ANIKETOS

Secure and Trustworthy Composite Services

Seventh Framework Programme:
Call FP7-ICT-2009-5
Priority 1.4 Trustworthy ICT
Integrated Project

Deliverable ID:	Preparation date:
D2.1	19th July 2011

Milestone: Released

Title:

Models and Methodologies for Embedding and Monitoring Trust in Services

Lead beneficiary (name/partner):
Hisain Elshaafi/TSSG

Internally reviewed by (name/partner):
David Llewellyn-Jones/LJMU
Fabiano Dalpiaz/UNITN

Approved by:
Executive board

Abstract:

Aniketos is about establishing and maintaining trustworthiness and secure behaviour in a constantly changing service environment. The project aligns existing and develops new technology, methods, tools and security services that support the design-time creation and run-time dynamic behaviour of composite services, addressing service developers, service providers and service end users.

This deliverable describes models and methodologies for managing trust for services, mainly reporting the results of the first two tasks in WP2. A specific focus is on the compositional aspects of services as well as in their dynamic nature. D2.1 investigates trust as a multi-faceted concept. Technical models and mechanisms such as certification and Security-by-Contract aim to establish and maintain the trustworthiness of the composite services while user-centric factors will influence the establishment of trust among users of a service. Trust models and mechanisms are defined for both design-time development and runtime trust monitoring of composite services. Additionally, patterns and guidelines for establishing trust are defined to support service developers in designing systems that provide a more trustworthy experience for end users.

Dissemination level		
PU	Public	X
CO	Confidential, only for members of the consortium (including Commission Services)	

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 257930.

Aniketos consortium

Aniketos (Contract No. FP7-257930) is an Integrated Project (IP) within the 7th Framework Programme, Call 5, Priority 1.4 (Trustworthy ICT). The consortium members are:

SINTEF ICT
NO-7465 Trondheim
Norway
www.sintef.com

Project manager: Richard T. Sanders
richard.sanders@sintef.no
+47 73 59 30 06
Technical manager: Per Håkon Meland
per.h.meland@sintef.no +47 73 59 29 41

Tecnalia Research & Innovation
(TECNALIA)
E-48160 Derio
Bizkaia (Spain)
www.tecnalia.com/en

Contact: Erkuden Rios Velasco
erkuden.rios@tecnalia.com

Consiglio Nazionale delle Ricerche
(CNR)
00185 Roma, Italy
www.cnr.it

Contact: Fabio Martinelli
Fabio.Martinelli@iit.cnr.it

Thales Services SAS (THALES)
78140 Velizy-Villacoublay, France
www.thalesgroup.com

Contact: Dhouha Ayed
dhouha.ayed@thalesgroup.com

Liverpool John Moores University
(LJMU)
Liverpool, L3 5UX, United
Kingdom
www.ljmu.ac.uk/cmp

Contact: Madjid Merabti
m.merabti@ljmu.ac.uk

Selex Elsag S.P.A. (ELSAG)
16154 Genova, Italy
www.selexelsag.com

Contact: Paolo Pucci
Paolo.Pucci@selexelsag.com

SEARCH-LAB Ltd.
Budapest 1117, Hungary
www.search-lab.hu

Contact: Zoltán Hornák
zoltan.hornak@search-lab.hu

Atos Origin (ATOS)
28037 Madrid, Spain
www.atc.gr

Contact: Pedro Soria-Rodriguez
pedro.soria@atosresearch.eu

Waterford Institute of Technology (TSSG)
Waterford, Ireland
www.tssg.org

Contact: Miguel Ponce de Leon
miguelpdl@tssg.org

Universita Degli Studi di Trento (UNITN)
38100 Trento, Italy
www.unitn.it

Contact: Paolo Giorgini
paolo.giorgini@unitn.it

Athens Technology Center SA (ATC)
15233 Athens, Greece
www.atc.gr

Contact: Vasilis Tountopoulos
v.tountopoulos@atc.gr

SAP AG (SAP)
69190 Walldorf, Germany
www.sap.com/research

Contact: Achim Brucker
achim.brucker@sap.com

ITALTEL S.P.A. (ITALTEL)
20019 Settimo Milanese, Italy
www.italtel.it

Contact: Maurizio Pignolo
maurizio.pignolo@italtel.it

Paris-Lodron-Universität Salzburg (PLUS)
5020 Salzburg, Austria
www.uni-salzburg.at

Contact: Manfred Tscheligi
manfred.tscheligi@sbg.ac.at

Deep Blue SRL (DBL)
00193 Roma, Italy
www.dblue.it

Contact: Valentino Meduri
valentino.meduri@dblue.it

Wind Telecomunicazioni S.P.A. (WIND)
00148 Roma, Italy
www.wind.it

Contact: Rita Spada
MariaRita.Spada@mail.wind.it

Dimos Athinaion Epicheirisi Michanografisis (DAEM)
10438 Athens, Greece
www.daem.gr

Contact: Ira Giannakoudaki
giannakoudaki@ daem.gr

Table of contents

List of figures

List of tables

Executive summary

Aniketos aims to help in establishing and maintaining trustworthiness and secure behaviour in a constantly changing service environment. The project aligns existing and develops new technology, methods, tools and security services that support the design-time creation and run-time dynamic behaviour of composite services, addressing service developers, service providers and service end users.

Towards achieving Aniketos goals in relation to trustworthiness, this deliverable describes models and methodologies for establishing and maintaining trust for services. A specific focus is on the compositional aspects of services as well as in their dynamic nature. D2.1 investigates trust as a multi-faceted concept. On one hand, technical models and mechanisms such as certification and Security-by-Contract aim to establish and maintain the trustworthiness of the composite services, while on the other hand, social and user-centric factors also have significant influence.

Trustworthiness and security are closely-related concepts. Security mechanisms such as encryption, authentication, and authorization are necessary steps in establishing trust. For example, authentication mechanisms assure the service consumer that the provider of the service is who he claims he is. Similarly, a service might interact with its consumer through encrypted communication ensuring confidentiality. Nevertheless, security mechanisms alone are not sufficient as they do not assure the behaviour of the service. The service may not behave in the way it is required or expected in terms of reliability, availability, privacy, etc. In Aniketos we aim to examine multiple aspects of trustworthiness in service compositions, including how trust and security models and mechanisms, though distinct, can complement and support each other in services environments, particularly in service composition.

The trust models and mechanisms are based on trustworthiness requirements as specified in WP1. They are defined for both design-time development and runtime trustworthiness monitoring of composite services. Additionally, patterns and guidelines for establishing trust are defined in the deliverable to support service developers in designing systems that provide a more trustful experience for end users. The models and mechanisms described in this deliverable will be implemented in a prototype for D2.2 which includes a trustworthiness prediction module and a trustworthiness runtime monitoring module.

1 Introduction

1.1 Aniketos motivation and background

The Future Internet will provide an environment in which a diverse range of services are offered by a diverse range of suppliers, and users are likely to unknowingly invoke underlying services in a dynamic and ad hoc manner. Moving from today's static services, we will see service consumers that transparently mix and match service components depending on service availability, quality, price and security attributes. Thus, the applications end users see may be composed of multiple services from many different providers, and the end user may have little in the way of guarantee that a particular service or service supplier will actually offer the security claimed.

