of an anomaly, i.e. sequence similarity. They noted that since anomalies occur temporally together, they can used the idea of *Hamming distance* for defining how much a sequence differs from the normal sequence. This method is more complex to computer, however. To detect an intrusion, at least one of the sequences generated by the intrusion must be classified as anomalous. They measure the strength of the anomaly by computing minimal Hamming distance and it was assumed that the higher the minimal Hamming distance, the more likely it is that the sequence was actually generated by an intrusion. Therefore the *signal of the anomaly* is the maximum value of the minimal Hamming distances in all the sequences. To make this measure applicable to variable length sequences, this signal value was used normalised over the sequence length value. This normalised value was used as the preferred measure of anomaly [HFS98]. So, if the normalised anomaly signal value is 0.5, it means that at some point half of the trace has been different from the original.

In the more detailed tests, it was found that after a certain length (6), the length of the sequences had very little effect in detection as measured by the normalised signal of the anomaly, so they used a fixed length of 10. In real exploit tests, the normalised signal of an anomaly was between 0.2-0.7 and mismatch percentages 1-38%. In summary, they were able to detect all the tested abnormal behaviours, including successful intrusions, failed intrusion attempts, and unusual error conditions [HFS98]. This of course depended on the correct alert threshold for the signal of the anomaly.

The original N-gram method of Forrest et al has been implemented as a special *seq-id* wrapper in the software wrapper IDS of Ko et al [KFBK00]. The wrapper system works in kernel space and is able to intercept and analyze system call sequences. The *seq-id* wrapper causes 5-7% penalty to the execution performance. It can also be used together with a misuse detection or specification based wrapper, but then the wrapper penalties get added.

This method has also been implemented when analysing anomalies in a distributed application utilising CORBA [SMS99]. There the observed sequences were not system calls, but CORBA remote method invocation sequences.

Wespi et al have augmented and modified this technique. First of all, they analysed audit events, not system calls. Secondly, their feature extraction has been done using variable-length sequences and the Teiresias algorithm, which was initially developed for discovering rigid patterns in unaligned biological sequences. The more abstract audit events could be used instead of system calls, since the system call arguments, which were not available in the audit trail, were not used for analysis. This variable-length method has improved the false positive rate [WDD00]. Variable-length sequences were also used by Marceau, but she constructed a finite state automaton from them and the method is thus discussed later [Mar00].

Cabrera et al turned the feature extraction process of this method upside down by using a string

matching classifier to extract data to separate *anomaly dictionaries*, i.e. system call sequences that are not in the normal data set [CLM01]. These anomaly dictionaries correspond to the idea of self for the anomalies instead of self of the monitored system. The incoming system call sequence was considered normal if the anomaly count in the sequence was zero against all anomaly dictionaries. The anomaly count was defined as the average of the relative anomaly counts in the set. The relative anomaly count is the number of anomalous sequences found in the trace, divided by the total number of sequences in the trace. The key problem of this approach is that all new anomalies are classified as one of the old or as normal. This can be solved by a hybrid approach where the anomaly count is first checked against a threshold value. If the anomaly count value was above threshold, the sequence was probably an attack even if it would not be in the anomaly dictionary. It was observed that the String Matching Classifier performs better for longer sequences, while the anomaly count detector performs better for shorter sequences. It was also shown that a hybrid detector, combining both schemes at their best choice of sequence lengths leads to substantial improvement of detection accuracy.

Jones et al also used sequence enumeration techniques, but the sequences were of calls to C-language libraries instead of system calls [JL01]. They utilised three different analysis methods: the original mismatch percentage from Forrest et al, the normalised anomaly signal from Hofmeyr et al and locality frames from Kosoresow et al. In two real tests against Apache web server, the mismatch percentages were 1.5 and 3.9 and the corresponding anomaly signal strengths 0.7 and 0.8. They also observed that the intrusion detection result is stable if the sequence length is >6. Their tests were done using a sequence length of 10. They also made successful tests with two buffer overflows, one trojan program and one memory depletion denial of service exploit. Additionally, they created a a trojan code attack against `mSQL` and were able to detect that better with library calls than system calls (signal strengths 0.8 vs 0.7, max. locality frame count 20 vs 7). They presume it is because system calls have too little variety in `mSQL`, it is doing basically I/O functions.

Ghosh et al compared in two papers five different algorithms for analysing the system call traces. In their analysis they compared the simple *N*-gram similarity measure technique from Forrest et al [FHSL96] with neural network methods (a back-propagation network and an Elman network) [GSS99]. Later they compared two other machine learning methods, string transducers and a state tester, to the Elman network [GMS00]. The results show that though the similarity matching approach worked fairly well, the performance can be significantly improved (particularly in reducing the false positive rate) by using neural networks [GSS99].

The sequence enumeration and similarity techniques have also been used for analysing anomalies in user command line behaviour [LB97]. Here the simple equality matching of system calls does not work very well, since the user can produce slightly different versions of the same basic commands. Therefore similarity matching techniques were used to score sequences of commands for closeness. The closeness measure was MCA-P, match count with adjacency and polynomial bound. While user input has quite different characteristics from system calls, similarity matching techniques may be an option for systems which take into account not only the system call, but also the parameters of the call. This is not quite in the domain of application intrusion detection, but is included here to illustrate other methods used to solve a similar task.

Call Policy Techniques

One potential approach to execution monitoring is to observe not the call sequences, but to create a list of calls the application makes when run normally, and then in subsequent runs limit the allowed calls to this known good set (i.e. profile).

Call policies have been explored in an operating system setting by Niels Provos in his *Systrace* system. There the system calls an application makes are recorded during a training time thus creating a security policy of the allowed system calls [Pro03]. This procedure is not altogether automated: if the application uses for example random filenames, the policy must be edited to allow that particular form of non-determinism.

The system call arguments are also considered here. Before making a policy decision, the system call and its arguments are translated into a system independent human-readable format. The policy language operates on that translation and does not need to be aware of system call specific semantics [Pro03].

This has been done also in dynamic Java sandboxing which for a given program consists of two activities: sandbox generation and sandbox execution. In the first, a sandbox profile is constructed by running the program with an instrumented JVM. During this training session, profiling information is recorded to the sandbox profile. The sandbox is initially empty and grows during the training run by accumulating records for each unique behaviour, i.e. method calls. Because nothing is added that is not observed, each sandbox is customised to a given program and context in which it is executed. During sandbox execution, behaviour (i.e. a method call) that is not in the profile is considered anomalous [IF02].

The JVM prototype looks only method invocations and their signatures, although the authors note that also memory behaviour, method arguments, patterns of methods, whole program paths or even other building blocks than methods could be used for anomaly detection [IF02].

This call policy approach assumes that the application run from which the profile was created did not include any attacks or exploits. Additionally, it assumes that any deviations from the profile are potential attacks. To lower the count of false positives, this requires that all variations of the valid application run-time behaviour must be present in the profile. Call policies are a sensible approach in a method or procedure call setting, where the potential number of calls is much greater than the number of system calls and thus the sequence enumeration technique detailed before may not be viable.

This method is somewhat similar to the policy-specification based techniques discussed later in Chapter 4.4.4, but with the difference that the list of allowed calls is not specified by an administrator but extracted from a known good application run.

State Automata Techniques

In the normal system call anomaly detection, the near real time speed of the detection is an issue. Therefore the system call traces need to be reduced for quicker processing. For that Kosoresow et al noted that the applications were deterministic in the sense that long sequences of system calls were repeated. This is also intuitive; if the application forks when it gets a connection, this forking sequence is always repeated. Therefore deterministic finite automaton (DFA) with macros can be used to build patterns of the various system call sequences.

However, creating the exact DFA of a process will be a problem. First, because every possible ordering of system calls would have to be made, the DFA would be very large. This is simply because the system call order can depend on the input data. And second, if the trace length is a function of data the application is processing, every variation in the data length should be included. If this is not done correctly this could lead to false positive alarms. Creating the macros for the DFA is also time consuming and potentially NP-hard [KH97]. For all these reasons deploying this approach to real systems would be difficult.

The simplest work constructing a state automaton was done by Marceau. There variable length N-grams of the system call sequences were used to create a suffix tree and an algorithm for creating

a finite state automaton (FSA) from it was presented. The resulting "self database" was smaller than the sequence database and thus the detector can be more effective at run-time. The algorithm was adjusted so that it does not create false positives. However, the resulting system was less sensitive to anomalies than the original N-gram work [Mar00]. The reason for this may be that the automaton was intentionally weakened by similarity compression so that a specific value for N would not be needed, but a general upper limit would suffice.

Various other state automata systems have also been researched. Michael and Ghosh used the N-grams from Forrest et al, to create a simple finite state machine. They compared the original work with the state machine and another technique, a string transducer. The results showed that the original N-gram work performed best if the detection rate was compared with the false positive rate. However, it had much longer training period than the new methods, i.e. with a limited set of training data the new methods had much better false positive rates. However, none of the techniques was able to detect more than 94% of the intrusions. This was due to limitations in the input data [MG02].

Sekar et al have also explored the use of finite-state machines to aid in the system call sequence analysis [SBDB01]. In their system they augmented the system call information with the program counter (PC), i.e. sequence-giving information, which allowed them to form the state machine automatically.

The actual analysis goes so that if an anomaly in the FSA occurs, it is not immediately flagged as an intrusion, but an anomaly counter is increased. If sufficient number of anomalies are observed during a time frame, an intrusion alert is generated. The anomaly count is decreased as time passes so isolated anomalies do not generate alerts. The system generated less false positives than the N-gram method used by Forrest et al. What is also interesting to note that the space and run-time overhead of the algorithm itself is low. However, the overhead for user-level system call interception used here was considerable, 100-250%.

Feng et al enhanced the FSA model by creating the *VtPath* model, in which the use of program counter is augmented and also call stack information is utilised for anomaly detection [FKF$^+$03]. In addition to the system call information and the program counter, the return address is obtained from the call stack into a virtual call stack list, which acts as a history of unreturned function calls. Virtual path between two system calls is the ordered list of non-common return addresses in the call stacks of the two system calls. It is an abstraction of the execution between two system calls.

All found virtual paths and call return addresses are stored during the training phase. In the detection phase each system call causes the virtual call stack being stored. Depending on the contents of the stack, return address and the virtual path between the current call and the previous call, the system can find out various anomalies, such as buffer overflows. The key benefit of the the *VtPath* technique is that it does not suffer from the impossible path problem. In other comparison, the false positive rate is similar to the FSA method, when they corrected some FSA implementation mistakes Sekar et al made. And since the system call interception was similar to the FSA research, the run-time overhead was still considerable.

Warrender et al compared various techniques in analysing system call traces, one of the methods was a probabilistic automaton called Hidden Markov Model (HMM) [WFP99]. HMM states are unobservable conditions in the system. The model attaches a probability to each state transition and to each type of output in the states. By this probabilistic modelling they can adjust their model over time and also easily represent branches in the system logic. However, the number of states and their probability must be given when the HMM is created, and therefore the training process of a HMM is very long, in the order of hours or days. On the positive side the detection accuracy was very high and the false positive rate was very low.

Also Wagner et Dean and Giffin et al have used state machines in modelling the application for

intrusion detection [WD01], [GJM02]. However, their works form a state machine from a program specification, the source or binary code, so their work is discussed with other specification based intrusion detection in Chapter 4.4.4.

Machine Learning

Several machine learning algorithms have been applied to application level intrusion detection as well.

As discussed previously, Ghosh et al compared three different algorithms for detecting the anomalies in the system call sequences: N-gram equality matching, back-propagation network and Elman recurrent network. The two latter systems are neural networks [GSS99]. Later, Elman networks were compared to string transducers and state testers [GMS00].

The results show that though the equality matching approach worked fairly well, the performance can be significantly improved (particularly in reducing the false positive rate) by using Elman networks. Elman networks were able to detect 77.3% of all intrusions with no false positives. Further, the Elman nets were able to detect 100.0% of all intrusions with significantly fewer false positives than either of the other two systems [GSS99]. Later, in user-to-root exploit tests, Elman networks achieved 100% detection of attacks very quickly at a false alarm rate of close to 3 per day. With remote-to-local attacks the results were not so good, 30% attacks detected at a rate of approximately 10 false alarms per day [GMS00].

The problem with Elman networks is the training time, it is in the order of thousands of minutes. The detection performance of string transducers were comparable to Elman networks, but the training phase is orders of magnitude smaller, in the range of tens of minutes [GMS00].

Comparing string transducers and state testers to Forrest et al's N-gram matching resulted in N-grams achieving slightly better performance. However, the N-gram method was much slower to learn, which is to say that it required a great deal more training data to achieve false positive rates comparable to string transducers and state testers [MG02].

Various rule-based systems have also been used. The earliest work used a time-based inductive engine (TIM) to infer rules from VAX/VMS user audit records [TCL90]. This was more of a successful feasibility study than an actual method performance evaluation. Later Lee, Stolfo and Chan used rule-based machine learning to determine anomalies from the system call traces [LSC97]. Their method was to apply RIPPER rule generation system to the traces to generate rules.

The rules generated from RIPPER can classify whether the trace is "normal" or "abnormal" compared to the rules. But it does not as such tell whether the trace indicates an intrusion or not. Therefore some post-processing was done to determine whether the abnormal traces really were intrusions. These post-processing heuristics try to weed out occasional prediction errors, which differ from intrusions by the length of the anomaly.

The key finding was that the rules for normal behaviour generated by RIPPER rule generator were well suited to anomaly detection. Additionally the best method for anomaly detection is to classify normal behaviour in RIPPER rules, classification of abnormal behaviour works best for misuse detection [LSC97].

Compared to the work in Forrest et al [FHSL96], this method is both faster and the rules have lower storage requirements. Because there were only 200-280 rules generated by RIPPER and only 100-140 rules on the average had to be checked for classification, authors state that this is potentially doable in a real-time environment. No practical tests have been performed, however.

As an example of applying machine learning to other IDS data, Gunetti and Russo used inductive learning in the form of relational decision trees to classify users into different categories based

on their Unix command-line command usage patterns [GR99]. They used a tool called ReliC and achieved about 90% user recognition rate, which is comparable to other similar research.

Data Mining

Lee and Stolfo, who previously investigated using machine learning for anomaly detection in system call traces, also extended their work to use data mining techniques [LS98]. The data mining approach relates to the construction of the base classifiers and how to identify meaningful features from the raw data stream. Data mining techniques, such as *association rules* and *frequent episodes* algorithms can be used to help in this feature extraction. An association rule algorithm can help find rules within one audit stream, while frequent episodes reasoning can help discover inter-audit patterns, i.e. patterns between audit data streams. And the latter is especially useful when looking at sequence information, such as system call traces.

Their aim was to make a multi-level model, where a set of base classifiers, such as the system call trace learner, would feed a meta learning task. The meta classifier would take input from all types of base classifiers (including network based mechanisms) and a learning algorithm would be applied for the overall anomaly analysis. This would be able to utilise information from many sources, not just application or network. However, in their work they have only very preliminary results from a *tcpdump* network packet trace but nothing from a system call trace.

Statistical Profiling

Anomaly detection in execution monitoring can also be made by creating a statistical profile of the execution, for example a call histogram, and comparing the executing program to the known good profile.