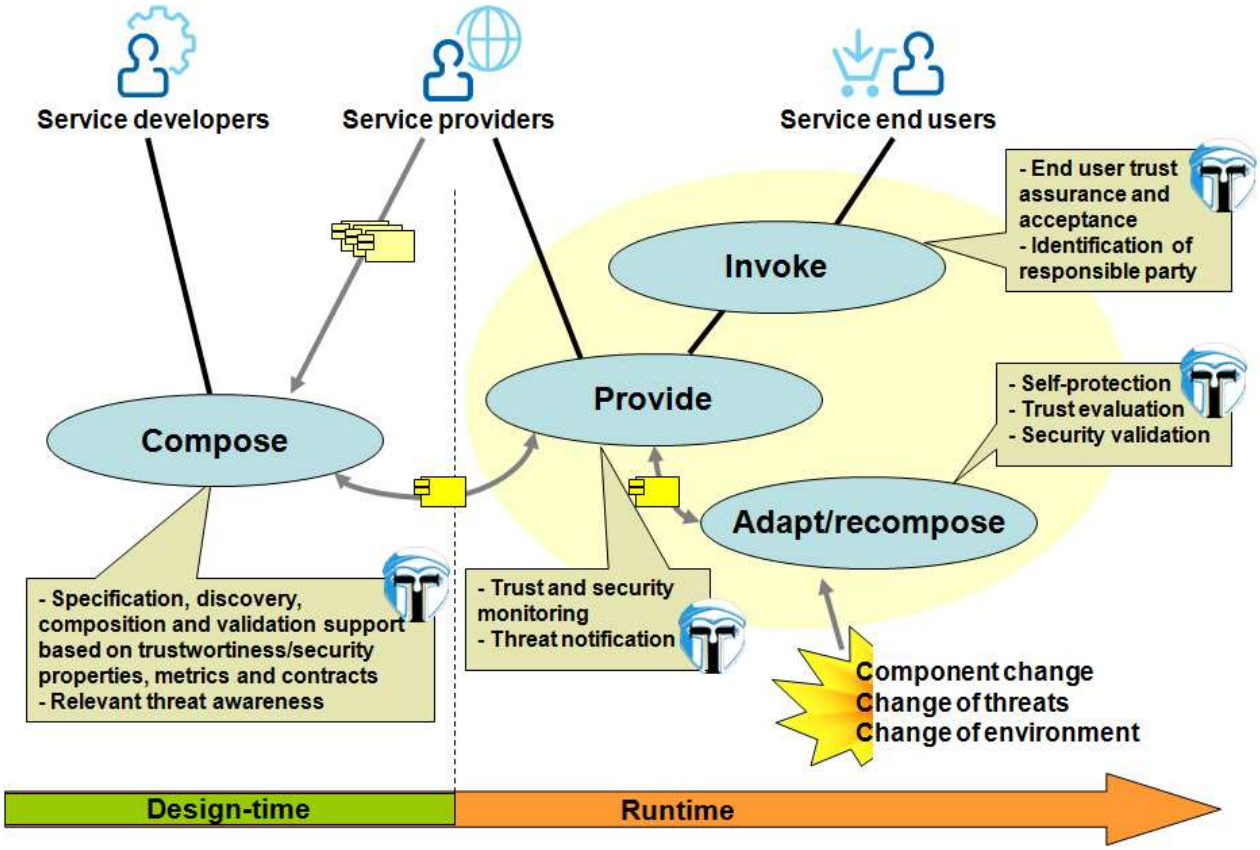


Figure 1: Goal: establish and maintain security and trustworthiness in composite services

Aniketos is about establishing and maintaining trustworthiness and secure behaviour in a constantly changing service environment. The project aligns existing and develop new technology, methods, tools and security services that support the design-time creation and run-time dynamic behaviour of composite services, addressing service developers, service providers and service end users.

Aniketos provides methods for analysing, solving, and sharing information on how new threats and vulnerabilities can be mitigated. The project constructs a platform for creating and maintaining secure and trusted composite services. Specifications, best practices, standards and certification work related to security and trust of composite services are promoted for inclusion in European reference architectures. Our approach to achieving trustworthiness and security of adaptive services takes account of socio-technical aspects as well as basic technical issues.

1.2 Structure of this document

The document is structured as follows: Chapter 2 provides an overview of existing relevant work on trust in computer science in general with particular focus on the social concept of trust and on technical methods and models of measuring and managing trust. The social trust concepts consider

domain-general factors which might influence trust, trustworthiness, and reputation. The chapter then explains lessons learned from the developments in technical trust models in Internet of Services (IoS), Web services, and service composition particularly in relation to their strengths and weaknesses.

Chapter 3 complements technical mechanisms for establishing trust with user-centric mechanisms for establishing trust among end-users of composite services. This area aims to help service developers in designing and running systems that are trustworthy.

Chapter 4 analyses functional, non-functional and architectural requirements based on WP1 tasks T1.3 (Aniketos platform requirements and scenarios) and T1.4 (Overall architecture of Aniketos platform). The requirements are relevant to both WP2 in establishing and maintaining trust in composed services and to WP1 in defining the socio-technical security modelling language and tool (T1.2).

Chapter 5 looks at various Aniketos mechanisms and techniques for managing service trustworthiness. It includes detailed models, metrics and systems, and how the systems can work together to establish and maintain trust.

The conclusion examines the coherence and integration of the models and mechanisms detailed in the deliverable in managing trust in service compositions.

1.3 Relationships with other deliverables

The D2.1 presented in this document relates to the following deliverables:

- *D2.2–Initial prototype of trust management, Security-by-Contract and Verification Modules:* this deliverable implements the models and mechanisms in a prototype and follows patterns and guidelines in developing and providing trustworthy services as described in D2.1.
- *D1.2–First Aniketos architecture and requirements specification:* provides scenarios and guidelines on the requirements and architectural constraints upon which solutions in D2.1 and other deliverables can be developed. D2.1 also helps provide feedback to the work in this deliverable.
- *D1.3–Initial version of the socio-technical security modelling language and tool:* chapter 4 in D2.1 analyses requirements relevant to WP1 in defining the socio-technical security modelling language and tool.
- *D4.1–Methods and design for the response to changes and threats:* The deliverable investigates measures for observing events that indicate the occurrence of threats or suggest the possibility of their occurrence. For instance, it discusses mechanisms to manage the trustworthiness of services that adopt fake identities to spoil the reputation of other services provided by competing providers. Trustworthiness monitoring requires receiving threat alerts that could result in changes in the trustworthiness of services. Conversely, a change in trustworthiness of a composite service as evaluated by the trustworthiness modules is notified to threat and change modules designed in D4.1.

1.4 Contributors

The following partners have contributed to this deliverable:

- TSSG
- PLUS
- SAP
- CNR
- Thales
- Tecnalia

1.5 Acronyms and abbreviations

CC	Common Criteria	CI	Confidence Interval
CM	Contract Monitoring Scenario	CVE	Common Vulnerabilities and Exposures

DDoS	Distributed Denial of Service	DHT	Distributed Hash Table
FV	Formal Verification	HMM	Hidden Markov model
IoS	Internet of Services	MCI	Manual Code Inspection
P2P	Peer to Peer	PECM	Policy Enforcement & Contract Monitoring Scenario
QoS	Quality of Service	RT	Role-based Trust-management framework
RTG	Random Test Case Generation	RTML	Role-based Trust Management Language
SCA	Static Code Analysis	SOA	Service Oriented Architecture
SSE-CMM	System Security Engineering Capability Maturity Model	STG	Specification-based Test Case Generation
SxC	Security by Contract	SxCxT	Security-by-Contract-with-Trust
TAN	Transaction Authentication Number	TTP	Trusted Third Party
UML	Unified Modelling Language	WS	Web Service

2 Trust models in computing and Internet of Services

This chapter provides surveys of existing relevant work on trust in computer science in general with particular focus on the social concept of trust and on methods and models of measuring and managing trust. The chapter then explains lessons learned from the developments in trust models in Internet of Services (IoS), Web services, and service composition particularly in relation to their strengths and weaknesses.

2.1 Social and cognitive models for trust

This section examines existing research in the area of social and cognitive trust that is relevant to Aniketos. The goal is to provide a broad overview of the issues addressed in the more general literature. Later sections of this report will deal more specifically with trust concerning interactions between the entities in Aniketos, e.g., services and developers.

Much research has investigated the puzzle of why genetically unrelated individuals often trust each other, a phenomenon which cannot be explained by assuming that people act in self-interest [32]. A further complication, present in the SOAs envisaged by Aniketos, comes from *disembedding* [34]: the shift from transactions between people who are spatially close to each other and which occur over a short time-span, towards transactions between people who may never meet, and where the outcome of the transaction may take a long period of time (consider the example of buying a rare book online from the other side of the world which may take weeks to arrive). There are also issues of control in Aniketos, e.g., through contracts, which have a bearing on trust.

2.1.1 What is cognitive trust?

There are many definitions and conceptualizations of trust. For social and cognitive trust, we use the theoretical framework of Castelfranchi and Falcone [30] on which to hang empirical results from elsewhere. This line of research has a long history; it uses elements from formal logic so it is compatible with the methodologies used in parts of Aniketos; it addresses trust between (living, breathing) cognitive agents, which is clearly important for both the user-centred development of the Aniketos platform itself and also for evaluations of the eventual outcomes; finally earlier manifestations of the theory have been used in empirical psychological investigations of trust (e.g., [28]).

Trust can be considered as a five-part relation [30],

$$trust(X, Y, c, \tau, g_x),$$

which denotes that X (the trustor) trusts Y (trustee) in context c to do task τ (action α, leading to state of affairs, p) to achieve X's desired goal, $g_x \subseteq p$.

The following subsections elaborate on important aspects of how the trust relationship comes into being.

2.1.1.1 Trust(worthiness) judgments made by the trustor

The basic structure of the trust relation between X and Y involves a belief of X (or lack of belief) about the trustworthiness (or untrustworthiness) of Y. This expands to four possibilities [30]. (Each of these possibilities refers implicitly to a particular context and goal.)

1. *(Distrust) X* believes that *Y* is untrustworthy.
2. *(Lack of trust) X* doesn't believe that *Y* is trustworthy.
3. *(Lack of distrust) X* doesn't believe that *Y* is untrustworthy.
4. *(Trust) X* believes that *Y* is trustworthy.

When taking into account the goals of *X*, there are two possible refinements of untrustworthiness:

1. *(Inadequacy) X* believes that *Y* can't achieve *g*, but *X* wants to achieve *g*.

2. *(Nocivity) X* believes that *Y* can achieve *g*, but *X* wants to achieve not-*g*.

2.1.1.2 Properties of the trustee affecting trust(worthiness)

The verb "to trust" applies to the trustor (*X*), whereas "to be *trustworthy*" applies to *Y*, the trustee. The degree to which *X* trusts in general can vary from trustor to trustor, so the degree of trust *X* has in *Y* may be at odds with *Y*'s actual trustworthiness. Objective trustworthiness is often unreachable (e.g., a program has a security flaw which hasn't yet been detected or we just haven't tested or proved all possibilities of execution), though untrustworthiness can be observed (e.g., supposedly private details are leaked and displayed on a Web page). Domain-general properties of *Y* affecting trust relations include [30]:

- How competent *Y* is at performing the tasks, τ. For a cognitive agent, this could be inferred from, e.g., cognitive ability, training, knowledge possessed. (For non-cognitive agents, such as software services, see Section 3.1.4.2 for a range of factors which software developers believe are necessary.)
- How predictable *Y* is in terms of actually perform the tasks, e.g., concerning motivation in cognitive agents and reliability in all kinds of agents.

2.1.1.3 Goals

Most goals cognitive agents want to achieve depend on sub-goals carried out by other agents. Correspondingly, a lot of the time individuals work on tasks to enable others to achieve their goals. It might therefore come as no surprise that a major set of cognitive functions concerns representing and drawing inferences about the goals of others. Proponents of so-called mirror-neuron theories argue that in doing so, we simulate the actions of others using, e.g., our own motor system. So for instance on perceiving the actions of another, we take advantage of our own mechanisms for *generating* the action in order to infer the goal behind another agent's action [33]. This sort of simulation-driven social cognition seems to develop early, before language.

In the 1940s it was observed that people tend to over-infer goals in entities which clearly cannot have them, such as animated triangles [36]. Interestingly (and independently), E. W. Dijkstra commented that anthropomorphic reasoning was rife in software development too [72]:

> *The anthropomorphic metaphor is perhaps even more devastating within computing science [...]. Its use is almost all-pervading. To give you just an example: entering a lecture hall at a conference I caught just one sentence and quickly went out again. The sentence started with "When this guy wants to talk to that guy...". The speaker referred to two components of a computer network.*

Dijkstra argues that such thinking impedes the development of correct software. Irrespective, this quotation does support the idea that software developers and designers are likely to reason, perhaps implicitly, about software components as if they had goals and perhaps trust (or distrust) the components in a similar way to how they trust cognitive agents.

2.1.1.4 The role of emotions

There is a close correspondence between processes concerning trust, goals, and emotions. For instance according to Oatley and Johnson-Laird's cognitive theory of emotion [42], to achieve or make progress towards a goal leads to happiness; to fail to make progress leads to sadness; to have progress towards a goal blocked by an agent leads to anger. There are also various more complex emotions concerning goals such as remorse, which is a social emotion related to regret but directed towards another person. Cognitive processes underlying such emotions are core to trustworthy behaviour, e.g., a major goal of cognitive agents is to avoid negative emotions. Nowak [41] cites the "warm inner glow" experienced when people perform an altruistic action. Baumgartner et al. [26] argue that trust involves processes for reducing fear processing, which should facilitate delegation to another agent.

Fehr and Gächter [31] argue that "emotions are an important proximate factor behind altruistic punishment".

2.1.1.5 Trust as a basic cognitive process

Systems found in non-human animals are also involved in trust in humans. For instance Kosfeld et al. [39] demonstrated that intra-nasally administered oxytocin, a neuropeptide involved in a range of behaviours including lactation, increased trust in humans during an economic game. This low-level intervention demonstrates that basic systems are involved in trust. Contrast this with, for instance, the mechanisms required for language. Given how basic trust processes are, it is likely that, whether they like it or not, trust decisions made by software developers working in a knowledge-rich domain are influenced by a range of factors from these more basic cognitive systems, i.e., trust in relation to reasoning about SOAs is likely to go beyond the purely technical mechanisms involved with SOAs.

2.1.1.6 The lifecycle of trust

There are three main stages in the trust lifecycle to be analysed [30]:

1. The *evaluation* of the trustee on the basis of available information, including e.g., reputation and previous personal experience of the trustee. There are two main types of evaluation, one involving implicit decisions (e.g., based on prior experience and routine), and the other a more explicit deliberation of reasons. The two types are not independent. Although it may be possible to find very general notions of trustworthiness, such as honesty and efficiency, the evaluation is always with respect to a particular goal.
2. An *intention* to delegate to the trustee. This is a positive evaluation, i.e., that *Y* is good for achieving the goals by carrying out the required actions.
3. The trust *decision*, i.e., the actual action of trusting. This has happened when, for instance, the system goes live with a particular trusted service being used as an integral part.

One important distinction is the difference between a *trust decision*, which is a binary "trust or don't trust", versus a belief-based degree of trust resulting from the evaluation. For instance it is possible that one uses a particular service whilst simultaneously having a low degree of belief that the service will carry out the actions necessary to achieve the trustor's goals. For Möllering [40], a key component of the trust decision is (p. 191), "the suspension of vulnerability and uncertainty (the leap of faith), which enables actors to have positive expectations of others." A range of authors argue that trust is not possible without this vulnerability and uncertainty – if it were removed then trust would not be necessary.

2.1.1.7 Delegation

Delegation, the act of relying upon the trustee is central to trust. There are three types of delegation [30]:

- *Weak delegation*: the trustee is unaware that he or she has become part of the trustor's plan to achieve the goal. The trustor is not actively doing anything in order to affect the trustee's behaviour.
- *Mild delegation*: the trustor is acting to influence the trustee, but the trustee is unaware of this and has not agreed.
- *Strong delegation*: the trustee is aware of and has agreed with the trustor's intention.

This analysis can again be related back to the presence or absence of cognitive trustees. For instance, an action cannot be strongly delegated to a software component as a software component is not aware of anything, not being a cognitive agent. However through a chain of trust connections it may be possible for cognitive agent somewhere in the chain to be aware of the delegation (e.g., it is recorded in a log file and eventually a person might face consequences for a failure to achieve the trustor's goal). Thus there could be strong human-to-human delegation through legal contracts and mild

human-to-service delegation to the software component. Weak delegation might involve aspects of a system which the trustor cannot influence, e.g., underlying network systems which are not directly affected by the trustor's specific goals at a particular time. Mild delegation would include operations designed to correct an error made by a trustee, but without the trustee being aware of this.

These distinctions might disappear at some levels of abstraction, but the key point is that whether or not a cognitive agent somewhere is aware of the delegation – the specific act of delegation rather than delegation more generally – can lead to different outcomes. See Section 2.1.3 for an example.

2.1.1.8 Reputation

Trust and *reputation* are close relatives. Reputation can be viewed as an aggregated opinion, distributed, e.g., by gossip, which can influence inferred trustworthiness. Reputation becomes important when *direct reciprocity*, i.e., cooperation between people who repeatedly encounter and depend on each other, is not possible, but where *indirect reciprocity* is in operation, e.g., fleeting encounters with strangers [44]. Nowak [41] (p. 1561) draws an analogy with money:

> *Direct reciprocity is like a barter economy based on the immediate exchange of goods, whereas indirect reciprocity resembles the invention of money. The money that fuels the engines of indirect reciprocity is reputation.*

A naïve analysis of trust between strangers might lead one to believe that people would be selfish in their transactions. This is not so. Nowak summarizes evidence of two kinds of indirect reciprocity:

- *Upstream* reciprocity, where *A* helps *B*, then as a consequence *B* goes on to help *C*.
- *Downstream* reciprocity, where *A* helps *B*, and on observing this, *C* helps *A*.

See Sections 2.3 and 5.2.2 for more on reputation, focusing on Aniketos concerns.