The histogram method has been used by Endler [End98]. All system calls in a call sequence vector were inserted to a large vector so that the occurrence of a call number 25 in the sequence increments the value in the 25th position of the larger vector. The larger vector represents how many times each event of all possible events has occurred in the sequence vector. A histogram classifier was used to divide the sequences to normal and abnormal. However, the histogram system size grows exponentially with the larger vector size so different techniques were used to selecting the most meaningful events. Hand-picking based on experience with buffer overflow attacks proved to be the most successful. To detect anomalous patterns from the histogram, a value was visually selected from histogram graphs for the anomaly threshold. This could be done, since this work was not real-time detection but work done on audit logs.

Additionally, module interaction can be used for execution monitoring. Elman and Munson have described a technique where any application written in C can be instrumented with a CLIC toolkit to insert hooks into the application source code modules [EM99]. A kernel-level tool is then able to follow the module interaction and compare the interaction profile statistically to a normal profile obtained beforehand.

4.4.3 Anomaly Detection in Input Semantics

There have been very few attempts to do semantic intrusion detection of any kind. Here the emphasis would be in analysing the user supplied data contents and its meaning in relation to the application and attempted action. In this sense this method falls under the class of anomaly detection.

Semantic analysis can be done in several levels. The most trivial one is writing these semantic checks into the code. This is of course dependent on the correct identification of potential hazard

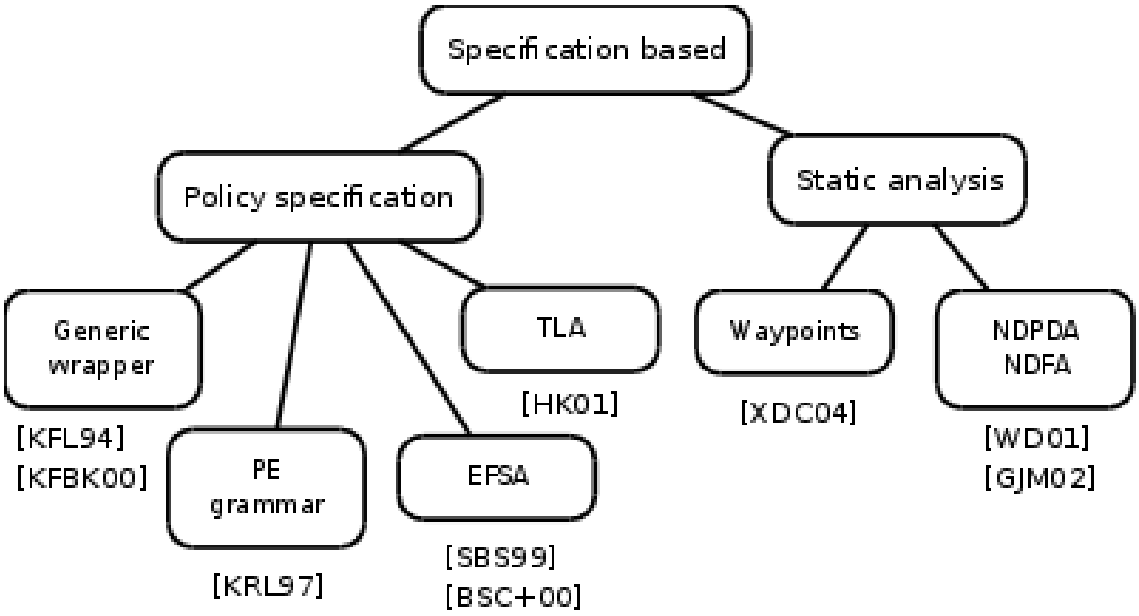

Figure 4.4: The specification based methodology map

spots and illegal values. Therefore substantial technical risk analysis, such as demonstrated by Sielken [Sie99] must be done when developing the software. This risk analysis must then be translated to actual agents to detect impossible conditions that could signal an attack.

Although Sielken suggested building the application IDS system so that it contains both application specific and application independent parts, the criticism expressed in Chapter 4.4.1 with regard to misuse detection in application code is still valid: much design and analysis and some coding is necessary to make this a reality.

Raz et al have investigated semantic anomaly detection in general data feeds, i.e. on input level, such as stock quote streams [RKS02]. They define an anomaly as an observed behaviour of a data feed that is different from our expectation. This expectation describes normal behaviour of the data feed. Since there was no real semantic data specification for their stock quote feed available, their method was to use two unsupervised machine learning tools, Daikon and Mean to infer invariants from the data feed. After the invariant learning, the feed data was compared against the invariants and an anomaly alert was generated if the invariant did not hold for the received feed data. Their method was able to successfully detect most of the data anomalies in the test scenario, the false positive rate being 0.3. It was usually reduced to under 0.02 (max 0.15) when adding voting heuristics, i.e. cross-checking with another data feed, but it did not help to reduce false negatives.

4.4.4 Specification Based AppIDS

There are also efforts to specifying the application behaviour more or less formally and alerting on any activity outside the specification. Two main styles have been identified; one specifies separately the policy that the application must follow, the other creates a model of the application based on the source or binary code and then observes whether there are any run-time deviations from that model. The found research and their relationships are shown in Figure 4.4.

Classic works in the specification based detection are modelling the network protocol(s) the

application uses. This is more in the domain of network based intrusion detection (NIDS), for example [SGF+02] and they are not discussed in more detail here.

Policy Based Systems

There is an approach where a specification of the privileged program execution is created and then their execution is monitored via Sun BSM [KFL94]. A policy specification language was also created for this task. The policy lists each system call the application is allowed to make, including key parameters. The policy language has some semantic information on the application and the kernel API, so it is able to process concepts like users, files and ports.

Most of these application policy definitions are site independent, i.e. a policy for `sendmail` in one Unix host is very similar to the policy in other Unix hosts, but some variables (such as log file names) are site dependent and must be tailored to fit each system.

Compared to some other system call AppIDS methods, this system does not specify the order of the system calls in the policy file. If the exploit uses allowed system calls in untypical order, or if the policy does not limit the system call parameters enough, or the exploitable problem lies in an application executed by this application, a penetration may still occur unnoticed, i.e. we get a false negative result.

A refinement of the method above was done a couple of years later [KRL97]. There a formalism for specifying the program specific policy (*trace policy*) was developed. The policy language, *parallel environment grammar*, is able to express parallelism and sequencing in the language. As above, it also has semantic devices to test some input values, such as the directory where the file or directory is to be created. They mention the system able to detect simultaneous edits of the same file by two parallel processes in their tests but do not give specifics.

Sekar et al have also modified this method [SBS99, BCS+00]. They defined a language, Auditing Specification Language (ASL) or Behaviour Monitoring Specification Language (BMSL), for application behaviour policy specification and integrated this with a compiler to produce an extended finite state automaton (EFSA) C++ class, from which an application specific system call detection engine is built. This engine then runs as part of the operating system kernel, intercepts system calls and uses the automaton to check them against the defined policy. In actual tests the system call monitoring system (SMS) caused 1.5% overhead. The overhead increases with the number of system calls in the application: an increase of 2000% in system calls caused the overhead to rise to 5% [BCS+00].

Calvin Ko has continued to work on specification based IDS with his team. Their kernel-level software wrapper system can be used also for specification based IDS, since the wrappers are defined in a wrapper definition language (WDL) which can be used to specify the allowed behaviour of the wrapped application [KFBK00]. Also this wrapper definition can access system call parameters and is able to impose semantic restrictions. The performance decrease from the specification based wrapper alone was 5-7% in tests.

All of these systems require some kind of security monitor, analyze or wrapper to detect the behaviour out of specification, as does the work of Herrmann and Krumm [HK01]. They have a formally defined and simulated security state automata with state dependent security constraints integrated with the container (i.e. wrapper). The security constraints are expressed with a formal specification technique cTLA, which extends Temporal Logic of Actions (TLA).

The basic implementation has Java application beans, adapters, observers and a monitor. There is an adapter wrapped around each bean. The bean interfaces with the environment only through the adapter. Thus all the bean interactions can be observed and if necessary, the bean can be sealed if anything attack-like occurs. The observers check that the bean complies with the cTLA process

specification. If the adapter detects an interface event, it forwards it to the observers by calling the corresponding action methods. If all relevant observers signalled the compliance of the event, the adapter really transfers the interface event. If, however, one observer refuses its corresponding action, the adapter seals the bean and reports the violation to the monitor. The monitor interfaces with the system administrator. The observers also link to the trust manager, which determines the intensity of security checks by their wrappers depending on the trust values carried by the scrutinised components [HK01].

The work also extended the Java security manager, since beans can also access system resources via standard streams, thus bypassing the adapter. However, resource access this way is controlled by the Java security manager and the manager was modified to report standard stream accesses.

Static Analysis

Static analysis is a form of testing that analyses the application source code or binary to detect errors or problems. This technique has also been used for application IDS.

Wagner and Dean used source code to create a model of the system call sequences [WD01]. The state machine is created from the source code, so this technique assumes the availability of it. On the other hand, Giffin et al, use the system binary to create the state machine [GJM02]. Both of these are in essence a form of static analysis. This work is here differentiated from the state automata discussed in 4.4.2 since the state machines here are generated from a specification: the source code or the application binary.

The work of Wagner and Dean suggests two non-trivial techniques for creating the state machine; a call-graph method, which creates a non-deterministic finite automaton (NDFA), and an abstract stack method, which creates a non-deterministic pushdown automaton (NDPDA). In this system there are no false alarms, since the state machine will accept all dynamically possible execution traces. On the other hand, the call-graph system will accept also impossible paths. The real problem with these methods is that they are non-trivial to implement and can incur a severe run-time overhead (in the order of minutes) [WD01]. However, the work on static analysis on binaries achieved significantly better efficiency (run-time increase of 13%). The reason for this was identified to differences in operating system library functions (Linux vs. Solaris) [GJM02]. This implies that static analysis is not as universal in practise as it is in theory.

Similarly, a set of kernel-supported waypoints can be created from the source code by static analysis. These function entrance and exit waypoints are then used for the application control-flow monitoring. When the application is run, a push-down automaton of the entrance/exit waypoints is created. An access monitor uses this push-down automaton information for control flow monitoring and what is more novel, permission monitoring. During the static analysis of waypoint generation, also the set of system calls (i.e. permissions) for each function is recorded and the access monitor can ensure system calls invoked in the context of a function appear in its permission set. From the analysis point of view this system has the good point that it can detect global mimicry attacks and return-into-others impossible path attacks [XDC04]. While Xu et al classify their work as anomaly detection, their method of using static analysis for the waypoint generation is more specification based.

However, local mimicry attacks are not detected by this waypoint method proper, although the authors have deployed other means (interface randomisation) to counter this problem. The performance tests were also somewhat problematic. When all system calls were monitored, the execution of an ordinary Unix program increased 3-5 times, most increase was in the time spent in kernel mode. By monitoring only dangerous system calls, the overhead was reduced by 16-70%.

4.5 Application IDS Reaction Models

There has been very little research done into the application IDS reaction systems, apart from the traditional "send an alert" feature. This is surprising since application level detection mechanisms, which are more or less integrated into the application, could also signal the application to change behaviour when a problem is observed. Some work exists, though.

One system call level example is in the pH system [SF00, Som02], where the mechanism is able to delay all subsequent system calls at kernel level by a configurable time factor if a problem with system call sequences is observed. Additionally, there is a separate configuration item for the `execve()` call delay threshold which configures the suspicion threshold where the execution of other programs via the monitored program should be blocked. This latter option is very useful against buffer overflows, which try to execute a shell. Although the performance hit of the pH system is several microseconds per system call, the overall performance penalty for real applications was between 4-10% in tests.

One other example is in the context of an auditing subsystem in the Java virtual machine. There is a separate response module that can be used to react to detected attacks in a specific environment. The response module initiates an appropriate response action when an attack or threat coming from a Java application is detected. The module sends to a dedicated thread in the JikesRVM a request for a particular response action [SKV03].

The specification based mechanism by Bowen, Sekar et al [SBS99, BCS+00] has also reactive capabilities. The policy defined in their language (ASL,BMSL) can include directives to delay execution, changing the application scheduling priority, changing the application environment, executing other monitor applications etc. And since the system call interceptor and the policy interpretation program function at kernel level, they are able to enforce these directives.

4.6 Intrusion Detection for Trust Management

As discussed previously in Chapter 3, this work attempts to automate trust-guarded authorisation, i.e. authorisation decisions based on trust evaluation (see Figure 3.6). What is the need for monitoring and intrusion detection in this scheme? Depending on the result of the evaluation, there are three different trust cases that need to be taken into account:

1. Successful trust-guarded authorisation, where the peer trust score is above the threshold and the action is allowed to proceed. However, we need to be aware of the fraud possibilities, where our trust is abused. From the negative point of view, this is the case of potential mistrust - misplaced trust.

2. Untrust, where the peer trust score is not above our co-operation threshold. Semantically this means that some trust exists, but in this particular case not enough to proceed with the collaboration.

3. Distrust, where the peer trust score semantically means that the enterprise has negative expectations on the peer conduct.

Intrusion detection and other security monitoring mechanisms surveyed in this chapter give us tools to handle these trust management scenarios. The distrust case is the easiest, distrusted peers must be prevented from connecting to the enterprise service. If this is not possible, for example because the application or service needs to be open to all the users on the Internet, any suitable intrusion detection or prevention mechanism can be deployed to detect and reject attacks. This

is a typical scenario for any internet based business and very much a standard practise in today's business world.

Untrust is middle ground between distrust and mistrust. Depending on the organisation trust policy, untrusted peers can be handled similarly to distrusted or given some other form of service that is within the peer trust score limits. In this latter case, the untrusted class of peers is elevated to the trusted class but for some other, perhaps downgraded service. Therefore the trust management issues are similar to the trusted class and no special treatment is required.

The main interest of this thesis lies in the first case with trusted and authorised peers, where mistrust must be managed. The trust management system evaluation criteria presented in Chapter 3.6 included requirements for peer behaviour monitoring and feedback to trust threshold calculations.

From the survey in this chapter several potential general approaches for this task can be preselected. First of all, the requirement for feedback to trust calculations means that the monitoring system needs to be *active* in its reaction model, i.e. an action is taken if something anomalous is observed. Whether the reaction is immediate or delayed depends on the needs and capabilities of the rest of the trust management system and the application. However, for collaborating community applications, where peers may dynamically interact with the service any time, immediate reaction would be best.

Secondly, it is clear that to separate abnormal, mistrusted behaviour from normal peer behaviour, mistrust monitoring requires some form of service level understanding. What is the expected format and semantic contents of the request, what is the expected answer? Therefore a form of *application intrusion detection* is needed.

From the analysis method point of view misuse detection methods are not of use alone here, since detecting known and typical attacks is not enough. Similarly, execution monitoring may not help us detect misuse if the attack proceeds very much like normal request as in a mimicry attack. The most useful analysis tools for misuse monitoring would then seem to be based on *specification* or *input semantics*. These of course can be augmented with other analysis methods.

Interestingly enough, these approaches have not been very much researched compared to misuse detection. Especially research is lacking on the semantic front. There were very few ideas that used any kind of semantic information either in aiding the analysis process or in defining the normal application behaviour. For example, practically all system call analysis mechanisms surveyed discarded the call parameter information. Exploring and developing methods that use semantic information could yield new and fresh outlooks also in the area of application intrusion detection.

As for the sensor data collection method, any method that gives the enterprise access to either the application behaviour for the specification analysis or input data semantics is suitable taking account the earlier requirement that the system should be application independent as much as possible. The TuBE system designed to implement the trust-guarded authorisation system with these constraints is introduced in the next chapter.