2.1.1.9 Cognitive versus non-cognitive agents

Castelfranchi and Falcone [30] argue that the trustor (*X*) must be a cognitive agent with goals and intentions. Or to put it another way, to trust, according to the authors, it is necessary (though not sufficient) to be able to care about achieving a goal. A computer cannot trust, according to the authors, for any sensible notion of trust. However the *trustee* (*Y*) need not be a cognitive agent, so, matching intuitions, it is possible according to the theory to trust a software component. It seems likely that building a bridge between the trustors and the living breathing software developers behind services would lead to different cognitive processes engaging than if trust concerns only SOA services.

Trust can also act as a signal, according to the authors, which may affect in turn trustworthiness of the trustee. So for instance if a cognitive agent realizes she is being trusted by another person, this can in turn increase her trustworthiness as she doesn't want to let down the trustor. A software component cannot be affected by a trust signal in the same way that a person is affected; however it is possible that the developers of a component, if they have some contact with the trustors, might change their behaviour as a result of being aware the component is being trusted. It could be argued that it would be possible for a component to detect a trust signal in some way, e.g., a user rating, and, say, allocate more resources to users who begin to distrust. This is not the same, however, as the emotional response that a person experiences on learning that he or she is distrusted.

There is also the important case of SOAs involving services where people, rather than only software, are directly involved in the service's actions. Consider Alice, who chooses to buy a book online. She trusts a website to take her credit card details and to deliver the book. Eventually, somewhere down the line, this trust relation extends, however implicitly, beyond the website to a cognitive agent, a postal worker.

Care must be taken when considering, for instance, monitoring. A software component does not and cannot care about intensive monitoring. If the same monitoring algorithms were applied without thought to an SOA linking to a person's behaviour, e.g., the postal worker, then the result is likely to be quite different, leading to untrustworthy behaviour and possibly negative emotional consequences.

There are also privacy issues involved if, e.g., the performance or "quality of service" of individual employees might be made visible on – or indirectly inferred from – an online marketplace, whereas again individual instances of software do not care.

Even the use of the term "trust" has an impact on cognitive agents. Williamson [45] expresses concern that "trust" is often used to mean something like expectation of absence of risk – a use which he terms *calculative* trust. The argument is best summed up in the following quotation (p. 485):

> *Not only is "calculated trust" a contradiction in terms, but user-friendly terms, of which "trust" is one, have an additional cost. The world of commerce is reorganized in favour of the cynics, as against the innocents, when social scientists employ user-friendly language that is not descriptively accurate-since only the innocents are taken in.*

Castelfranchi and Falcone [30] take up the issue in terms of whether or not there is something in trust beyond risk and calculation, which clearly in their theory there is. However the main issue taken up by Williamson is that the word "trust" is often abused as it carries warm, pleasant connotations – it is used as a marketing ploy to fool people. We should be wary of any attempt to call purely calculative, inferential, mechanisms "trust". However the mechanisms *underlying* trust, both in humans' cognitive systems at a sub-personal level of explanation and in the Aniketos platform, can be given a purely calculative and inferential characterization. From the perspective of human-computer interaction, care must be taken when communicating these mechanisms to users.

2.1.2 Empirical studies of trust

There are three main perspectives to take when studying trust: that of the trustor, the trustee, or the perspective of some third-party. In SOAs, the various agents involved may take on different roles. For instance the service developers must trust the framework to charge service users (i.e., they are trustors). They are also trusted by the service users to provide services which do what they want them to do (i.e., they are trustees). So trustors are often simultaneously trustees and vice versa.

One way to study trust is to ask people to describe situations in which they have chosen to trust others. Möllering [40] reports a series of qualitative interview-based studies on trust between people, which illustrates the kinds of answers people tend to give. Examples of reasons for trust given by participants include:

- *Personal experience* with trustee, e.g., "… past history and experience" (p. 165). One participant goes as far as to say that he can *only* trust through experience. Another claims that "you can only establish trust if you put it to the test" (p. 168). Concerning tests of trust, one participant warned "Don't make too many promises, because the surest way not to have any trust is when you let somebody down" (p. 183).
- *Inferred competence*, e.g., "[the trustee's] ability to achieve what they say they can achieve and their consistent ability to achieve their promises" (p. 166) and another participant noted "He knows his business, he knows his papers, he responds very well to anything that I put in front of him" (p. 183). Many of the factors participants listed falling under this category may be seen as fairly domain specific, i.e., to infer someone's competence to carry out some actions implies knowledge of the domain.
- *Mutual dependence*, e.g., developing over time as people help each other out. Another example of this came up in supplier-buyer relations (p. 169): which partner has the "upper hand" (e.g., buyer's market versus seller's market) can change in a cyclic manner, again motivating dependence. Another noted that "we both know our mutual benefits" (p. 182).
- The *quality of relationships* built through socializing outside the work context. One trustor notes that "to get to know each other is very important in building new [trust] relationships" (p. 176).
- Appeals to fairly stable and *general traits*, e.g., the character of the trustee. One trustor described the trustee as "quite open and honest" (p. 172), another as "just nice people" (p. 174) and that "[the trustee] tries to get through the day with a smile and is not confrontational" (p. 175).

Appeals to intuition and an absence of discursively accessible reasons were common, e.g., (p. 182) "You have a feeling for someone, and their actions and deeds will lead you to the conclusion that actually you [can or] can't trust them. It's a very difficult thing to put your finger on. I tend to work on gut feeling." The descriptions of how trust grows by experience were often similarly unenlightening, e.g., the "[trust relationship] has gradually grown, really" (p. 175). This is a general problem. Much of cognition is inaccessible to consciousness. Another (general) problem is that people often tell stories about the supposed basis of their actions which bear little resemblance to the "real" reasons.

Behavioural economists, experimental economists and more recently neuroeconomists, study how people play games involving real money in order to investigate the various factors influencing trust and trustworthiness. These experiments tend to avoid interpreting what participants *say* their reasons are, instead focusing on the *actions* people make. One example game [27] involving two players, variations of which are prevalent across the economics literature, goes as follows:

1. Both players initially receive $10
2. The trustor can choose either
    a. to keep the $10 or
    b. to send the $10 to the trustee
3. If the trustor sends the money, then it is doubled to $20, giving the trustee a total of $30
4. Now the trustee can choose either
    a. to send back $15 or
    b. to send back $8.

The various actions indicate various trust behaviours: (2a) indicates distrusting behaviour, (2b) trusting behaviour, (4a) is a trustworthy behaviour and (4b) is an untrustworthy behaviour, even if no prior agreement has been entered into between the participants. One assumption here is that the "trustor" isn't happy just to donate money to a stranger and doesn't expect to receive anything back – this would require neither trust nor trustworthiness. (Or this could be analysed as the trustor trusting that the trustee will keep all the money.) Another assumption is that the players haven't agreed in advance that, say, the trustee is not going to send back the money – it is assumed that all participants want to receive their fair share of the available money. Much of the task is implicit; but this is often the case with other forms of interaction. For instance people do not explicitly enter into an agreement with others to follow maxims of cooperation in conversation, such as to be informative, but still people are often consistent with these maxims, even in quite abstract contexts involving probability tasks [88]. It's also not clear why participants would punish each other for failing to be fair with the money (see next section) if there weren't implicit social norms concerning fairness. There are cultural differences in these norms [89].

There are many variations on this basic theme, for instance in some experiments (and conditions), the trustor has the option of punishing the trustee for unfair behaviour [32].

2.1.3 Issues of monitoring, control, and punishment

Agents often try to minimize the role of trust in interactions. For instance legal contracts can be used to try to incentivize trustworthy behaviour. The trustor can monitor the actions of the trustee and, for instance, withdraw trust if it is seen that the trustor's goals are not being achieved. These actions are often costly to both parties. For instance the trustee might have to record his or her actions, using resources for monitoring which might be better applied to the delegation-specific tasks, although increasingly monitoring can be done automatically with minimal cost to trustor or trustee. Legal interventions are often time consuming and monetarily expensive.

Fehr and Rockenbach [32] performed an experiment which has a bearing on the issue. They used a game similar to that devised by Bohnet and Zeckhauser [27] described above. The crucial differences were that:

- The trustor could specify the desired amount of money to be sent back.

- In one *incentive* condition, the trustor could decide whether a fine would be applied if less than that amount was not sent. (The other *trust* condition was as before with no fines.)
- The trustee in the incentive condition knew in advance whether a fine would be applied.