Chapter 5

The TuBE Approach to Trust Management

In the previous chapters, two key concepts: trust and application behaviour monitoring have been discussed in detail. How do these concepts relate to our goal of protecting enterprise collaboration? This thesis describes a system that binds these concepts together to provide a unified framework well based on trust theory to facilitate first of all trust-guarded authorisation and secondly detection of mistrust scenarios. Hence, the TuBE [1] framework flexibly protects the community member in an enterprise collaboration scenario.

In the following discussion, *client* refers to a piece of software that connects to a service or a single application in the collaborative community. Thus a client is an agent for a peer.

The structure of this chapter is as follows. First the TuBE high level concept and components are introduced, then the technical environment is described. The third section goes into design and architectural details where the high level concept is bound with the technical environment. Finally, the concept is evaluated against the requirements presented earlier in this work.

5.1 The Concept

To achieve the goals set previously, several separate components are needed. First of all, to utilise trust in an enterprise, an underlying trust model is required. In Chapter 3 it was discovered that trust is built using various pieces of information, so several trust support data sources and a component to combine all that data to information suitable for trusting intention assessment are necessary. Second, the trust-guarded authorisation function utilises this information from the trust model to grant or deny client authorisations. The third key part is the monitoring mechanism in the form of misuse detection function and the subsequent information feedback to the trust-guarded authorisation via both the trust model data sources and as immediate reaction. These components of the TuBE system are depicted in Figure 5.1 and their general functionality is discussed below.

5.1.1 The Trust Model and Supporting Infrastructure

In the TuBE system, trust is defined as *the extent to which one party is willing to participate in a given action with a given partner in a given situation, considering the risks and incentives involved*. This definition falls under the category of trusting intention in the trust type classification discussed in Chapter 3.2. Since the TuBE system is designed to automatically act on the basis of

[1] TuBE is a shorthand from Trust based on Evidence.

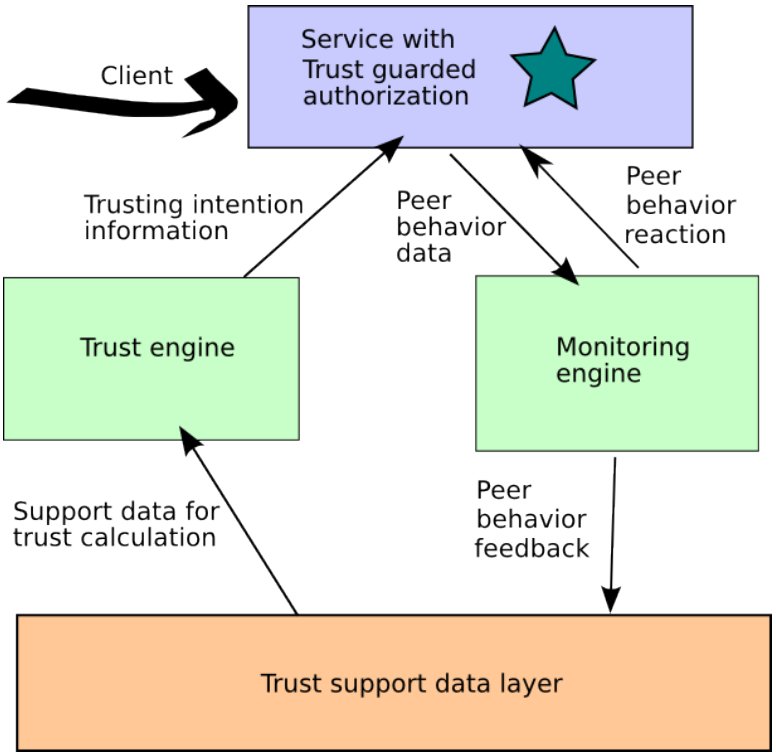

Figure 5.1: The components of the TuBE framework.

this intention, it must translate this intention to trusting behaviour. Trusting belief and context are utilised to form the trusting intention. Trusting belief consists of the action and trustee specific information, and potentially some default values, i.e. dispositional trust.

In the TuBE trust model, the key element is a relation, an n-tuple, between peers, actions, and certain properties of the environment. The trust-guarded authorisation, which is the trusting behaviour in TuBE, involves finding a balance between the uncertainty and risk present in the system and the potential benefits of the interaction.

The decision support facts in the TuBE trust model form a 7-tuple of *trustor, trustee, action, action risk, action importance, trustee reputation, trustor context*. Using the vocabulary developed in Chapter 3.4, the TuBE model is identity-aware, action-aware, business value aware, context-aware and history-aware. Considering the goals of supporting wide range of services and trust-guarded authorisation, the presence of trustor, trustee and action were obviously necessary. Of the business value factor group, risk and importance were chosen for experimentation although any other combination could have been selected as well. History-awareness is necessary to propagate the experiences from the monitoring function to the trust evaluation. In the TuBE system, the local peer reputation is used as the history component, no third party recommendations are used for the sake of simplicity. Finally, the presence of context is necessary as a link to the system trust component and situational awareness because the TuBE system is developed to be used in a highly dynamic and automated environment. High degree of automation benefits from the flexibility provided by context.

In Chapter 3.5 the theoretical trust use model was shown in Figure 3.1 and its applicability to computing authorisations was developed in Figure 3.6. The actual realisation of the trust formation process in the actual TuBE model is shown in Figure 5.2.

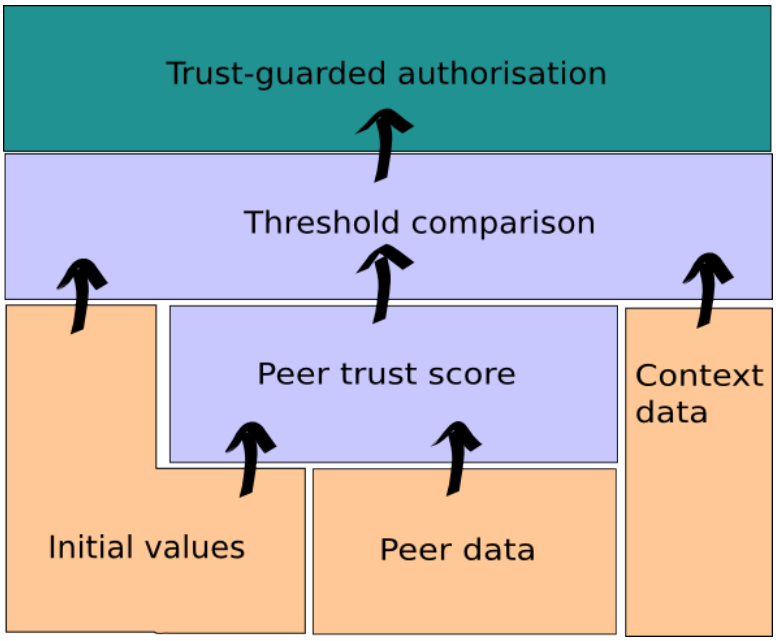

Figure 5.2: The trust model components in TuBE.

The TuBE model uses a threshold computation as the trusting behavior i.e. trust-guarded authorisation decision. The underlying belief formation process, which manifests itself in some kind of peer trust score, gathers data from various sources for the 7-tuple. The trusting belief is formed by finding the peer related information by using the peer identity, and finding the action risk and importance values by using the attempted action type. If no reputation for the peer is available, the initial value for dispositional trust is used.

This differs somewhat from the models presented earlier in Figures 3.1 and 3.6. These models had two ways for the action type to affect trusting behavior. One way was through situational decision to trust where peer identity was not used, the other through trusting belief, i.e. peer trust score, when peer is known. Since TuBE assumes the availability of a persistent peer identity, action type influences the trust valuation only through the peer trust score and the direct route is removed as unnecessary. Dispositional trust, i.e. initial values, still have two separate mechanisms. One, as explained above, is the peer related initial value if no reputation is available. The other mechanism is the threshold value for trusting in the threshold comparison calculation. The initial value selected for the threshold is a good way of expressing the dispositional trust attitude of the organisation not related to specific peers but to co-operation in general.

The focus of this work has been on how to achieve trusting behaviour using underlying trust concepts. Due to lack of resources some trust types described in Chapter 3.2 have not been integrated into this work. The trust types the TuBE framework does not currently support are the situational decision to trust and non-context system trust. The situational decision to trust is the trust type that ignores the peer identity and only considers the situation and potential context. This type was not directly considered in the TuBE system since the aim was to investigate the effect of local peer reputation information and feedback on the trust-guarded authorisation. Hence the focus has been on uniquely identifiable and authenticated peers. Authorisations guarded by situational trust may be an interesting future research direction for people interested in anonymity and privacy in collaborative systems.

The non-context aspects in system trust may also be an interesting research question, however not within the scope of this thesis and the current TuBE framework. Non-context system trust is the belief on the external and impersonal structures, such as laws and contracts, that facilitate the success of the co-operation. In a community setting, this is an interesting question since much of the co-operation success depends on the community participants. For future researchers I leave a question, how can the system trust to the community structure and operations be modelled and used?

Despite the current trust type focus, the aim has been to create the TuBE framework in a way that does not prevent its extension to accommodate other trust types. Additionally, the framework has been designed so that the trust decision support fact sources have been encapsulated so that it is relatively simple to change the trust model to consider different input factors.

To support the model described above, technical interfaces to various trust decision support fact sources are required. In the TuBE system these sources are:

- The reputation or history service. This is the subsystem linking the peer behaviour experiences to the trust-guarded authorisation. Observations made in the behaviour monitoring system change the peer history, which in turn affects the trust valuation of future requests. If necessary, this local history mechanism can be linked to a community reputation system making the trust model 3rd party aware. Although the current focus is to implement only the local history functions, in the following chapters this service is called the reputation service for generality.

- The context service or services. In the TuBE trust model we acknowledge the need for the trusting intention evaluation to be dependent on the system situational normality, context. The context information made available can cover data ranging for the enterprise financial situation to the IT systems' security status.

- The action business value service. Each service or application has a set of actions, such as "query price" or "buy" that a client can request. Each action has a set of business value data associated with it. In the TuBE trust model, this data is "action risk" and "action importance". These values must be declared and stored when the TuBE system is deployed to protect the service.

These three components require also management interfaces to bootstrap the data and in the context case, mechanisms to define context sources and their effect on the trusting intention. To limit the scope of this work, management interfaces are not discussed in more detail.

5.1.2 The Trust-guarded Authorisation

Trust-guarded authorisation produces the automated trusting behaviour service. Thus a component analysing the client service request, evaluating the trusting intention for that peer and converting this to authorisation is required. The result of this analysis is the decision whether the trust to that particular client–request combination in this particular situation is sufficient to let the service request be processed further.

For the authorisation decision, an underlying trusting intention calculation is required. In the TuBE system this is a simple function of the data received from the trust model compared with a threshold value. The three trust data values used for this function are action risk, action importance and trustee reputation.

Additionally, these three values are augmented with context. Theoretically, the situational normality side of system trust should influence just the trusting intention, i.e. threshold comparison.

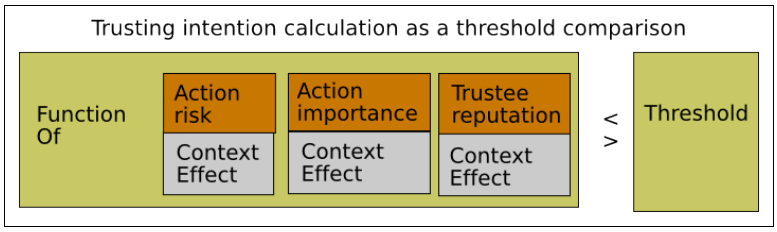

Figure 5.3: The conceptual model of calculating trusting intention as a threshold comparison.

However, in the TuBE system it was decided to experiment with a more fine-grained context that can directly influence intention calculation components. Therefore, the TuBE context changes these values directly. Compared to the theoretical framework the context data is given a more fine-grained effect in the TuBE system: instead of one adjustment point in the intention calculation three adjustment points are used. It is intended to investigate the effects of this change in more detail in the functional TuBE system.

These augmented three values are then processed and compared to a threshold value. If the comparison is positive, the trusting intention turns into automated trusting behaviour, i.e. authorisation is granted. The conceptual model of the calculation is shown in Figure 5.3.

The calculation function and threshold value should be settable by policy. There may be organisations that may want to emphasise the local peer reputation more and in some cases the attempted action risk and importance may weigh more in the trust evaluation process. Implementing and experimenting with different functions should give interesting data on the overall system behaviour in different usage scenarios.

Since clients need to be uniquely identified, some form of peer authentication is also required. It is assumed that such a system is in place and provides TuBE with unique peer identification, but the TuBE framework does not itself include an authentication framework.

The authorisation function requires also administrative support. This includes mechanisms to write trust decision policies and dispositional trust defaults or initial values, and the threshold value itself.

5.1.3 The Monitoring Service and Feedback

The introspective peer behaviour monitoring is in TuBE done with a mistrust detection system. Mistrust detection aims to find out whether the trust in the form of authorised action, was misplaced. Therefore TuBE deploys an application level service for intrusion detection, which forms the TuBE introspective monitoring mechanism. It is necessary to monitor the client interaction with the enterprise service to determine whether the peer behaviour as seen in the requests is anomalous or not. The tests to experiment with and select the most suitable algorithms for this task are left for the TuBE implementation phase.

If a problem is detected a reaction is necessary immediately. In TuBE the reaction is twofold. First of all, a request is made to the authorisation component to stop or limit the interaction immediately. Second, a message changing the peer reputation in the enterprise is sent to the reputation mechanism. Since the reputation score is used by the trust-guarded authorisation, the subsequent authorisation attempts are influenced by this changed reputation score and possibly denied.

5.2 Technical Environment

To ground the TuBE concept and the theoretical background into the real world, it is assumed that the technical environment of the enterprise collaboration and business communities is Web Services. Web Services is also the currently preferred platform for Service Oriented Computing.

A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP protocol messages, typically conveyed using HTTP with an XML serialisation in conjunction with other Web-related standards [BHM+04].

The extensible markup language, XML, used in the Web Services environment has gained a strong position as a general content encoding format. The key ideas of XML are support for wide variety of applications (i.e. extensibility), human-readability and easy development of XML processing applications [Wor04].

The benefits of using XML in a business environment are the following [MG98]:

- Businesses can describe services in a manner that can be widely understood.
- One set of documents, forms and messages can be exchanged by businesses with different internal business systems.
- Errors in re-keying data are reduced because data can be transformed through gateways.
- Frequent changes in business process can be handled without substantial engineering cost.
- It leverages investment in legacy systems and can be used with latest Internet technology.

Since XML is a general content encoding format, it can also be used in a networking setting to describe protocol messages. It has rapidly become the most widely used wire format in new messaging systems. The most widespread example of XML messaging is SOAP, Simple Object Access Protocol, which is a lightweight protocol intended for exchanging structured information in a decentralised, distributed environment [Wor03]. SOAP messages are defined as XML infosets. SOAP enables exchange of SOAP messages using a variety of underlying protocols. The description how the SOAP messages are transported on top of another protocol is called a binding. A typical SOAP transport protocol is HTTP.

The effect of XML to businesses in the form of spontaneous commerce, business automation and virtual enterprises has been understood for some years now [GTM99] but the technology has not been quite mature yet. Web Services have in recent years emerged as the pervasive glue that will enable applications with varying implementation platforms and internal messaging techniques to interoperate. In this sense the Web Services initiative is an unqualified success. However, the Web Services world is far from finished. While companies currently develop Web Services applications for business purposes, major issues have been left unexplored in the specifications or are still in the development phase.

In the collaborative communities interoperability stack discussed in 2.1 SOAP, XML, WSDL and other associated standards are in the technical layer on top of which community services and processes can be built.

In the service-oriented computing environment, basic services can be combined into composite services. Combining the services in a Web Services environment may be done utilising the Business Process Execution Language for Web Services (BPEL4WS) [BIM+03]. It provides a language for defining business processes utilising the component services. A choreography defines the sequence and conditions under which multiple cooperating independent agents exchange

messages in order to perform a task to achieve a goal state. W3C has defined Web Services Choreography Interface (WSCI) [W3C02] and the accompanying Web Service Choreography Description Language (WS-CDL) [W3C04b] for defining community choreography. An orchestration defines the sequence and conditions in which one Web service invokes other Web services in order to realise some useful function [BHM+04]. In practical terms these define the externally visible functionality of a community. BPEL4WS is an orchestration language, but it could be extended to cover also inter-organisational issues.

In addition to being functionally important, monitoring of the choreographies and business processes is also important security-wise. For example, denial-of-service attacks targeted towards one or several participating systems are very problematic in a business process and they can be observed with suitable monitoring tools. When a problem of this type is observed, the community can possibly reorganise itself around the observed problem.

One of the key problems is that monitoring functions are not specified yet, not in either technology, service, community or business level. The specifications above do not include any mechanisms for monitoring. Therefore this thesis starts by offering a solution to the service and technology layers.

Some other unfinished or work-in-progress sections are trust and security. Specifications do exist for some simple technical tasks, such as secure messaging, but more advanced security mechanisms, for example on the collaboration side, are unspecified. This lack of a complete security framework causes problems for organisations planning to deploy Web Services applications in a major scale. Therefore it is also beneficial to the Web Services community to describe new and emerging solutions such as introspective monitoring, to aid in the security and trust framework development.

5.3 Design and Architecture

From the overall components depicted in Figure 5.1 a technical architecture shown in Figure 5.4 was developed. Each of the components are discussed below in more detail.

5.3.1 Trust-guarded Authorisation and the Wrapper

The TuBE framework has been developed for an environment where the participants in a collaborative community receive and offer services from a (set of interconnected) business applications via a Web Service interface. This interaction and its effect in the business application needs to be captured to detect security problems at the application level. As discussed in Chapter 4 this event capture can be achieved by listening to one of the interfaces, by hooking to the operating system or application platform (such as Java VM) or by modifying the application.

Because the application description in the Web Services Definition Language (WSDL) is the only common infrastructure element in Web Services applications, in the TuBE project it was decided to intercept the protocol messages, which are defined in the WSDL description. The interception happens with the help of a *wrapper*, which in its core is a simple WSDL proxy. Each network interface, or TCP port, of a particular service has its own wrapper. The trust guarded authorisation is achieved in the wrapper by making decisions on whether the incoming message can be allowed to be processed or not. In addition, the wrapper interacts with the monitoring intrusion detection system (IDS) by providing it a copy of the traffic. When the intrusion detection system observes an anomaly, it may want to stop or restrict a particular interaction and the wrapper must be able to process those requests.

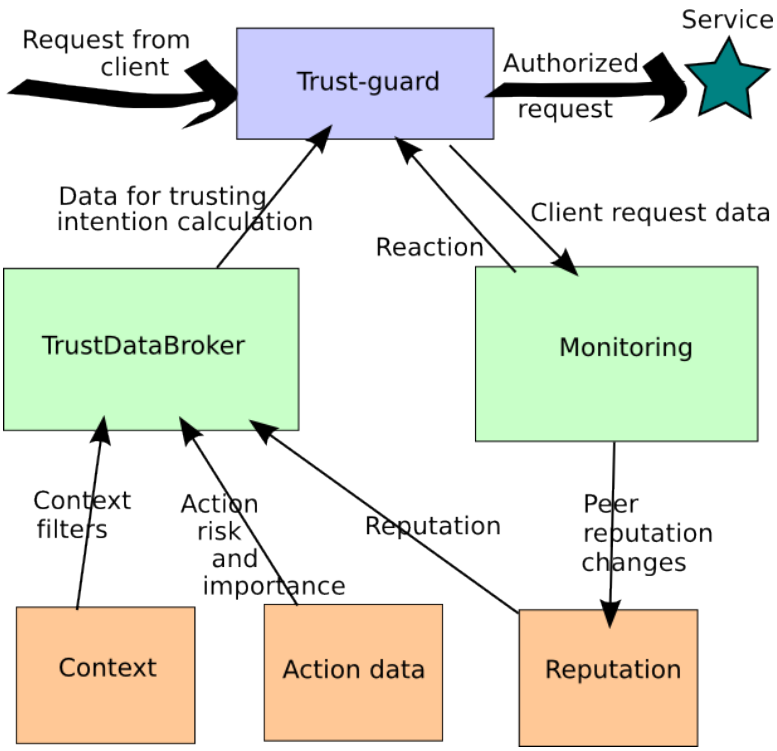

Figure 5.4: The components and data flow of the TuBE framework.

Since the wrapper must be application-specific, the concrete wrapper is specialised from a general wrapper model included in the TuBE platform by an application-specific WSDL description. Due to this specialisation based on the available WSDL definition, this platform core component remains application platform independent, unlike in some other wrapping systems [HK01]. This approach is also legacy application friendly, since application source code is not needed for analysis or modification.

This interception style has the drawback that the application cannot be observed directly. The view of the application is strictly limited to the communication between the peer and server, and there are no means for observing the internal state of the application from this level. It would be beneficial to have this type of internal information, thus platform specific interception possibilities are left for future expansion.

The wrapper also feeds the message data to the application intrusion detection mechanism. Additionally, it takes feedback from the intrusion detection mechanism when the IDS has observed an anomaly and wants a particular interaction to be stopped or restricted.

The wrapper is responsible for evaluating the trust status of the peer making the request by utilising the trust model presented in Chapter 5.1.1. This evaluation is then translated into an authorisation decision. Since a trust query must block the peer request while the decision is being made, the trust decision is a very performance-critical component. Therefore, it was decided to perform the trust decision in the wrapper instead of a separate decision component. This includes first calculating the peer trust score (trusting belief), utilising this score in a threshold comparison (trusting intention) with context information (system trust) and finally doing the authorisation decision (trusting behaviour).

To make the trust-guarded authorisation decision, the wrapper needs to have all the supporting

data, i.e. trust tuple values, available. As this data is volatile, it is placed in a cache hashed by the peer identification and action. Since frequent peers tend to stay frequent, the cache should purge the least recently used (LRU) data first. This policy can be implemented with a two-space or semi-space algorithm, where client data is moved from Old-space to New-space whenever it is used. If the cache needs to be purged, the Old-space entries are first removed and all of the New-space is flipped to Old-space. If the cache does not contain the support data needed for a trust decision, the wrapper must request more data from the trust data broker. The decision may therefore be delayed.

The actual threshold formula for calculating the trusting intention and also how to calculate peer trust scores are isolated into redefinable functions, in order to support using alternative calculation methods for other trust models. The specific function to translate the values in the trust model tuple to a decision is specified as a policy element by the administrator. Experiments or simulations would be useful to determine what kind of functions work best in community settings and whether the service characteristics bring any requirements for the calculation.

5.3.2 The Trust Data Broker

The TuBE trust model requires information from several sources. Action risk and importance values are seldom changing data that come from an administrator-configurable GUI application. Trustee reputation is received from the reputation subsystem. Local context data may be received from multiple context subsystems. Finally, the combined data should again be passed to multiple wrappers, one for each instance of the service.

The broker stores all information for the trust decisions as maps `(peer_identification, action, reputation)` and `(action, risk_formula, importance_formula)` in a database. 'Peer identification' is a unique trustee identification, such as a public key. 'Action' is the name of the attempted action in the WSDL description. Risk and importance formulae are discussed later in relation to the action subsystem.

The changes received from the data source components trigger different operations in the broker. Reputation changes as well as action risk and importance changes change the corresponding values in the database; a reputation change is user-specific, while risk and importance values are action-specific.

Context changes are formulae that may potentially change every value in the database. Therefore, it is more efficient to keep the change as a *context filter* instead of modifying every data item in the database. Context filters are applied when data is sent to a wrapper so that if the data matches the filter, the data is changed according to filter instructions. Each context source has one context filter and as there may be several context sources, these filters can be chained in the data broker. For example, a context filter could state that all actions are twice as risky as normally if the system context is "under security attack" and the filter would double all action risk values before sending them to the wrapper. The concept is illustrated in the Figure 5.5.

While wrappers are application specific and each interface requires its own wrapper instance, one trust data broker could serve several wrappers for the same application.

Brokers can also serve several different services within the enterprise which are using the same trust valuation. In practise this would mean that if a peer misbehaves with one service, its trust valuation would decrease also for other services within the enterprise. This would be a good experiment for the actual framework. However, for scalability and fault tolerance reasons there should be several brokers in an enterprise setting.

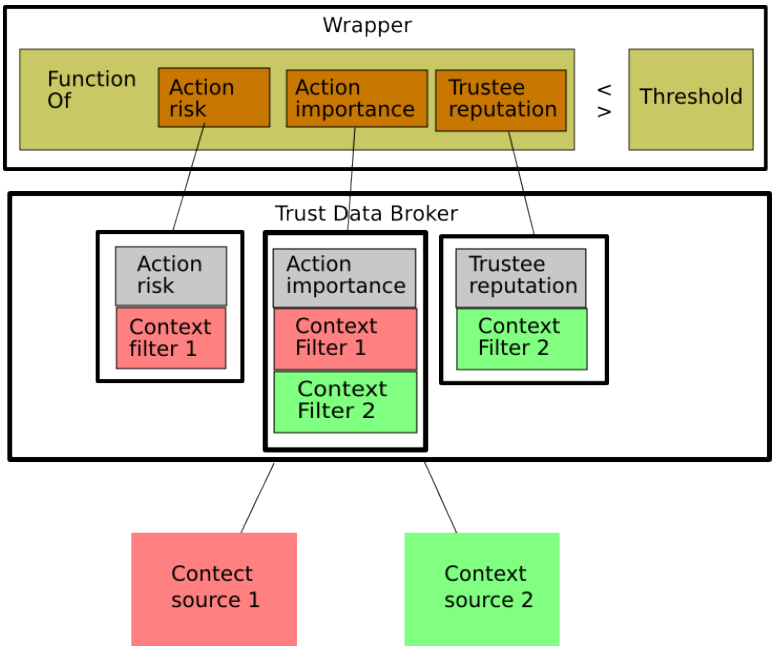

Figure 5.5: The effect of context filters to trust calculation.

5.3.3 The Monitor

The intrusion detection system receives intercepted peer message data from the wrapper and must decide whether the peer requests are against the system security policy. As discussed in Chapter 4, there are three basic philosophies to intrusion detection: misuse detection, anomaly detection and specification based detection. The misuse approach relies on patterns (i.e. signatures) of known attacks for security problem detection. The specification based approach is interested in finding out whether the system behaves out of specification. Anomaly detection compares the current observations to known good behaviour profile and raises alarm if sufficient difference between them is found.

In the TuBE project anomaly detection was selected as the main behaviour monitoring approach from the palette of intrusion detection choices. The reasons for this are twofold. First of all, there are not very many known problems in generic Web Services application environment for effective misuse detection. Secondly, a good application level specification is time-consuming to create and monitor [KRL97, WD01]. Thirdly, it is the aim of this work to be as application independent as possible, so deep application level specifications do not fit the requirements.

In anomaly detection there are several possibilities for the particular method. As discussed in Chapter 4, this general approach can be applied to monitoring the execution of the application, for example in the system call or remote procedure call context, or to detecting anomalies in data semantics. In the TuBE project especially the semantic anomaly detection is of interest.

The semantics of an action can be divided into two broad areas: control flow semantics and data semantics. Data semantics is straightforward: what is the meaning of the input parameters and the action output? In some cases it is also interesting to know the meaning of the data the action internally handles and stores to produce the output from the input. The control flow semantics is more complicated and is interested in the question, what does this action accomplish. In a local business process an action may have a simple meaning, but in a series of actions, especially in an

external business process within a community, the set of actions may have a different combination semantics. For example, in a local business process an action of reading a single record is just obtaining user information, but making a thousand similar reads about different users may indicate some kind of data harvesting operation, which probably should not be allowed for privacy reasons.

The Web Services interface is given in a WSDL description, which does not contain explicit semantic information, but provides information about the Web Services operations, such as message components and parameters. However, this description can be augmented with semantic information, such as ontological annotation and action preconditions and effects [SVSM03]. This type of augmentation will allow control flow semantics to be embedded into a WSDL description. Additionally, some preconditions or limits should be placed on the data as well. This is currently an uninvestigated area.

The TuBE project focus is on the semantics of data and anomaly detection in the data aspects of the Web Services interaction. The control flow semantics is also interesting, but the area is examined in more detail in the web-Pilarcos-project. The TuBE framework has also provisions for changing the analysis algorithm provided that the algorithm is using the the message data for analysis.

The TuBE architecture has been specifically designed in a way that the application IDS engine is decoupled from the trust decision component. This means that any suitable intrusion detection methodology can be used as long as it can utilise the data from the client–server message exchange, which is captured by the wrapper. For example, specification based network level intrusion detection is quite possible, if a full specification is available for the application message interchange. The TuBE framework includes mechanisms to redefine the IDS analysis class. The analysis can also be made by several IDS components, each with a different methodology.

5.3.4 The Reputation Subsystem

The reputation mechanism stores and manages information about peers' past behaviour, aggregated into reputation. Locally, the reputation information (or history in the trust vocabulary) is updated through first-hand experience on how a particular peer has behaved towards the system. On the community level, members can share their experiences and provide second-hand information about the peer as well. This makes it possible to receive reputation information even prior to making initial contact with a peer.

In order to successfully share reputation information, there must be a commonly agreed protocol for the reputation exchange, and a common ontology and semantics to support the interpretation of the shared information. This may prove a challenge if the members of the reputation-exchange community have very different action set s associated with their experiences, possibly forcing the community to share only generalised reputation information instead of action-specific items of experience. This area is left for exploration in future work, the TuBE framework assumes for now that such a reputation mechanism is in place.

If there is a link to a community wide reputation mechanism, a separate interface is required to communicate with it. The interface receives reputation change data from the IDS, transforms the data to a form with commonly accepted syntax and semantics, and sends that to the reputation system for sharing. This same component also publishes the peers' reputation information to the Trust data broker. This interface definition is an administrative task at TuBE deployment time.

There are two choices in sending experience data from the IDS to the reputation system. One is to send only negative reputation reports and one is to send also positive reports. The negative-only style would be more efficient, since negative reports, meaning an alert from the intrusion detection subsystem, are probably very rare. On the other hand, if there is no way to ever increase

the participants' reputation in the community, the reputation system would eventually become challenged and possibly by ever lowering reputation scores. Therefore, it was decided to send also positive reports in the TuBE framework. However, this is something that needs to be experimented with: how do the changes in the peer reputation affect the use of the service.

If the contacting peer has no reputation or history information in the system, the TuBE system uses an internal dispositional trust default value in building the trusting belief for that peer. This can happen when the TuBE system first comes operational and the whole reputation database is empty, or when there is a connection with a peer with whom there has been no interaction before. After the initial encounter there will be some behavioural history so that a reputation score for that peer can be determined.