The least amount of money was sent from the trustee to trustor in the incentive condition in which fines were in operation, especially when the desired amount of money was high. The largest amount returned was for the condition where the trustee was aware of the possibility of fining, but that the trustor had chosen not to use the option. Values returned for the trust condition, where there were no fines and also where the possibility was not an option, lay in between. How trustworthy trustees are thus appears to be affected by the kinds of controls put in place by trustors.

There is also evidence that people will altruistically punish others for untrustworthy behaviour, even if it is costly to do so. Fehr and Gächter [31] present evidence that this is possible, again in the context of trust games involving money transfers, even when reputation would be unaffected since the games are "one-shot". This was specifically for the case of free-riding in a financial context, so might not generalize to other forms of (un)trustworthy behaviour.

2.1.4 Summary of relevance to Aniketos

This section briefly reviewed a multidisciplinary theoretical model of trust and a series of empirical results on trust. These theories and data are of importance to Aniketos as they suggest domain-general factors which might influence trust, trustworthiness, and reputation. An important implicit assumption is that how software designers and developers deal with trust in their day-to-day lives (e.g., how explicit their inferences about trust are; the impact of reputation; issues of control, etc.) will have some bearing on how they deal with trust in their professional life. To some extent we can test this assumption – and we make steps towards doing so in Section 3.1. Later we also investigate on domain specific factors in Aniketos.

2.2 Trust and security

This section explains the relationship and differences between the concepts of trust and security in computing and how this applies to Aniketos.

2.2.1 Commonalities between trust and security

In the field of computer security, the word "trust" is often synonymous with "security". An example of this is the Trusted Computing concept and Microsoft's Palladium [57]. However, in business and social contexts they have different meanings. Security mechanisms such as encryption, authentication, and authorization are necessary steps in establishing trust. For example, authentication mechanisms assure the service consumer that the provider of the service is who he claims he is. However, the Web service may not behave in the way it is required or expected in terms of reliability, availability, privacy, etc. Similarly, a service might interact with its consumer through encrypted communication ensuring confidentiality. However, the security mechanisms are not sufficient as they do not assure the behaviour of the service provider or service consumer. Instead, they only partly ensure the trustworthiness of the service.

In Aniketos we aim to contain multiple aspects of trustworthiness in service composition. In general, trust and security are distinct issues but their models and mechanisms complement and support each other in the services environments, particularly in service compositions.

Security in general is relevant whenever access to information must be guaranteed or restricted. The essential considerations are the availability, integrity, and confidentiality of information. It is usually assumed that the organization aiming at preserving those properties is also the one with the means of controlling them. For example, typical security models such as Role-Based Access Control (RBAC) are designed to provide an organization with a fine-grained control over access to its resources. However, due to their reliance on a centrally managed database of users and roles, they are not suited

to model security policies across organizations. In the latter case, the need arises for an organization to allow another some control over the attribution of access rights. This ability for an organization to delegate some control over parts of its security policy to another one naturally brings concerns over the risks involved in such a choice. As those risks are never eliminated, a relation of trust must exist between the organizations (hence the perceived need for the Trusted Computing concept and Microsoft's Palladium as mentioned above). Risks include failure by the trustee to enforce the security policy expected by the trustor, and deliberate betrayal due to diverging interests.

2.2.2 Differences between trust and security

In the context of Aniketos (i.e. SOA), we see "trust" as a property that characterises the relation between a service provider and a service consumer, whilst "security" relates *individually* to the service provider, and potentially also to the service consumer (cf. Figure 2).

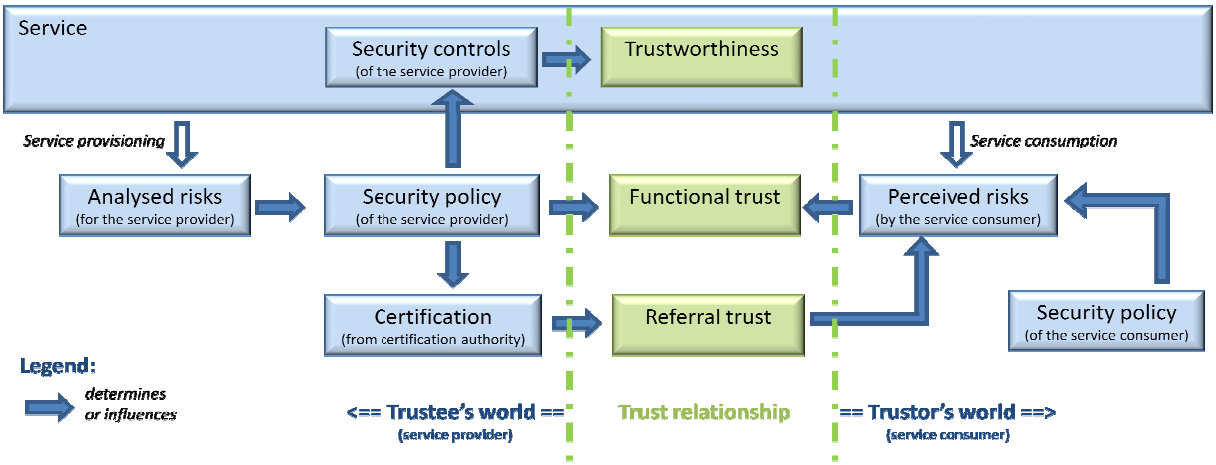


Figure 2: Relationship and differences between the concepts of trust and security

For a given service, the service provider analyses the security risks from his perspective[1], and:

- defines the adequate security policy: this security policy is usually made known[2] to the potential service consumers, who will use it to decide whether or not to trust the service (functional trust [48]);
- implements the corresponding security controls: the reliability of the implementation of the security controls is often difficult to assess by the service consumer[3], however in some cases, e.g. use of HTTPS, the service consumer may be able to assess the trustworthiness of the service, and this will influence his perceived risks in using the service;
- may call upon a certification authority to certify his service: typically the certification authority will check for malware and assure the authentication of the service provider; additionally, the

[1] It is important to understand here that the risks to the service customer are assessed only through the consequences for the service provider, e.g. will the unhappy customer sue the service provider, or will he publicize so widely his mishaps that the image of the service provider will be dramatically damaged.

[2] We assume here that the publicized security policy is the real implemented policy (a good portion of the Aniketos platform is intended to determine precisely this). A mismatch between the two may indeed lead to disaster, e.g. The LinkUp experience, when the company deleted nearly half of its user files, some of which were not recoverable, cf. http://downloadsquad.switched.com/2008/07/11/mediamax-is-dead-the-linkup-is-dead-streamload-is-dead/).

[3] Again, this is one of the major challenges that Aniketos is addressing, cf. in particular security-by-contract (SxC).

certification authority can provide adequate encryption means[4]; the certification authority's logo on the service is a major source of referral trust [48] for the service consumer.

Note that actors other than certification authorities may also provide referral trust. This is not shown in the above figure.

2.3 Technical modelling and measurement of trust

This section looks at existing mechanisms and techniques for modelling, managing and measuring trust. It compares trust systems developed or proposed by researchers and others including theoretical, algorithmic, and implemented systems. It also covers existing work on trust in Internet of Services (IoS), Web services, and service composition.

Over the last decade, several research initiatives in computer science have worked on the problems related to trust and trust management. A trust model provides mechanisms for establishing trust among entities and verifying the identity of participating entities and their credentials and characteristics such as name, password, certificates, reputation, etc. Trust models identify specific mechanisms for responding to specific threats and/or vulnerabilities. They facilitate validation of an entity's identity and/or the characteristics necessary for a particular event or transaction. Web services standards have allowed several identity-based trust architecture models to evolve including brokered, pairwise and federated trust models. However, these models are limited to the support of trusting the identity but not the overall trustworthiness of the service. Additionally, in trust relationships that span multiple organizations, the requirements and expectations of individual Web services may vary. Hence, whether the provider is a trusted entity in terms of its identity, the requester may still receive inadequate or malicious response to a request. Likewise, providers may receive erroneous or malicious service requests. The following are popular trust models and approaches used to improve consumer trust towards service and providers. They can be classified into:

- policy and security based,
- compliance and certification based,
- reputation and QoS based models and approaches.

The main focus is on how Aniketos will benefit from the knowledge of strengths and weaknesses of those models in building the project's trust models. Figure 3 shows the main approaches commonly used in establishing and maintaining trust by consumers towards services and service providers (modified from Malik and Bouguettaya [19]). The service provider aims to build and expand its client base through maintaining its service's trustworthiness. The consumer benefits from the trust by reducing risk and making more informed decision in relation to the service use.

[4] We assume here that security is not the domain of expertise of the service provider, and that he may therefore subcontract encryption to the certification authority (if and when this service is offered by the certification authority).