5.3.5 The Action Subsystem

Since trusting belief in the TuBE model depends on the attempted action and the risk and importance factors that are attached to the action, there needs to be a subsystem that contains this information. The action information is stored in an application specific database

The service actions and the associated values are defined by an administrator via a user interface with the help of the service's WSDL description. Additionally, the community action information is described during the community role realisation. The risk and importance values can also be changed during the application run time via the management interface.

The action risk and importance values are integers. The idea is not to define any real semantics for these values, other than the basic numerical order, i.e. a larger number means a larger risk or importance. This enables the administrator to use any kind of tool or mechanism, such as rank ordering or monetary risk analysis, for determining the correct risk and importance values. The effect of different risk analysis methods on the actual behaviour of the system is an interesting research question for the future.

Since the risks are not necessarily the same even for all similar actions, such as "buy", there needs to be a parameter mechanism for relating the risk to the monetary value or other numerical parameter of the action. The parameter information is available since the action definitions must be done with the help of the WSDL file anyway. This formula or parametrisation must be propagated to wrappers for the trust calculations.

5.3.6 The Context Subsystem

The context subsystem supports the trust model by providing systems trust in the form of context data, which is utilised in the trusting intention calculation. As discussed earlier, context can include any description of the internal or community situation normality that affects the trust evaluation. The context information may come from a wide variety of sources, potentially even several sources So every context source has an interface component that manipulates the context data to a suitable filter format to be utilised by the trust data broker.

When the context sources are defined by an administrator, the definition includes information on which trust decision factor the context information changes and how the context source data is mapped to a suitable context filter.

As a result, it is possible to layer named filters on top of the data sources, depending on which named contexts are in currently in effect. Thus, the data values flowing from each data source towards the wrappers change to reflect the current contexts.

5.3.7 The Data Flow and Application Programming Interfaces

Each of the technical components in the TuBE framework discussed above have an application programming interface (API) which also describes the data flow between them.

The Component Interaction Paradigm

There are several logically separate components in the TuBE framework. These could be implemented as a single service residing in one system. However, the requirements of the individual components make this choice impractical. First of all, the intrusion detection component is a very much time and performance critical component, the sooner the misuse or anomaly is detected the better. Additionally, to support multiple trust models, the trust data sources need to be modular and easily added or removed. Therefore loose coupling between the TuBE components is desirable.

The publish/subscribe interaction paradigm provides subscribers with the ability to express their interest in an event or a pattern of events, in order to be notified subsequently of any event, generated by a publisher, that matches their registered interest [EFGK03]. Publish-subscribe systems provides loose coupling in three dimensions: space, time and synchronisation. Space decoupling is achieved since components do not necessarily know each other and they do not need to share references to each other. Time decoupling is caused by the fact that the participants do not need to interact at the same time. The notification of a particular event or action may happen after the fact when a suitable subscriber is connected. Also the event providing application does not need to block while the event is produced, which achieves synchronisation decoupling.

All these characteristics are of use to the TuBE framework, thus the data flow between components by default utilises a publish/subscribe paradigm.

This paradigm can be realised in the TuBE implementation by several means. There is the the Java Message Service (JMS), which uses topic-based subscription mode, but it can also provide point-to-point queues if the publish/subscribe domain is not desirable [HBS+02]. Alternatively, there are a set of Web Services publish/subscribe standards, which enable Web Services components to utilise this paradigm [OAS06]. The latter may have some benefits since the protected service is a Web Service and the data sent for analysis is XML based. However, the actual selection is left for the implementation.

The IDS API

The wrapper intercepts application messages and sends them to the monitoring i.e. intrusion detection subsystem. In the publish/subscribe paradigm, the wrapper acts as a publisher and the IDS subsystem(s) as consumers.

The data channel from the wrapper to the IDS can be implemented either as a queue or as a publish/subscribe topic. A queue would be natural when only one IDS consumer listens to the messages the wrapper sends. However, a publish/subscribe topic is preferable since the framework should be flexible enough to enable the enterprise to use several different monitoring or intrusion detection systems that listen to the same traffic flow but analyse the data using different algorithms. Additionally, this approach would enable several wrappers to send their data to the same analysis engines. The best division of topics and the practical interaction of the wrappers, monitoring components and trust valuation in this multi-pronged approach are interesting questions for future research.

For the intrusion detection system the order of messages received and analysed may be significant. Therefore the actual publish/subscribe implementation needs to support ordered message delivery.

The intercepted messages are sent to the IDS "as is", without modification, so that the IDS has as much original information about the messages as possible. Depending on the actual intrusion detection model, some data loss may be acceptable, but a decision to leave some data unused or unprocessed should be made by the intrusion detection subsystem.

In addition to the lossless transfer, the wrapper has possibilities to enrich the intercepted messages with context information that are not necessarily present in the intercepted XML messages. Examples from such data are the lower protocol level connection metadata: the client IP address, timestamp, connection duration, encryption details etc. Whether such information actually enhances the IDS analysis capabilities, is left for future research.

Thus the published message contents to the IDS subscribers is as follows:

- Message: the message client sent to the service in its complete XML form.
- Application context: encryption details, such as SSL encryption mechanisms
- TCP context: the TCP port, connection duration thus far
- IP context: the source IP address

It is assumed that the message or the application context contains the peer identification in some acceptable form for the service (for example as an XML signature or SSL client certificate).

The Reputation API

Since the focus of the TuBE system is in authorisation and monitoring at this stage it is assumed there is a reputation mechanism in place. Therefore nothing can be assumed anything from its nature or interface. This means there must be a component that receives reputation changes from the IDS component and translates them to be semantically and syntactically valid changes for the specific reputation system.

The reputation system subscribes to a reputation topic, where the IDS subsystem publishes messages with the triple:

- Peer identification.
- Transaction identification.
- Attempted action.
- Success opinion.

The peer must be uniquely identified because the reputation is personal, for example using the peer public key, which is potentially available from the XML message signature. Transactions need to be uniquely identified so that several opinions from the same transaction can be managed. The attempted action may influence the reputation system. The success_opinion is the intrusion detection system's view on how well the transaction was completed.

The natural way of an intrusion detection system to view things is by alerting on intrusions or other anomalies. It has no semantic tools to score success. Therefore a very simple discrete scale [-1, 0, 1] is sufficient for the success opinion. The semantics of 1 are such that the transaction was a success and no problems were observed. On the other hand, -1 means that there was a clear sign of intrusion. The zero can be used to denote minor oddities that do not warrant a real intrusion alert. The availability of 0 depends on the actual intrusion detection algorithm; for example in a

specification based analysis the transaction is either out of specification or not, there may not be any middle ground.

This success scoring can then be translated in the reputation mechanism specific component to reputation change messages. For example, a positive success report could change the peer reputation with an increment of 0.00001, while a negative report would instantly set the reputation to 0 or any suitable value signifying bad reputation. This transformation procedure is very reputation system specific and thus it not discussed any further. It is assumed a reputation mechanism exists and a suitable function and an interface component can be created.

It is important to note that the success opinion is the view of one particular intrusion detection component and if the component does not recognise the attack, a positive opinion will be given. A success opinion does not necessarily mean that there weren't any problems in the request. This is countered in the TuBE system by the ability to have several intrusion detection methods all analysing the same data with a different method. Even if a specification based IDS component sends a success report, another IDS component doing semantic analysis can report an anomaly. These different reports on each single transaction should be correlated into one trust value update per transaction identification to prevent result skewing. From the security point of view, anomaly reports should have priority over success reports, so it may be sufficient to discard the positive report if any of the reports indicates a problem.

If the enterprise utilises several different introspective monitoring or intrusion detection systems, the monitor subsystem in TuBE must include a component that receives input from different monitor components, correlates them and then sends the update to the reputation subsystem. The message delivery from the IDS to the reputation system must be reliable, as it is crucial for the operation of the dynamic trust system that all changes to the reputation of the peer are actually received by the reputation system.

The Wrapper API for IDS Reaction

If the intrusion detection system observes an intrusion or other type of an anomaly, it needs to signal the wrapper to do something to the connection to prevent or limit any further harm. It should be noted that when the IDS system observes the problem, the command has already been executed in the real server, so the TuBE model is not intrusion prevention but detection and the key reaction is sending behaviour feedback to the trust system.

As discussed above in the connection with reputation, the intrusion detection mechanism can not always differentiate clients to several different classes where the reactions should be different. The methods have usually been developed for identifying strongly only the class of real intrusions. Therefore, the IDS subsystem may not be able to utilise a wide range of reaction possibilities.

Various reaction possibilities are discussed in Chapter 4.5, and the chosen reactions are influenced on the component that does the reaction. Currently in the TuBE architecture the component capable of affecting the application operation is the wrapper, thus the reaction possibilities in the TuBE system are currently:

1. Break client connection, if it still is active.
2. Refuse future connections from this client either permanently or for a designated time.
3. Add a delay for processing each future request.

These reaction possibilities are not mutually exclusive, the system should be able to execute several of these choices if so requested. This set of reactions towards a client can be expressed as a message of chosen action combination together with the peer identification:

- Peer identification.
- Break: boolean
- Refuse time
- Delay time

In this message, refuse time or delay time indicate how long the peer connections need to be refused or how large a delay factor needs to be applied.

Other reactive possibilities, such as modifying the service application scheduling or resource availability, could be added with suitable additional components.

Using the publish/subscribe paradigm, there can be one reaction message topic to which all wrapper instances subscribe.

The Broker APIs

The trust data broker component is responsible for managing the trust data coming from separate data sources. In the current TuBE trust model these sources are reputation, action and context.

This can be achieved by forming each source a dedicated publish/subscribe topic. The trust data broker subscribes to all of these sources and the data sources publish the changes to these topics.

As has been previously noted, the actual reputation mechanism is not known, so the data received from there must be adapted to suit our needs. Therefore the same component that translates the IDS opinions to reputation specific change messages must also adapt the data from the reputation mechanism for broker use.

The TuBE system requires the reputation to be an integer. The data sent from the reputation subsystem to the trust data broker will be wrapped to a message:

- Peer identification.
- Action.
- Reputation.

It is assumed in TuBE that the action information sent from the reputation system (or the reputation interface) is compatible with the action information created in the Action subsystem. This requires there to be a common ontological base for the reputation interface component to translate outside reputation to a format suitable for TuBE. This is not a trivial requirement but since reputation mechanisms are not in this work's main focus, this is not explored further but left for future research.

The action database is a key user interface to the TuBE system. The action subsystem gets the application WSDL description, lists the actions and the administrator assigns risk and importance values to the actions. The message to the trust data broker topic consists of the following data:

- Action.
- Risk formula.
- Importance formula.

Action is a string that is in the standalone case copied directly from the WSDL description. Risk and importance formulae are strings that describe how the risk and importance are finally calculated for this action.

For example, the risk of a "buy" action could be described by the string "1*$amount", where 1 is the risk base value and "amount" is the name of the parameter in the WSDL description that specifies the number of bought items. Alternatively, the formula can be just an integer in a string form, which denotes a fixed risk or importance value.

These values can be modified by context. For every context source it must be defined how the context data changes the trust vector values. This filtering information sent to the broker in a message consists of

- Context name.
- Factor name.
- Factor parameter.
- Change formula.

Context name is the name for this particular context source. It needs to be unique among the context sources usable by this application. The factor name is one of the trust vector sources from the list of (action risk, action importance, peer reputation). Since all of the factors listed require a target (either a peer identification or a specified action) factor parameter is a string or regular expression denoting that target. Finally, change formula is the arithmetic formula denoting how the factor is to be modified.

An example filter could be (finance, "risk", "buy_*", "+2") denoting that the financial situation requires all buy action risks to be incremented by two. When a context is deployed, the context interface needs to be built and filters configured. Since the configuration needs the list of available actions, the configuration should be done in the management user interface.

The API for Trust Valuation Data

To achieve trust-based authorisation, the wrapper needs to calculate the trusting intention and compare it to a threshold level based on a set of data. When the TuBE system is deployed to protect a service, the wrapper needs to prime its data structures with a set of initial support data.

This is a single initialisation time action and can be implemented as a straightforward synchronous API call. The broker responds with initial data in the form of tuples:

- Peer identification
- Vector of action vectors

to the wrapper. This initial set can be the set of previously known community members or a random user set just to initiate the wrapper cache.

For the run-time trust valuation changes, the data broker has a one topic for which the wrappers can subscribe to. The messages for this topic are in the form of:

- Peer identification.
- Action name.
- Risk change.

- Importance change.
- Reputation change.

Here the peer identification and action name fields should support wildcard values, i.e. one message could authorise several changes to the support database.

5.4 Evaluation

The hypothesis of this thesis was that the use of trust combined with monitoring mechanisms and a feedback loop from the monitoring system to the trust evaluation would be a suitable approach for enterprise collaboration protection needs. Additionally, the drafted TuBE framework needs to be assessed as the system achieving this.

Thus in this section TuBE is evaluated using four different criteria: trust model robustness, trust management, enterprise protection and framework flexibility. Additionally, new threats and problems brought by the TuBE solution are discussed.

5.4.1 The TuBE Trust Model

Current trust models have been criticised for showing one or more of the following three weaknesses [MMH02]:

1. The relationship of trust and reputation is not clear: Differentiation of trust and reputation is either not made or the mechanism for inference between them is not explicit.
2. Trust and reputation are treated as independent of context or time: Trust and reputation are taken to be the same across multiple contexts, or are treated as uniform across time.
3. The sociological rooting is ignored: Despite the strong sociological foundation for the concepts of trust and reputation, existing computational models for them are often not grounded on understood social characteristics of these quantities.

However, the TuBE model avoids these pitfalls. First of all, it has been developed with the help of the trust type system analysed in Chapter 3.2 and the input factor analysis shown in Chapter 3.4, thus grounding the work to social structures, clear inference and difference between various concepts.

While in the TuBE trust model reputation is definitely a separate concept, an undelying factor affecting trust evaluation, this work has not been extended very far into the realm of reputation. This is a clear direction for future research.

Additionally, the factors that influence the trust valuation in TuBE include the situation i.e. context, and time dependency is taken into account with the behaviour monitoring and feedback to future trust evaluations.

Therefore the TuBE trust model can be considered robust and free from these weaknesses.

5.4.2 The TuBE Trust Management

In Chapter 3.6 a set of requirements for an enterprise class trust and mistrust management system were detailed. How does the TuBE system fulfil them?

1. It automates the translation of trusting intention to trusting behaviour. In the TuBE system this is done by using trust-guarded authorisation to decide whether the peer SOAP request can be permitted to proceed to the background application.

2. It presents a unified approach to trust management across applications. This is achieved in the TuBE system by leveraging Web Services technologies. By using the service specific wrapper as a key component, both native Web Service applications and legacy systems can be supported by the TuBE approach.

3. There should be locality of control in the authorisation decisions. This means that each organisation can decide themselves whom to trust and how. This is achieved in the TuBE system by trust-guarding the authorisation function in the organisation. While this enables local control and enterprise protection, in a wider community setting this is not a simple requirement. If a single member organisation in a community decides it does not trust a particular peer anymore and suddenly refuses to co-operate, the community does not work as intended. This is a problem that needs to be taken into account in the community level.

4. It is flexible in supporting complex trust relationships and multiple decision input factors that influence the trusting belief to a particular trustee. The default TuBE trust model consists of the 7-tuple of *trustor, trustee, action, action risk, action importance, trustee reputation, trustor context* and thus takes into account multiple factors that influence the trust evaluation. Conceptually it is easy to add a new factor to the model by just providing a new trust source module and trust calculation function.

5. The mechanisms for evaluating trust should be separated from the policies thus enabling reuse of the mechanisms across applications requiring different policies. In TuBE the trust calculation function and the authorisation threshold value for the wrapper are settable by policy. Also the use and effect of situational context can be set by the management. Thus the enterprise management can deploy the TuBE system in a variety of communities and circumstances.

6. It should introspectively monitor the peer behaviour to detect ongoing mistrust situations. This is achieved in the TuBE system by leveraging an application level intrusion detection system.

7. It must produce behavioural feedback to the trusting belief calculations to prevent further mistrust. In TuBE, the application level intrusion detection system is also linked to a reputation subsystem, so any detected problems lower the peer reputation accordingly. Likewise, positive experiences with a new peer increase the peer reputation.

All in all, the TuBE system is a well designed trust management system taking into account the four aspects of trust management and their requirements.

5.4.3 Impact as Enterprise Protection

Previously it was shown that TuBE is a good trust management solution from the trust model and management points of view. However, the real test criteria come from the collaborative enterprise environment and its protection needs as discussed in Chapter 2.8.2. Is trust management a good solution for the threat of malicious peers in an open dynamic community and how does the TuBE system in particular respond to the overall requirements of the solution?

1. The solution must work with an open community structure. Because of this requirement, TuBE utilises trust based authorisation, where the trust model includes provisions in the form of dispositional trust default trust value for new and unknown peers.

2. The solution must work in a dynamic environment without continuous manual intervention. This is achieved in TuBE with trust-guarded authorisation and settable policies. Trust-guarded authorisation uses trust evaluation as a basis for authorisation instead of an authorisation database which needs to be manually updated. Trust evaluations are dynamic and take into account the peer reputation and enterprise context when making the authorisation decision. The manual input in the TuBE system consists of setting up the trust support fact sources and their policies and inputting the trust calculation functions and threshold values for the wrapper.

3. The solution needs to protect an individual member of a community. This is achieved in the TuBE system by introspective monitoring in the form of application intrusion detection mechanism.

4. The solution should me minimally application specific. TuBE is not totally application or service independent, but has application specific elements. To increase the accuracy of monitoring, the wrapper is service specific. It is created from the WSDL description of the service the enterprise offers to the community. Additionally, the trust model requires the knowledge of actions and their risk and importance values.

Overall, the TuBE system responds to these challenges very well. The only deviation from the ideal comes in the discussion of application or service independence. As we observed in Chapter 4, there are mechanisms in the intrusion detection area that do not require application dependence. However, that would deny the TuBE monitoring component the capability of detecting behaviour anomalies, i.e. scenarios where the peer acts within syntactically legal limits but with semantically anomalous content. In the TuBE system it was decided to explore monitoring systems capable of detecting these problems aw well. Any kind of semantic awareness in the monitoring system requires application dependence. Keeping this requirement in mind the TuBE system was designed to be application specific as little as possible. The only application specific software component is the wrapper. The other components are not bound to any single service or application other than by configuration. For example, the action names and values need to be configured to the action data source. Similarly, the semantic knowledge about the application needs to be configured to the intrusion detection monitoring component if semantic anomaly detection is used.

5.4.4 TuBE as a Framework

The TuBE framework is a reusable design of a system enabling Web Services to use trust-guarded authorisation. For a community framework to be truly practical, it should be flexible, extensible and allow its use in many different scenarios and deployment configurations.

The TuBE framework can be utilised in different configurations. In the simplest case it possible to protect one single enterprise service and service instance with the TuBE framework by specialising and deploying a service specific wrapper instance from a general wrapper class.

The publish/subscribe paradigm also enables to deploy the TuBE in a scenario where there are several same service instances just by deploying several wrapper instances. Additionally, the publish/subscribe mechanism makes it is possible to use several different intrusion detection mechanisms to monitor the web service.

It should also be possible to extend the TuBE framework to utilise other trust models as well. This requires first of all the ability to develop new support data components from existing base classes, secondly the possibility to define new APIs in the trust broker and thirdly the possibility of defining the effect of this new data on the trusting intention calculation in the wrapper.

Implementing all these framework flexibility options in the actual TuBE framework is a challenge and provides much fertile ground for practical research questions and answers.

5.4.5 New Threats

While the TuBE system is intended to solve the previously discussed issue of cost-effective management of secure enterprise collaboration, there may be new threats that the newly designed system introduces to the collaborative environment. Every time a system or framework is changed, it is necessary to analyse the changed situation if there are new threats or problems that the change has caused. Typically new vulnerabilities arise just because something as been changed and its impact has not been fully analysed. What threats to the does the introduction of TuBE bring to the general threat analysis performed earlier in Chapter 2.6? Here the threats are discussed both from the security point of view as done previously but also from the viewpoint of deployment showstoppers. What threats are there in the designed TuBE system that would prevent it to be deployed at an organisation?

One key issue is of course the increased complexity and the security threats associated with it. TuBE brings more components that can fail and cause a denial of service situation. This is an issue needing addressing during TuBE development, the system needs to be both robust enough to survive in the proposed environment and fault-tolerant enough so that one faulty component does not cause a total denial of service. Also the system needs to fail securely, i.e. it should be analysed during development what is the most secure way to handle observed failures of various components.

However, the complexity is present only to the organisation deploying the TuBE framework. Deploying TuBE does not increase the number of interfaces visible out of the enterprise so the attack surface presented to external parties or community members is not increased by the TuBE deployment. If an external reputation system is attached to the trust support layer, it will add one external interface, but this is not the default.

There are some deployment threats also caused by the additional complexity. The first deployment threat arises from the fact that the authorisation decision is dependent on many underlying trust factors and it may not be immediately clear to observers why a particular request has been granted or denied. This opaqueness may cause some additional workload to system management especially in a problem solving situation. Nevertheless, the workload should be much smaller than managing individual access rights one by one in a dynamically changing community. This threat could be at least partially solved by providing good management tools for the TuBE administrators.

Additionally, the complexity of the calculations may cause the system deployment phase to be somewhat longer than in a standard non-trust-guarded application deployment because the context parameters, action risk and importance values and the trust threshold should be fine-tuned to the required level of security. In this the TuBE system does not really differ from other security solutions such as firewalls and IDS/IPS systems, which also need an initial fine-tuning period.

The TuBE system uses a model where trust is based on previous actions, i.e. behaviour based reputation. Reputation based systems are theoretically vulnerable to a Sybil attack where one malicious party creates several false identities and uses those identities to influence reputation scores. However, this particular version of the Sybil attack is not a threat in TuBe because of the way the

default TuBE system uses reputation information. The reputation scores in TuBE are calculated based on direct experience with the particular identity, they are not by default influenced by third parties with potentially false identities. But if the reputation subsystem in TuBE is connected to an external reputation system, where the reputation score is somehow formed by collective opinion, this may become a real threat if the collective opinion can be influenced by false identities created in a Sybil attack.

A new security threat to the offered service is brought by the reaction possibilities of the monitoring component. If the monitoring system has too many false positive alerts that cause it to cause a denial of service situation with a particular client or connecting member, it may mean lost revenue to the TuBE using member. The monitoring/IPS component should be fine-tuned so that the amount of false positives is sufficiently low not to cause major financial impact. This threat is not only a security threat, it will also directly impact organisations' interest in using a TuBE-like framework.

There is also the threat of service performance degradation due to the additional processing in the TuBe framework. When every service request has to wait for the trust-score based authorisation decision, it is possible that the performance of the service interface is not sufficient. It is foreseeable that especially performance problems would be real showstopper threats to the trustguard system adoption. Therefore this threat must be taken into account in the TuBE development phase so that the performance loss introduced by TuBE is negligible. The effect of various monitoring algorithms and trust calculation methods to performance would be good future research questions.

The TuBE system may also cause a threat to the overall functionality and performance of the community. It is likely that adopting a TuBE system at some members increases choreography problems stemming from trust-based authorisation denials. The community choreography engine has to manage this type of events anyway, so this is not really a new task but the event frequency may rise and thus the community throughput may be lower. This is a separate performance issue from the performance of a single member's service and should be separately investigated.

Whether these new threats are sufficiently severe issues preventing TuBE to be a successful system and framework can be judged when a prototype is built and real test results are available.

Chapter 6

Conclusion and Future Directions

The modern business environment can be characterised by rapid change. New innovations in products, services and also business models are continuously emerging and enterprises need to adapt to these changes to stay in business. This is driving enterprises to seek agility and adaptability from networked business collaboration and dynamic business decisions. Information technology can be both an enabler of these business networks and also a driver for pushing the change to even wider corporate audience.

Therefore, concepts like virtual enterprises, business networks and service oriented computing have been both in research and rapid development focus in recent years. Business adaptability requirements have brought increasing needs for managing and securing the dynamic business relationships in the information technology systems. The current solutions relying on manually managed static access and authorisation rules are not flexible enough nor cost-effective. Thus the infrastructure supporting future business networks must have better methods of managing the collaboration risks and securing the participants. Otherwise the promise of information technology as a business network enabler will not be fulfilled. Insecure business network middleware requiring lot of manual configuration and adjustments may hinder or even prevent the widespread adoption of busness networks.

Thus, the key research question for this thesis was how to secure enterprise applications in an open dynamic business network community without sacrificing dynamism and scalability, especially from the point of view of the community member organisation. The hypothesis presented in this thesis that we can draw some ideas from the human experience and apply the concept of trust to enhance community security especially from the community member point of view.

To fulfill this goal, this thesis presents a novel community security framework having three separate elements: an automated trust evaluation process, continuous peer behaviour monitoring, and constant feedback from the monitoring system to trust evaluation for readjustments.

To successfully manage relationships between enterprise agents using trust, the management framework needs to be well grounded into the nature of trust. This thesis shows such a theoretical base and one of the key contributions is to show how the theoretical concept of trusting behaviour can be automated in a computer system as trust-guarded authorisation. The developed concept is well grounded in trust theory and independent of the chosen trust model. This generality enables trust decision automation which in turn enables enterprises to dynamically allow and authorise other entities to use their services and resources for mutual benefit.

As a step towards the key goal several existing trust models were analysed. Based on this analysis a partial domain ontology was built. This step accomplished a common vocabulary for describing facts to be considered for trust calculation. This new vocabulary facilitates communication when models use different terminology for similar concepts.

Using trust to authorise actions brings in its wake several risks: misplaced trust and the risk of loss. As trust theory shows, it is not sufficient just to trust, we must also dynamically adjust our trust valuation according to the observed peer behaviour so that misplaced trust, i.e. mistrust can be detected. Therefore, a new view of trust management was developed, which consists of trust initialisation, trust use, monitoring and reaction to the observations. An enterprise using trust must thus introspectively monitor the collaborations for its own protection. This thesis surveyed various possibilities for the peer behaviour observation method in the field of application level intrusion detection.

These two theoretically separate ideas, trust-guarded authorisations and introspective monitoring in the form of intrusion detection, have in this thesis been brought together as an answer for the key research question. A novel community security framework, TuBE, has been drafted as a solution framework. From a theoretical point of view, TuBE confirms our hypothesis that trust can be applied in securing business network members from some key risks in the collaboration.

The main contributions of this work so far are as follows. First of all, it has been shown that trust theory brings forth trust-guarded authorisation as a suitable method for guarding access to computer systems. This has the benefit of enabling automated and dynamic authorisation decisions, which are required in business networking. Second, it was shown how these trust-guarded systems can manage misplaced trust by continuous introspective monitoring and dynamically adjusting the trust evaluation based on the monitoring results. Third, there are several interesting tools in the application intrusion detection arena that can be used for introspective monitoring also in a community setting. Finally, it was shown how it is possible to combine these ideas to a coherent framework capable of fulfilling community member security requirements.

Thus, the overall conclusion of this thesis is that enterprises taking part in business networks can secure themselves from key business networking threats by using collaborative middleware combining trust-guarded authorisation and misuse monitoring as security features. This gives enterprises methods and in the future some tools to secure themselves in the collaboration, thus enabling more widespread business networking. This in turn enables businesses to survive in the current rapidly changing business environment.

6.1 Technical Research Questions for Implementation Phase

The TuBE framework presented in this thesis is at this point a draft design and its success and performance needs to be evaluated with a working implementation. From this design several interesting technical research questions are found for the implementation phase:

1. General key question: How well does this concept actually work from different viewpoints such as security, dynamic management and service performance? Does the implementation confirm our hypothesis of trust-guarded authorisation and introspective monitoring being suitable for securing enterprise collaboration?

2. Monitoring questions: What are the best algorithms for the introspective monitoring functions in this scenario? Does utilising several different monitors and thus algorithms benefit the monitoring function?

3. Reputation questions: What kind of reputation mechanism would suit this scenario? How would the use of several monitoring mechanisms giving potentially conflicting behaviour scores to the same transaction be handled in a reputation mechanism?

4. Trust model questions: How can the trust broker and underlying trust data sources be built so that the TuBE framework could easily support changes in the trust model and underlying

data sources. To continue this further, do changes in the trust model change the performance of the framework or the level of protection to the service?

6.2 Future Research Directions in Trust and Collaborative Communities

In addition to the implementation questions above, during this work several new research avenues have been found but not investigated due to lack of resources.

Generally, research on trust can be divided to three groups based on its context. On the lowest, most fundamental level, trust is a part of the infrastructure. Early trust research has been concentrating on this level. On top of it are the service and community levels. As electronic commerce has gained a foothold and open systems become more common, trust forms an important part on the service level. There are still many problems to be solved on this level before research on the highest level, the community, can proceed freely [RK05].

As discussed briefly in Chapter 3.5.2, our test scenario for the TuBE prototype system is based on these lower technical and service levels and trust-guarding the service action authorisation process. However, there are several community level issues that the TuBE concept could influence but currently does not support.

For example, the trusting behaviour discussed in trust theory could in practise be any action in any part of the community operational life cycle. The focus of this work has been in the operational phase and monitoring. However, there are several interesting trust questions elsewhere that may warrant further research. For example, how could an enterprise utilise the trusting intention and trusting beliefs in community population when selecting the participants for the community roles? Or how does the trust valuation affect our behaviour in the contract negotiation or community establishment phase? Or what about our reactions to community structure change requests, should change requests be accepted more readily if we trust the request initiator? Additionally, some thought should be given to various systems trust issues. For example, what aspects in electronic contracting processes would increase or decrease the systems trust?

It may be possible to build these community support services on top of the same TuBE framework that is also trust-guarding the operations phase for the member. This would require that we have two sets of actions, the operation phase actions described for example in the application WSDL file, and the community level actions that are described in the community operational contract or an orchestration specification. Thus the TuBE monitoring function would also need extension to the higher levels in the collaborative community stack. There are quite many monitoring challenges as mentioned in Chapter 2.7.

The actual feasibility and mechanisms of this community lifecycle support need to be described in a separate research paper, but this area is a direct continuation of the current research and should be investigated further.