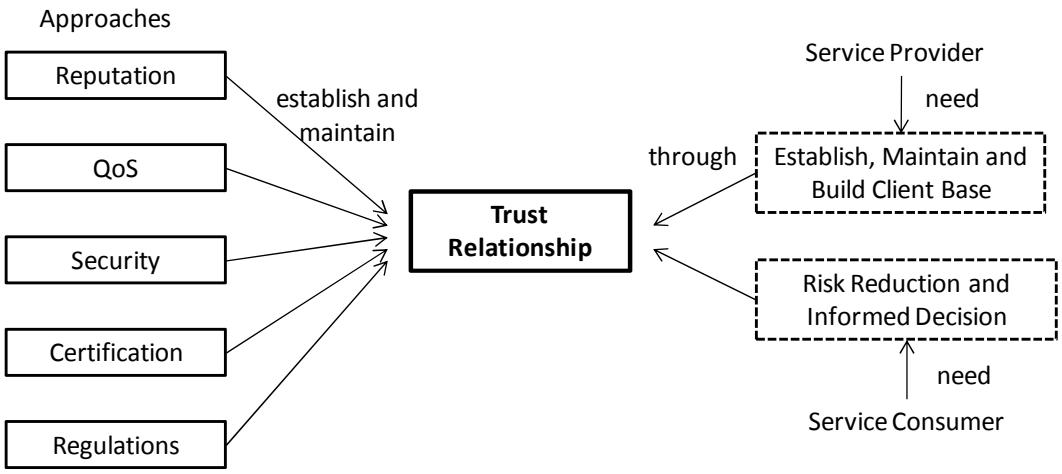


Figure 3: Approaches in establishing and maintaining consumer trust in services and service providers

2.3.1 Policy and security based trust models

The research on assessment of trustworthiness using declared policies started several years ago. With the development of Service Oriented Architectures and other related concepts, such as Web services, Grid, P2P, and Clouds, this research became even more important. Since the service consumer interested in the trustworthiness of the provided service is situated far from the service itself and may not have knowledge or control of the service, the only information or assurance available to the consumer is the set of policies the service provider declares. Thus, the decision whether to trust the data or execution of a business process is often based on a set of policies and/or a trust authority. The research on assessment of trustworthiness using declared policies started several years ago and it was mainly based on the definition of formal policy languages and tools.

Martinelli and Petrocchi [50] suggest a direction for the modelling and analysis of both security and trust aspects of distributed systems like, e.g., mobile ad hoc networks, peer to peer systems and Web services. The authors consider two well-known policy languages: 1) the Role-based Trust-management framework (RT, [53]) and, in particular, its most basic language RT0, and 2) the transitive trust model [52], for defining trust and recommendation relationships. First, an encoding of the transitive trust model into part of RT0 is shown. Then, this subset is mapped into the inference construct of the Crypto-CCS process algebra [54]. Indeed, process algebras enriched with cryptographic primitives are usually the input languages for automated machinery performing formal analysis of security protocols. The idea proposed by Martinelli and Petrocchi [50] is to use the same machinery for formal analysis of trust properties. Also, some language operators dealing with quantitative levels of trust are introduced by the authors.

The relationships between these languages allow modelling and analysing trust and recommendation policies in distributed systems by means of standard formal techniques, based on inference systems. In particular, the framework can be extended for enforcing security policies with mechanisms for managing reputation and recommendation information. However, the work lacks correctness guarantees for the encoding and considers only the basic version of the RT family of languages.

In [51], the authors focus on process algebras and show how the same machinery applied for formal verification of security protocols can be useful to model and analyze trust management procedures. The presented framework is intended to uniformly model security protocols and some form of trust and reputation management procedures, like Automated Trust Negotiation. The authors aim at showing that a strong connection exists between methodologies used to establish, manage and negotiate trust and the security mechanisms used to guarantee the confidentiality and integrity of information.

It is possible to exploit process algebra-based languages, like Crypto-CCS, as suitable languages modelling message exchange and manipulation. Along with native security features, their expressiveness also allows the definition of credential chains, trust/recommendation policies, and access control policies based on credentials. The weakness of this work, that however gives hints for future work, is that the automated tool providing both security and trust analysis has not been developed yet.

In [10] the authors deal with the integration of access control policies and access rights management in a Grid architecture. This is one of the first proposals in the Grid area for the definition of a fine-grained security policy integrating behavioural and trust management aspects. The paper describes the implementation of an enhanced access control system for the Globus toolkit that results from the integration of two security tools: Gmon and iAccess. iAccess performs the credential negotiation between the user and the service provider when the request for a new job is submitted by the user. Gmon exploits those credentials, during the execution of the job, to determine whether the accesses requested by the job should be allowed or denied. In particular, the security policy enforced by Gmon evaluates the trust level of the user by processing the submitted credentials.

The paper presents some key features with respect to other access control systems**,** among which are a fine-grained behavioural control combined with a credential-based access control mechanism, and an application-level granularity management of users' credentials. However, the performances evaluation of the proposed architecture is preliminary and more complex real Grid scenarios need to be addressed. Also, trust management aspects need to be investigated more deeply.

Along this line of research, Colombo et al. [12] present an integrated architecture, extending the previous one, with an inference engine that manages reputation and trust credentials. They implement a role-based trust management (RTML) module, enriched with trust management functionalities. In particular, they add the possibility of expressing the trust relations together with the credentials specifications. In this way, principals can apply operations on the trust values in order to obtain the propagation of these values even in the case of previously unknown parties.

The strengths of the paper are: i) an implementation of the RTML framework, embedded in a behavioural policy, to perform fine grained access control and trust management in Grid; and ii) an extension of the RTML framework to allow also reputation management. Improvements to this work could be brought about by evaluating the performances of the overall proposal in more complex scenarios.

In [13] a mechanism for negotiating trust credentials is introduced by the authors to overcome some scalability problems. The authors propose a policy-driven credential-based authorization system to protect Grid computational services against the malicious behaviour of applications submitted for execution. The authorization process is split into two levels: a coarse-grained level that manages access to a computational service and a fine-grained level that monitors the behaviour of applications executed by the computational service. The framework guarantees that users authorized on a coarse-grained level behave as expected on the fine-grained level, by enforcing a fine-grained security policy that monitors the behaviour of the application and exploits the trust credentials to determine the rights of the application to execute the accesses. The framework defines trust negotiations at a coarse-grained level to overcome the scalability problem, and preserves privacy of credentials and security policies of, both, Grid users and providers.

Even if the work is strongly tied to Grid architectures, the basic idea consists of considering trust as a metric for deciding the reliability of the provider and the user. Nevertheless, this proposal relies on a monitoring environment that simply halts the execution when something that contravenes policy happens. However, the model is general enough to allow other kinds of countermeasures rather than simply stopping the execution. This is left by the authors for future work.

In SOA a service consumer needs to invoke the service which provides the required functionality. If several alternatives exist the consumer has to select the service which provides the functionality in the best way, i.e., the non-functional requirements come in to play. Since Aniketos has its focus on

security we consider security as the main quality for selection. Thus, the service consumer should select the service which it trusts the most. First, the service consumer should find the service which claims to provide good protection. Then, the service consumer wants to be sure that the service will fulfil its claims. The first part of the process of establishing trust is done based on the security policies provided by the service. The main problem here is to rate the services according to the security level they provide (i.e., the level of trust that the service is secure enough). The second part of the trust establishing is performed by other means (e.g., reputation check, certifications, establishing monitoring/enforcement mechanisms). First we consider how a service consumer is able to select the most trusted service using the provided policies.

R. Henning [2] was one of the first researchers who tried to quantify the security for an enterprise. The main idea in [2] is to consider how an enterprise carries out general security practices. Fifteen generic security practices have been defined and broken down into more specific practices. Every practice is evaluated using some criteria and a level from 1 to 4 is assigned according to how well the practice is addressed by the enterprise.

This approach allows the description of security SLAs in a standardized way. On the other hand, the approach only provides a set of levels (one level for each specific security practice). These levels are not aggregated even for generic practices. Therefore, the proposed approach allows comparing different SLAs using only a partial order. Moreover, the levels have been defined by the author and, thus, are subjective.

Casola et al. [3] provide an approach based on [2] and specified for SOAs. This approach focuses more on comparison of alternative security policies rather than on descriptions of them. Similarly to [2] the authors suppose that security policies are presented in a predefined way and a level of service is determined for every policy. Thus, every proposal is represented as a matrix where rows are the specified policies and columns are the level of service. Every cell contains a 1 if the level of service is guaranteed and a 0 if it is not. The author uses Euclidean distance among matrices in order to compare them.

The proposed approach allows estimating the distance (or difference) between several alternative security policies. On the other hand, the distance does not mean that one proposal is better than another one, because one alternative may have one set of policies which are better and another alternative have other more secure policies. Moreover, every policy is considered as equally important and, thus, it is impossible to estimate which proposal serves better the concrete needs of the user.

Krautsevich et al., [4] propose a qualitative approach based on risk assessment. The idea of the paper is to select the least risky provider. First, desirable and declared policies are mapped. Then, a client defines the relative importance of every policy for him. The next step is to determine the thresholds for the attributes used in the policy according to five relative security levels (unacceptable, low, medium, high, perfect). Then, the declared policies are evaluated using the thresholds and a level of strength is assigned to every policy. The next step is to multiply the importance level and security level for every policy. Finally, the number of policies at each risk level (no risk, low, medium, high, unacceptable) are determined. The comparison of different alternatives is performed by first comparing high risk numbers, then medium, and so on. The alternatives which have at least one unacceptable risk are immediately rejected.

The proposed method allows the comparison of alternatives automatically, once the thresholds and importance levels are defined. On the other hand, the proposed method is subjective and qualitative and, thus, does not allow an accurate analysis to be performed.

From the analysis of the various approaches illustrated in this section, we learn that most of the work aimed at defining quantitative and qualitative aspects of trust in distributed systems is based on policies specifiable by formal languages and verifiable by formal methodologies and tools. The Aniketos trust model will benefit from the connections between formal methods and trust definition and verification, in particular for rigorously define and analyse trust properties that the Aniketos platform should satisfy. An example of this approach is illustrated in Section 5.2.1, where the SxCxT

framework is presented. This framework is based on policies and contracts formally specified, and exploits a formal methodology to check if a contract matches a policy.

2.3.2 Compliance and certification approaches

Evaluation of security policies could be performed by a trusted third party. Such evaluation is often performed using a well-known security standard (like ISO 27001) or, in other cases, the assessment methodology can be a standard itself (e.g., Common Criteria [8] or SSE-CMM [10]). The certification process is performed by a Trusted Third party (TTP), which is trusted by all participants (as it is follows from the term). The TTP checks the target of evaluation itself, the provided documentation or performs the evaluation on-line depending on the certification process. If the target of evaluation satisfies all the required criteria a certificate is issued. This certificate can be used to prove to another party that the target of evaluation indeed meets some requirements. In other words, the external party is provided with the evidence that the claimed requirements (or policies) are guaranteed. Thus, the certificate helps to increase the trust in the target of evaluation (see Figure 2). Often different levels of certification are used to indicate how well the required criteria are satisfied. This level can be used in order to select the most trustworthy product.

The *ISO 2700x* family of standards [5] (based on BS7799) are well-known and often used to indicate the trustworthiness of a system. The most well-known standard in the family is ISO/IEC 27001 (ISO/IEC 27001:2005 – Information technology – Security techniques – Information security management systems - Requirements). The certification agencies (TTP) are available worldwide. Moreover, some software products are available in order to help checking compliance with the standard, like CRAMM [6]. CRAMM is a commercial tool provided by Siemens. The tool helps the analyst to define the scope of the Information Security Management System to be evaluated, conduct the gap analysis, perform the risk assessment (required by the standard) and prepare the improvement program.

The standard itself is well-known and widely-accepted as a trusted security guideline. On the other hand, the certification process is subjective, since the standard is very generic and it is up to the certification authority to judge if security requirements are achieved. Moreover, a 27001 certificate does not allow judging how well the standard is satisfied (unless a percentage of compliance metric is used, e.g. [7]).

Common Criteria [8] *(CC)* is a standard developed by the USA and Europe to align TCSEC and ITSEC. A product is evaluated against a Protection Profile (PP), a set of requirements for the corresponding category, and Security Target (ST), a set of security requirements for the identified product. Each evaluated product is checked to establish if it satisfies the PP and ST and receives an Evaluation Assurance Level (from EAL1 to EAL7) according to the strictness of the evaluation.

CC is a well-known standard that allows software to be selected based on a systematic and methodical process for ensuring security. On the other hand, the standard does not allow judging “how secure the software is itself”, but only “how rigorous the process was to make it secure”. The process of certification is also very subjective and depends on the certification agency. Finally, the Common Criteria can be used for system evaluation, but “the interactions between variety of products (hardware and software) are such that detailed evaluation is very close to a practical impossibility” [9].

The *System Security Engineering Capability Maturity Model (SSE-CMM)* [10] is a well-known method for assessment of a security engineering process. It is a part of the well-known CMM series. The SSE-CMM model has two dimensions: “domain” and “capability”. “Domain” is a set of “base practices” for security engineering which is grouped into process areas. The first half of the process areas is responsible for security and the remaining 11 process areas address the project domain, taken from a more general standard: Systems Engineering and Software Capability Maturity Model.

The “capability” dimension describes practices for process management and organizational capabilities. These practices are called “generic practices” because they are common for all domains. The generic practices describe activities which have to be performed during fulfilment of base

practices. The SSE-CMM assigns a level of maturity to the security system engineering process according to achievement of generic practices for all base practices.

The maturity level assigned by SSE-CMM certification helps to select the system which uses the most deliberate security process. On the other hand, this level does not mean that the system itself is more secure. For example, a scheduled and well documented assessment process does not mean that the process itself is performed correctly (and the conclusions are right). In other words, the well-established security management process is required for high security level, but the quality of its implementation is also important. Similar to other certification methods SSE-CMM highly depends on the certification agency.

VeriSign is a well known certification company which provides a very simple way to prove that a web site is trusted. VeriSign certification is issued to the web site which provides an approved (by VeriSign) SSL encryption and Authentication to its customers. Moreover, a daily malware scan has to be performed in order to guarantee that the site does not contain any malware installed by a hacker. As a result, a web site receives a well-known VeriSign mark, which assures the customers that the site satisfies all requirements for the certification. The certification is a very simple on-line process. The result of the certification (the mark) is easy to see (and understand its meaning) and has been acknowledged by the customers. On the other hand, the certification is very limited (SSL encryption, authentication and malware scanning) and can hardly be significantly extended in order to work with other security requirements.

According to Figure 2 certificates increase referral trust and, thus, certification is used for establishing trust in ANIKETOS. For example, usage of a tool with EAL 3 (Common Criteria) should increase trust in the overall service, in comparison with other services which use uncertified products. Therefore, available certificates contribute directly to computation of trustworthiness value in D2.2. Certification could be used as a pattern for trust establishment (see Section 3.2.2) and as an essential part for security-by-contract (see Section 5.2.1).

2.3.3 Reputation and QoS based trust models

This subsection describes popular models that consider reputation and/or QoS as approaches to evaluate trustworthiness. Note that some researchers use reputation and QoS concepts interchangeably or use QoS metrics as the source of reputation information. In Aniketos, reputation is focused on user feedback on the service quality.

2.3.3.1 Online reputation models

Several websites allow users to provide product and service rating as part of reputation systems that aim to support a commercial activity. Examples of such websites include eBay, Amazon, Epionions, BizRate. On eBay users can give one of three ratings for a product after they buy it; positive (+1), negative (-1), or neutral (0) in addition to short comments. The reputation value is computed as the sum of these ratings over a past period such as six months including a percentage of the sum of ratings relative to the overall number of ratings. Similarly, Amazon uses the mean value of the ratings to calculate the reputation value. These centralised reputation systems are used to remove the inherent risk associated with online interactions involving unknown buyers and sellers. Reputation systems are also used in some discussion forums e.g. Slashdot and in email filtering systems e.g. Bayesian filtering [58].

The marketplace reputation systems are generally primitive and have several weaknesses such as:

- No consideration of context of the ratings and the trust e.g. a bad rating of a service can be caused by low quality product, late delivery, or insecure payment method.
- No specific mechanism to deal with dishonest ratings.
- Correlations between ratings indicating reciprocation and retaliation in the two way ratings.

Despite these weaknesses these commercial reputation systems receive significant user contribution and they provide a considerable positive impact on the marketplace as seen in eBay's success. The models generate sufficient trust among buyers to convince them to assume the risk of carrying out transactions with complete strangers [69].

SPORAS [18] is a centralised reputation system that extends the online reputation models such as those used in eBay and Amazon by introducing a new method for rating aggregation after each transaction. In order to make accurate predictions of user behaviour, a recursive and adaptive algorithm for updating reputation is used without the need to store all ratings. Reputation is calculated continuously using the previous value of reputation; the previous value of reputation may be increased or reduced depending on the new ratings. This aggregation method also allows newer ratings to have more effect on the reputation.