A second interesting set of future research questions can be found in the general larger areas of community security and monitoring in electronic communities. What aspects of the community interaction and behaviour can be monitored, how, by whom and how should the results affect the community interaction. The TuBE system introduced here is a solution for a smaller focused problem of community member security, but as observed previously, there are several different monitoring task in a community as a whole. Some of these monitoring tasks could be achieved by linking the TuBE instances at every participant together to help in the community monitoring. But this can be researched, as many of the higher level community issues discussed above, when at least some of the lower community layer functions have been implemented and tested. Thus

TuBE is a small stepping stone to forming a more complete security and protection framework for electronic communities in the future.

The third group of interesting issues can be found from the general threat analysis performed in Chapter 2.6. The work in this thesis concentrated on just one group of threats, malicious members and external parties in a dynamically changing environment, there were many more threats needing good solutions. For example, the impact and management of a Sybil attack in different kinds of business communities is an interesting question. So is the question of how to ensure confidentiality in a community environment where some middleware components are shared between possibly competing communities.

Bibliography

[ABB+05] Anderson, S., Bohren, J., Boubez, T., Chanilau, M. and Della-Libera, G., Specification: Web services trust language (WS-Trust), 2005. URL http://www-128.ibm.com/developerworks/library/specification/ws-trust/.

[AL01] Almgren, M. and Lindqvist, U., Application-integrated data collection for security monitoring. *RAID'01: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, volume 2212 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001, pages 22–36, URL http://springerlink.metapress.com/app/home/content.asp?wasp=5n5d6ynmwg1xwvf1bd9l&referrer=contribution&format=2&page=1&pagecount=0.

[Amo94] Amoroso, E. G., *Fundamentals of Computer Security Technology*. Prentice-Hall International, Inc., 1994.

[And01] Anderson, R., *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley & Sons Inc, 2001.

[And04] Anderson, D. P., BOINC: A system for public-resource computing and storage. *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, Washington, DC, USA, 2004, IEEE Computer Society, pages 4–10.

[ARH97] Abdul-Rahman, A. and Hailes, S., A distributed trust model. *NSPW '97: Proceedings of the 1997 New Security Paradigms Workshop*. ACM, 1997, pages 48–60, URL http://citeseer.ist.psu.edu/abdul-rahman97distributed.html.

[ATS04] Androutsellis-Theotokis, S. and Spinellis, D., A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36,4(2004), pages 335–371.

[Axe00] Axelsson, S., Intrusion-detection systems: A survey and taxonomy. Technical report no 99-15, Dept. of Computer Engineering, Chalmers University of Technology, Sweden, 2000.

[BCS+00] Bowen, T., Chee, D., Segal, M., Sekar, R., Shanbhag, T. and Uppuluri, P., Building survivable systems: An integrated approach based on intrusion detection and damage containment. *DISCEX'00: Proceedings of the DARPA Information Survivability Conference and Exposition*, volume 2. IEEE Computer Society, 2000, pages 84–99.

[BFK99] Blaze, M., Feigenbaum, J. and Keromytis, A. D., KeyNote: Trust management for public-key infrastructures (position paper). *Proceedings of the 6th International Workshop on Security Protocols*, volume 1550 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999, pages 59–63, URL http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=1550&spage=59.

[BFL96] Blaze, M., Feigenbaum, J. and Lacy, J., Decentralized trust management. *Proceedings of the 1996 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1996, pages 164–173, URL http://ieeexplore.ieee.org/iel3/3742/10940/00502679.pdf.

[BHM+04] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. and Orchard, D., Web services architecture, 2004. URL http://www.w3.org/TR/ws-arch/. W3C Working Group Note 11.

[BIM+03] BEA, IBM, Microsoft, SAP and Siebel, Business process execution language for web services version 1.1, 2003. URL http://www.ibm.com/developerworks/webservices/library/ws-bpel/. Specification.

[Bis03] Bishop, M., *Computer Security: Art and Science*. Addison-Wesley, 2003.

[BZ02] Bhargava, B. and Zhong, Y., Authorization based on evidence and trust. *DaWaK '02: Proceedings of the 4th International Conference of Data Warehousing and Knowledge Discovery*, volume 2454 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002, pages 94–103, URL http://citeseer.ist.psu.edu/bhargava02authorization.html.

[CFL+97] Chu, Y.-H., Feigenbaum, J., LaMacchia, B., Resnick, P. and Strauss, M., Referee: Trust management for web applications. *Computer Networks and ISDN Systems*, 29,8–13(1997), pages 953–964. URL http://citeseer.ist.psu.edu/58910.html.

[CGS+03] Cahill, V., Gray, E., Seigneur, J.-M., Jensen, C., Chen, Y., Shand, B., Dimmock, N., Twigg, A., Bacon, J., English, C., Wagealla, W., Terzis, S., Nixon, P., Serugendo, G. D. M., Bryce, C., Carbone, M., Krukow, K. and Nielsen, M., Using trust for secure collaboration in uncertain environments. *IEEE Pervasive Computing*, 2,3(2003), pages 52–61. URL http://ieeexplore.ieee.org/iel5/7756/27556/01228527.pdf.

[CLM01] Cabrera, J. B. D., Lewis, L., and Mehra, R. K., Detection and classification of intrusions and faults using sequences of system calls. *SIGMOD Record*, 30,4(2001), pages 25–34. URL http://delivery.acm.org/10.1145/610000/604269/p25-cabrera.pdf?key1=604269&key2=4040363011&coll=GUIDE&dl=ACM&CFID=33931896&CFTOKEN=21280247.

[CMA01] Camarinha-Matos, L. M. and Afsarmanesh, H., Virtual enterprise modeling and support infrastructures: Applying multi-agent system approaches. *WASSS'01:*

Selected Tutorials from Multi-Agent Systems and Applications : 9th ECCAI Advanced Course ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School, volume 2086 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001, pages 335–364.

[CMA05] Camarinha-Matos, L. M. and Afsarmanesh, H., Collaborative networks: A new scientific discipline. *Journal of Intelligent Manufacturing*, 16,4-5(2005), pages 439–452.

[CNS03] Carbone, M., Nielsen, M. and Sassone, V., A formal model for trust in dynamic networks. *SEFM'03: Proceedings of the 1st International Conference on Software Engineering and Formal Methods*. IEEE Computer Society, 2003, pages 54–61, URL `http://ieeexplore.ieee.org/iel5/8747/27706/01236207.pdf`.

[Coh08] Cohen, B., Bittorrent protocol specification, 2008. URL `http://bittorrent.org/beps/bep_0003.html`. Version 11031.

[CSW06] Chapin, P., Skalka, C. and Wang, X. S., Trust management: Features and foundations. Submitted for publication, 2006.

[CY00] Chen, R. and Yeager, W., Poblano a distributed trust model for peer-to-peer networks. Technical paper, Sun Microsystems, 2000. URL `http://www.jxta.org/docs/trust.pdf`.

[DDCdVP+02] Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati, P. and Violante, F., A reputation-based approach for choosing reliable resources in peer-to-peer networks. *CCS '02: Proceedings of the 9th ACM conference on Computer and Communications Security*, New York, NY, USA, 2002, ACM, pages 207–216.

[Dem04] Demolombe, R., Reasoning about trust: A formal logical framework. *iTrust'04: Proceedings of the 2nd International Conference on Trust Management*, volume 2995 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004, pages 291–303, URL `http://www.springerlink.com/media/3N259B5UXH0QTVE2CB9Q/Contributions/Y/A/L/Y/YALYRU2BRPQ4FQ2U.pdf`.

[Deu89] Deutsch, L. P., Design reuse and frameworks in the Smalltalk-80 programming system. In *Software Reusability, Vol. II*, Addison-Wesley, 1989, pages 55–71.

[Din00] Dingledine, R., The Free Haven project: Design and deployment of an anonymous secure data haven. Master's thesis, MIT, 2000. URL `http://theory.lcs.mit.edu/~~cis/theses/dingledine-masters.ps`.

[DJPJ04] Desmet, L., Jacobs, B., Piessens, F. and Joosen, W., Threat modelling for web services based web applications. *CMS'04: Proceedings of the 8th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security*, volume 175 of *IFIP International Federation for Information Processing*, 2004, pages 161–174.

[Dou02] Douceur, J. R., The Sybil attack. *IPTPS'02: Revised Papers of First International Workshop of Peer-To-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, Cambridge, MA, USA, 2002, Springer, pages 251–260.

[EFGK03] Eugster, P., Felber, P., Guerraoui, R. and Kermarrec, A., The many faces of publish/subscribe. *ACM Computing Surveys*, 35,2(2003).

[EM99] Elbaum, S. and Munson, J. C., Intrusion detection through dynamic software measurement. *ID '99: Proceedings of the 1st Workshop on Intrusion Detection and Network Monitoring*. USENIX Association, 1999, URL `http://www.usenix.org/events/detection99/full_papers/elbaum/elbaum.pdf`.

[End98] Endler, D., Intrusion detection. applying machine learning to Solaris audit data. *ACSAC'98: Proceedings of the 14th Annual Computer Security Applications Conference*. IEEE Computer Society, 1998, pages 268–279, URL `http://ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=738647`.

[Ess98] Essin, D. J., Patterns of trust and policy. *NSPW'97: Proceedings of the 1997 New Security Paradigms Workshop*. ACM Press, 1998, pages 38–47, URL `http://doi.acm.org/10.1145/283699.283738`.

[Eur07] European Network and Information Security Agency, *Emerging-risks-related Information Collection and Dissemination*, 2007.

[FBF99] Fraser, T., Badger, L. and Feldman, M., Hardening COTS software with generic software wrappers. *Proceedings of the 1999 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1999, pages 2–16.

[FHS97] Forrest, S., Hofmeyr, S. A. and Somayaji, A., Computer immunology. *Communications of the ACM*, 40,10(1997), pages 88–96. URL `http://portal.acm.org/citation.cfm?id=262793.262811`.

[FHSL96] Forrest, S., Hofmeyr, S. A., Somayaji, A. and Longstaff, T. A., A sense of self for Unix processes. *Proceedings of the 1996 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1996, pages 120–128.

[FKF+03] Feng, H., Koleshnikov, O., Fogla, P., Lee, W. and Gong, W., Anomaly detection using call stack information. *Proceedings of the 2003 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2003, pages 62–75.

[GAHL00] Grefen, P., Aberer, K., Hoffner, Y. and Ludwig, H., CrossFlow: Cross-organizational workflow management in dynamic virtual enterprises. *International Journal of Computer System Sciences and Engineering*, 15,5(2000), pages 277–290.

[Gam88] Gambetta, D., Can we trust trust? In *Trust: Making and Breaking Cooperative Relations*, Basil Blackwell, 1988, pages 213–237, URL `http://www.sociology.ox.ac.uk/papers/gambetta213-237.pdf`. Reprinted in electronic edition from Department of Sociology, University of Oxford, chapter 13.

[GJM02] Giffin, J. T., Jha, S. and Miller, B. P., Detecting manipulated remote call streams. *Proceedings of the 11th USENIX Security Symposium*. USENIX Association, 2002, pages 61–79.

[GMS00] Ghosh, A. K., Michael, C. and Schatz, M., A real-time intrusion detection system based on learning program behavior. *RAID'00: Proceedings of the 3rd International Symposium on Recent Advances in Intrusion Detection*, volume 1907 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000, pages 93–109.

[GMT04] Grimsley, M., Meehan, A. and Tan, A., Managing internet-mediated community trust relations. *iTrust'04: Proceedings of the 2nd International Conference on Trust Management*, volume 2995 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004, pages 277–290, URL `http://www.springerlink.com/openurl.asp?genre=article&id=49K9QP1XUDJGLLD6`.

[GPH03] Golbeck, J., Parsia, B. and Hendler, J., Trust networks on the semantic web. *CIA VII: Proceedings of the 7th International Workshop on Cooperative Intelligent Agents*, volume 2782 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003, pages 238–249, URL `http://www.mindswap.org/papers/CIA03.pdf`.

[GR99] Gunetti, D. and Ruffo, G., Intrusion detection through behavioral data. *Proceedings of the 3rd International Symposium in Advances in Intelligent Data Analysis*, volume 1642 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999, pages 383–394, URL `http://www.springerlink.com/app/home/contribution.asp?wasp=g19tay35wlcyyn9c0vtp&referrer=parent&backto=issue,32,44;journal,1577,1828;linkingpublicationresults,1:105633,1`.

[GS02] Grandison, T. and Sloman, M., Specifying and analysing trust for internet applications. *I3E'02: Towards The Knowledge Society: The 2nd IFIP Conference on E-Commerce, E-Business, E-Government*, volume 233 of *IFIP Conference Proceedings*. Kluwer, 2002, URL `http://citeseer.ist.psu.edu/grandison02specifying.html`.

[GS03] Grandison, T. and Sloman, M., Trust management tools for internet applications. *iTrust'03: Proceedings of the 1st International Conference on Trust Management*, volume 2692 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003, pages 91–107.

[GSS99] Ghosh, A. K., Schwartzbard, A. and Schatz, M., Learning program behavior profiles for intrusion detection. *Proceedings of the 1st Workshop on Intrusion Detection and Network Monitoring*. USENIX Association, 1999, pages 51–62, URL `http://www.usenix.org/publications/library/proceedings/detection99/full_papers/ghosh/ghosh_html/`.

[GTM99] Glushko, R. J., Tenenbaum, J. M. and Meltzer, B., An XML framework for agent-based e-commerce. *Communications of the ACM*, 42,3(1999), pages 106–114.

[Haa05] Haataja, J.-P., Liiketoimintaverkostoja hallinnoiva väliohjelmisto ja automatisoitu valvontafunktio. Master's thesis, University of Helsinki, Department of Computer Science, 2005.

[HBS+02] Hapner, M., Burridge, R., Sharma, R., Fialli, J. and Stout, K., Java Message Service, 2002. URL http://java.sun.com/products/jms/docs.html. Version 1.1.

[HFS98] Hofmeyr, S. A., Forrest, S. and Somayaji, A., Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6,3(1998), pages 151–180.

[HK01] Herrmann, P. and Krumm, H., Trust-adapted enforcement of security policies in distributed component-structured applications. *ISCC'01: Proceedings of the 6th IEEE Symposium on Computers and Communications*. IEEE Computer Society, 2001, pages 2–8, URL http://ls4-www.cs.uni-dortmund.de/RVS/Pub/TS/ISCC01.pdf.

[HL03] Howard, M. and LeBlanc, D., *Writing Secure Code, 2nd edition*. Microsoft Press, 2003.

[HS05] Huhns, M. N. and Singh, M. P., Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9,1(2005), pages 75–81.

[IF02] Inoue, H. and Forrest, S., Anomaly intrusion detection in dynamic execution environments. *NSPW'02: Proceedings of the 2002 New Security Paradigms Workshop*. ACM Press, 2002, pages 52–60, URL http://portal.acm.org/ft_gateway.cfm?id=844112&type=pdf&coll=portal&dl=ACM&CFID=19905017&CFTOKEN=31765782.

[ISO96a] ISO/IEC IS 10746-2, Open distributed processing reference model, part 2, foundations, 1996.

[ISO96b] ISO/IEC IS 10746-3, Open distributed processing reference model, part 3, architecture, 1996.

[JL01] Jones, A. and Lin, Y., Application intrusion detection using language library calls. *ACSAC'01: Proceedings of the 17th Annual Computer Security Applications Conference*. IEEE Computer Society, 2001, pages 442–449, URL http://ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=991561.

[Jøs96] Jøsang, A., The right type of trust for distributed systems. *NSPW'96: Proceedings of the 1996 ACM Workshop on New Security Paradigms*. ACM, 1996, pages 119–131, URL http://security.dstc.edu.au/staff/ajosang/papers/trdsyst.ps.

[Jøs98] Jøsang, A., *Modeling Trust in Information Security*. Ph.D. thesis, Norwegian University of Science and Technology, Faculty of Electrical Engineering and Telecommunications, Trondheim, Norway, 1998.

[KFBK00] Ko, C., Fraser, T., Badger, L. and Kilpatrick, D., Detecting and countering system intrusions using software wrappers. *Proceedings of the 9th USENIX Security Symposium*. USENIX Association, 2000, URL http://ieeexplore.ieee.org/iel5/8932/28288/01264947.pdf.

[KFL94] Ko, C., Fink, G. and Levitt, K., Automated detection of vulnerabilities in privileged programs by execution monitoring. *ACSAC'94: Proceedings of the*

10th Annual Computer Security Application Conference. IEEE Computer Society, 1994, pages 134–144, URL http://seclab.cs.ucdavis.edu/papers/pdfs/ck-gf-kl-94.pdf.

[KH97] Kosoresow, A. P. and Hofmeyr, S. A., Intrusion detection via system call traces. *IEEE Software*, 14,5(1997), pages 35–42.

[KM05] Kutvonen, L. and Metso, J., Services, contracts, policies and ecommunities – relationship to ODP framework. *WODPEC'05: Workshop on ODP for Enterprise Computing*, 2005, pages 62–69, URL http://www.lcc.uma.es/~av/wodpec2005/wodpec2005-Proceedings.pdf.

[KMR06] Kutvonen, L., Metso, J. and Ruohomaa, S., From trading to ecommunity population: Responding to social and contractual challenges. *EDOC'06: Proceedings of the 10th International Enterprise Distributed Object Computing Conference*. IEEE Computer Society, 2006, pages 199–210.

[KRL97] Ko, C., Ruschitzka, M. and Levitt, K., Execution monitoring of security-critical programs in distributed systems: A specification based approach. *Proceedings of the 1997 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1997, pages 175–187, URL http://portal.acm.org/citation.cfm?id=884386.

[KRM07] Kutvonen, L., Ruokolainen, T. and Metso, J., Interoperability middleware for federated business services in web-Pilarcos. *International Journal of Enterprise Information Systems, Special issue of Interoperability of Enterprise Systems and Applications*, 3,1(2007), pages 1–21.

[KRRM08] Kutvonen, L., Ruokolainen, T., Ruohomaa, S. and Metso, J., Service-oriented middleware for managing inter-enterprise collaborations. In *Global Implications of Modern Enterprise Information Systems: Technologies and Applications*, Gunasekaran, A., editor, IGI Global, October 2008, pages 208–241.

[KS99] Kuperman, B. A. and Spafford, E., Generation of application level audit data via library interposition. Technical Report CERIAS TR 1999-11, COAST Laboratory, 1999. URL https://www.cerias.purdue.edu/tools_and_resources/bibtex_archive/archive/99-11.pdf.

[KT93] Krause, M. and Tipton, H. F., *Handbook of Information Security Management*. Auerbach Publications, 1993.

[Kut02] Kutvonen, L., Automated management of inter-organisational applications. *EDOC'02: Proceedings of the 6th International Enterprise Distributed Object Computing Conference*. IEEE Computer Society, 2002, pages 28–39, URL http://computer.org/proceedings/edoc/1742/17420027abs.htm.

[LASS00] Lazcano, A., Alonso, G., Schuldt, H. and Schuler, C., The WISE approach to electronic commerce. *International Journal of Computer System Sciences and Engineering*, 15,5(2000), pages 345–357.

[LB97] Lane, T. and Brodley, C. E., Sequence matching and learning in anomaly detection for computer security. *AAAI-97: Proceedings of Workshop on AI Approaches to Fraud Detection and Risk Management*. AAAI Press, 1997, pages 43–49, URL http://citeseer.ist.psu.edu/lane97sequence.html.

[LS98] Lee, W. and Stolfo, S., Data mining approaches for intrusion detection. *Proceedings of the 7th USENIX Security Symposium*. USENIX Association, 1998, pages 79–94, URL http://citeseer.ist.psu.edu/article/lee98data.html.

[LSC97] Lee, W., Stolfo, S. J. and Chan, P. K., Learning patterns from Unix process execution traces for intrusion detection. *AAAI-97: Proceedings of Workshop on AI Approaches to Fraud Detection and Risk Management*. AAAI Press, 1997, pages 50–56, URL http://www1.cs.columbia.edu/ids/publications/wenke-aaai97.ps.

[Mar94] Marsh, S., *Formalising Trust as a Computational Concept*. Ph.D. thesis, Department of Mathematics and Computer Science, University of Stirling, 1994. URL http://citeseer.ist.psu.edu/marsh94formalising.html.

[Mar00] Marceau, C., Characterizing the behavior of a program using multiple-length N-grams. *NSPW'00: Proceedings of the 2000 New Security Paradigms Workshop*. ACM Press, 2000, pages 101–110, URL http://delivery.acm.org/10.1145/370000/366197/p101-marceau.pdf?key1=366197&key2=8276292011&coll=GUIDE&dl=ACM&CFID=33734302&CFTOKEN=20018340.

[MC96] McKnight, D. H. and Chervany, N. L., The meanings of trust. Technical Report, University of Minnesota, MIS Research Center, 1996. URL http://misrc.umn.edu/workingpapers/fullPapers/1996/9604_040100.pdf. Tables included as attachments in the online version.

[MD05] Marsh, S. and Dibben, M. R., Trust, untrust, distrust and mistrust—an exploration of the dark(er) side. *iTrust'05: Proceedings of the 3rd International Conference on Trust Management*, volume 3477 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005, pages 17–33, URL http://www.springerlink.com/link.asp?id=hlvhnt431yte3lay.

[MG98] Meltzer, B. and Glushko, R., XML and electronic commerce: Enabling the network economy. *SIGMOD Record*, 27,4(1998).

[MG02] Michael, C. C. and Ghosh, A., Simple, state-based approaches to program-based anomaly detection. *TISSEC: ACM Transactions on Information and System Security*, 5,3(2002), pages 203–237. URL http://portal.acm.org/ft_gateway.cfm%3Fid%3D545187%26type%3Dpdf%26dl%3DGUIDE%26dl%3DACM%26CFID%3D11111111%26CFTOKEN%3D2222222.

[MJDP02] Milosevic, Z., Jøsang, A., Dimitrakos, T. and Patton, M. A., Discretionary enforcement of electronic contracts. *EDOC'02: Proceedings of the 6th Internationa Enterprise Disctributed Computing Conference*. IEEE Computer Society, 2002, pages 39–51.

[MLG+04] Milosevic, Z., Linington, P. F., Gibson, S., Kulkarni, S. and Cole, J., Inter-organisational collaborations supported by e-Contracts. *Building the E-Service Society: E-Commerce, E-Business, and E-Government*, volume 9 of 14 of *IFIP 18th World Computer Congress*, Toulouse, France, August 2004, IFIP, Kluwer, pages 413–429, URL http://www.cs.kent.ac.uk/pubs/2004/1982.

[MMH02] Mui, L., Mohtashemi, M. and Halberstadt, A., A computational model of trust and reputation for e-businesses. *HICSS'02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, volume 7. IEEE Computer Society, 2002, URL http://csdl.computer.org/comp/proceedings/hicss/2002/1435/07/14350188.pdf.

[OAS06] OASIS, Web services base notification 1.3 (ws-basenotification), 2006. URL http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf.

[OMG04] OMG, OWL full and UML 2.0 compared, 2004. URL http://www.omg.org/docs/ontology/04-03-01.pdf.

[Pap03] Papazoglou, M., Service oriented computing: Concepts, characteristics and directions. *WISE'03: Proceedings of the Fourth International Conference on Web Information Systems Engineering*. IEEE Computer Society, 2003, pages 3–12.

[PG03] Papazoglou, M. and Georgakopoulos, D., Service oriented computing. *Communications of the ACM*, 46,10(2003), pages 25–28.

[Pos82] Postel, J. B., Simple mail transfer protocol, 1982. RFC 821.

[Pro03] Provos, N., Improving host security with system call policies. *Proceedings of the 12th USENIX Security Symposium*. USENIX Association, 2003, pages 257–272, URL http://niels.xtdnet.nl/papers/systrace.pdf.

[RD00] Reichert, M. and Dadam, P., ADEPTFlex - supporting dynamic changes of workflow without losing control. *Journal of Intelligent Information Systems, Special Issue on Workflow Management*, 10,2(2000), pages 93–129.

[RK05] Ruohomaa, S. and Kutvonen, L., Trust management survey. *iTrust'05: Proceedings of the 3rd International Conference on Trust Management*, volume 3477 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005, pages 77–92, URL http://dx.doi.org/10.1007/11429760_6.

[RK06] Ruokolainen, T. and Kutvonen, L., Interoperability in service-based communities. *Business Process Management Workshops: BPM 2005 International Workshops, BPI, BPD, ENEI, BPRM, WSCOBPM, BPS*, Bussler, C. and Haller, A., editors, volume 3812 of *Lecture Notes in Computer Science*. Springer-Verlag, 2006, pages 317–328, URL http://dx.doi.org/10.1007/11678564_28.

[RK10] Ruohomaa, S. and Kutvonen, L., Trust and distrust in adaptive inter-enterprise collaboration management. *Journal of Theoretical and Applied Electronic Commerce Research, Special Issue on Trust and Trust Management*, 5,2(2010), pages 118–136. URL http://www.jtaer.com/aug2010/ruohomaa_kutvonen_p7.pdf.

[RKS02] Raz, O., Koopman, P. and Shaw, M., Semantic anomaly detection in online data sources. *ICSE'02: Proceedings of the 24th International Conference on Software Engineering*. IEEE Computer Society, 2002, pages 302–312, URL `http://pag.csail.mit.edu/daikon/pubs-using/raz-icse-2002-abstract.html`.

[SBDB01] Sekar, R., Bendre, M., Dhurjati, D. and Bollineni, P., A fast automaton-based method for detecting anomalous program behaviors. *Proceedings of the 2001 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2001, pages 144–155, URL `http://citeseer.ist.psu.edu/sekar01fast.html`.

[SBS99] Sekar, R., Bowen, T. and Segal, M., On preventing intrusions by process behavior monitoring. *ID'99: Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring*. USENIX Association, 1999, pages 29–40.

[SF00] Somayaji, A. and Forrest, S., Automated response using system-call delays. *Proceedings of the 9th USENIX Security Symposium*. USENIX Association, 2000, pages 185–197.

[SGF+02] Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H. and Zhou, S., Specification-based anomaly detection: a new approach for detecting network intrusions. *CCS'02: Proceedings of the 9th ACM conference on Computer and Communications Security*. ACM, 2002, pages 265–274, URL `http://doi.acm.org/10.1145/586110.586146`.

[Sie99] Sielken, R., Application intrusion detection. Technical report CS-99-17, Department of Computer Science, University of Virginia, USA, 1999. URL `ftp://ftp.cs.virginia.edu/pub/techreports/CS-99-17.ps.Z`.

[SKV03] Soman, S., Krintz, C. and Vigna, G., Detecting malicious Java code using virtual machine auditing. *Proceedings of the 12th USENIX Security Symposium*. USENIX Association, 2003, pages 153–168, URL `http://cs.ucsb.edu/~ckrintz/papers/usenix03.pdf.gz`.

[SMS99] Stillerman, M., Marceau, C. and Stillman, M., Intrusion detection for distributed applications. *Communications of the ACM*, 42,7(1999), pages 62–69.

[Som02] Somayaji, A. B., *Operating System Stability and Security through Process Homeostasis*. Ph.D. thesis, University of New Mexico, 2002.

[SS00] Sabater, J. and Sierra, C., REGRET: A reputation model for gregarious societies. Research Report 2000-06, Institut d'Investigacio i Intelligencia Artificial, 2000. URL `http://citeseer.ist.psu.edu/sabater00regret.html`.

[Sun94] Sun Microsystems, *SunSHIELD Basic Security Module Guide*. Sun Microsystems, 1994. URL `http://docs.sun.com/app/docs/doc/801-6636/6i10gcto4?q=BSM&a=view`.

[SVSM03] Sivashanmugam, K., Verma, K., Sheth, A. and Miller, J., Adding semantics to Web Services standards. *ICWS'03: Proceedings of International Conference on Web Services*. CSREA Press, 2003, pages 395–401.

[TCL90] Teng, H. S., Chen, K. and Lu, S. C., Adaptive real-time anomaly detection using inductively generated sequential patterns. *Proceedings of the 1990 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1990, pages 278–284, URL http://ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=63857.

[vdHLP05] van den Heuvel, W.-J., Leune, K. and Papazoglou, M. P., EFSOC: A layered framework for developing secure interactions betweeen web-services. *Distributed and Parallel Databases*, 18,2(2005), pages 115–145.

[Vil05] Viljanen, L., Towards an ontology of trust. *TrustBus'05: Proceedings of the 2nd International Conference on Trust, Privacy and Security in Digital Business*, volume 3592 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005, pages 175–184.

[W3C02] W3C, Web service choreography interface (WSCI) 1.0, 2002. URL http://www.w3.org/TR/wsci/. W3C Note.

[W3C04a] W3C, OWL web ontology language overview, 2004. URL http://www.w3.org/TR/owl-features/. W3C Recommendation.

[W3C04b] W3C, Web service choreography description language version 1.0, 2004. URL http://www.w3.org/TR/ws-cdl-10/. W3C Working Draft.

[WD01] Wagner, D. and Dean, D., Intrusion detection via static analysis. *Proceedings of the 2001 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2001, pages 156–168, URL http://ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=924296.

[WDD00] Wespi, A., Dacier, M. and Debar, H., Intrusion detection using variable-length audit trail patterns. *RAID'00: Proceedings of the 3rd International Symposium on Recent Advances in Intrusion Detection*, volume 1907 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000, pages 110–129, URL http://www.cs.fit.edu/~pkc/id/related/wespi-raid00.pdf.

[WFP99] Warrender, C., Forrest, S. and Pearlmutter, B. A., Detecting intrusions using system calls: Alternative data models. *Proceedings of the 1999 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1999, pages 133–145, URL http://citeseer.ist.psu.edu/article/warrender98detecting.html.

[Wor03] World Wide Web Consortium, W3C recommendation: Simple object access protocol (SOAP) 1.2, 2003. URL http://www.w3.org/TR/soap12-part1/.

[Wor04] World Wide Web Consortium, W3C recommendation: Extensible markup language (XML) 1.0, 2004. URL http://www.w3.org/TR/REC-xml.

[XDC04] Xu, H., Du, W. and Chapin, S. J., Context sensitive anomaly monitoring of process control flow to detect mimicry attacks and impossible paths. *RAID'04: Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection*, volume 3224 of *Lecture Notes in Computer Science*. Springer-Verlag,

2004, pages 21–38, URL http://citeseer.ist.psu.edu/702165.html.

[YKB93] Yahalom, R., Klein, B. and Beth, T., Trust relationships in secure systems - a distributed authentication perspective. *Proceedings of the 1993 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1993, pages 150–164, URL http://citeseer.ist.psu.edu/yahalom93trust.html.

[Zam01] Zamboni, D., *Using Internal Sensors for Computer Intrusion Detection*. Ph.D. thesis, Purdue University, 2001. URL http://www.cerias.purdue.edu/homes/zamboni/pubs/thesis-techreport.pdf.

Helsinki 2011