The SPORAS model has more advanced characteristics to model the trust dynamics than the existing commercial online models. In order to prevent a user with a bad reputation from leaving the community and entering with a fresh reputation, the reputation of a new user is the lowest reputation possible. However, this penalises newcomers and may discourage them from participating in the community. In SPORAS the reputation values are dependent on the reputation of the entity that is providing feedback; however, it mixes two reputation contexts. While a user can be very trustworthy in deals, nevertheless that same user may be a dishonest rater of other users.

PageRank [24], on which Google's search engine is based, was designed to rank search engine results based on a measure of the number and weight of links to a Web page from other pages, which in essence is a reputation system. The score applied to a page is the sum of the normalised weights of links pointing to it, the weight of a link itself based on the score of the referring page.

The main strength of PageRank is that votes are weighted by the score of the page making the referral. Attempts to "ballot stuff" – i.e. to increase score by creating otherwise useless pages with links to the target page – are thwarted by the low weighting that such pages will get. Two limitations can be identified; firstly that the scheme is inherently centralised (which may or may not be a weakness when applied elsewhere), and secondly that it is designed for a specific purpose and may not be easily applicable to a multi-service or composed service environment.

In one of the earlier papers that advocates moving away from 'hard' trust models to models that are based on distributed recommendations [25], Abdul-Rahman and Hailes propose a protocol for handling recommendations from which trust is derived, and a scheme for parties to build up a view of global trust using transitivity. They distinguish between *direct trust* and *recommender trust* (i.e. trusting the quality of another party's recommendations). Trust is modelled using discrete levels (integers ranging from -1 to +4).

Using a limited number of discrete levels to represent trust adds to simplicity, but is perhaps less flexible than a continuous representation. It needs to be examined how the robustness of this scheme can be improved in the event of colluding malicious parties.

2.3.3.2 Models for P2P and multi-agent systems

PeerTrust [15] is a system that uses reputation to evaluate the trustworthiness of peers. The system includes:

- an adaptive trust model for quantifying and comparing the trustworthiness of peers based on the reputation formed through a transaction feedback (experience) system, and
- a decentralized implementation of such a model over a structured P2P network.

PeerTrust takes into consideration potential threats a reputation system may be subjected to such as adaptive peers. Adaptive peers change their malicious strategy based on the pattern on which the reputation algorithm calculates reputation values e.g. occasional cheating that doesn't significantly harm their reputation. An adaptive time window algorithm is used making the reputation of a peer hard to build and easy to lose.

In PeerTrust, peers use a personalised similarity measure to give more weight to opinions of peers who have provided similar ratings for a common set of past providers. However, in a dynamic large P2P system, finding such a set of partners may be difficult. As a result peers will likely have to select between peers for which there's no information.

REGRET [14] relies on direct experience, witness, and social information to decide on the trustworthiness (reputation) of an interacting agent in a multi-agent system while also taking context (i.e. what to trust in?) into consideration. In REGRET there are three dimensions of reputation;

- Individual reputation based on an agent's own information about another agent. This reputation value is calculated as a weighted mean of an agent's ratings giving more weight to more recent ratings. A reliability value is also calculated based on the quantity and variation between ratings.
- Social reputation derived from public information about an agent. This includes the reputation of the agent's group and reputation information from the trusting agent's (i.e. trustor) group relating to the other agent and its group.
- Context dependent reputation or ontological dimension derived from contextual information.

The REGRET system has limitations due to its requirement of a minimum number of interactions to reliably evaluate the reputation of an agent. It also does not describe how to build the social network which the model heavily depends upon. Additionally, it does not consider the problem of dishonest ratings. However it has made a step forward in the knowledge of reputation systems and their evolution.

The *FIRE* model [16] combines multiple trust sources namely, direct experience, witness information, role-based rules, and references. References can be produced by agents that have previously interacted with the target agent certifying its behaviour. Other factors considered in the trust calculation include context, credibility and recency of information.

FIRE integrates the four sources to provide trust metrics in a variety of situations including where there is scarcity of information about an interacting agent and to enhance the precision of the trust model. However, the model lacks mechanisms to deal with dishonest or inaccurate agents providing false witness or reference information.

According to the authors of *PowerTrust* [17] the relationship between users and feedback on eBay follows a power-law distribution. It utilizes the observation that most feedback comes from a few power nodes to construct a robust and scalable trust modelling scheme. In PowerTrust, nodes (peers) rate each interaction and compute local trust values. These values are then aggregated to evaluate global reputation through random walks in the system. Once power nodes are identified, they are used in a subsequent look-ahead random walk that is based on a Markov chain to update the global reputation values. Power nodes are used to assess the reputation of providers in a system-wide absolute manner.

PowerTrust provides a fast and scalable reputation system that is resistant to malicious behaviour. However, it requires a structured overlay DHT P2P system, and the algorithms are dependent on this architecture. Service-oriented environments or the Web in general do not exhibit such structure.

The *EigenTrust* [23] algorithm is an example of a reputation scheme for peer-to-peer systems that attempts to estimate a global, shared view of reputation. Its original motivation was to reduce the effect of flooding of file sharing networks with inauthentic files (such as bogus music files released onto file sharing networks by record labels).

Each peer i maintains a local trust value, c_{ij} relating to its experience of peer j. Each time peer i downloads an authentic file from peer j, c_{ij} increases. Each time peer i downloads a bogus file from peer j, c_{ij} decreases. Local trust values are normalised, to reduce the effect of dishonest ratings, by forcing:

$$\sum_j c_{ij} = 1$$

These values are then aggregated, by applying trust transitivity in an iterative manner, to provide a global view of trust. One proposed way to do this in a distributed fashion includes having each peer submit local trust values relating to peer *i* to designated *score managers* for peer *i*. Then these score managers vote on peer *i*'s score, with a view to outvoting malicious peers (or even malicious score managers).

This system is designed to work robustly in the presence of malicious nodes. Trust/reputation is modelled using a simple scalar probabilistic value, which is reasonable for a single service scenario (such as file sharing). A richer representation (i.e. a vector) of trust may be required though for more complex cases where there are many services with different requirements.

2.3.3.3 Models for services

RATEWeb [19] is a framework for establishing trust in service-oriented environments. It supports a collaborative model in which Web services share their experiences of the service providers with their peers through QoS feedback ratings. The different ratings are aggregated to derive a service provider's reputation. The concept of reputation is used interchangeably with QoS (called QoWS) in RATEWeb. This in turn is used to evaluate trust. RATEWeb aims to support trust-based selection and composition of Web services. It uses the concept of communities to cluster Web services based on domains of interest. Communities are also used to bootstrap the reputation of newcomers and setting policies relating to members when it decreases below certain levels. The framework extends traditional models of service selection based on functionality criteria to include the reputation of service providers. At the end of an interaction a service consumer rates the provider according to predetermined criteria. These ratings are then used to update the provider's reputation. In cases where ratings are inadequate, statistical forecasting using a Hidden Markov Model is used. RATEWeb also deals with issues relating to rating recency (temporal sensitivity), personalised preferences and rater credibility. Rater credibility is called feedback trust as opposed to service trust.

RATEWeb provides a thorough framework and mechanisms for dealing with a variety of problems that face reputation systems as described above. However, RATEWeb is a theoretical model that does not discuss how it can be implemented in real-world Web services or how it relates to existing and future Web service standards. It also does not provide mechanisms for responding to dynamic changes in the environment that for instance may affect a provider's trustworthiness. Reputation mechanisms that support service composition (e.g. aggregation of component service reputations) are not detailed. Additionally, the framework does not provide a mechanism to differentiate between accidental and planned changes in a provider's behaviour e.g. periodic malicious behaviour may not have a major effect on a provider's reputation.

The reputation-based trust model [20, 21] is based on a shared conceptualisation of QoS attributes. Their architecture uses service agents that conceptually reside between the service and its consumer. The consumer uses one agent per Web service interface. This architecture extends the SOA with partly automated software agents. Service brokers are extended with agencies that facilitate the sharing of data among the participating agents. The QoS specifications are modelled as an ontology that allows services to be matched semantically and dynamically. The semantic matching allows the service agent to match consumers to services using the provider's advertised QoS policy for the services and the consumers' QoS preferences. The provider policy and consumer preferences are expressed using the concepts in the ontology.

In the model all agents that report reputation ratings are assumed to be trustworthy and no mechanism is proposed to determine the credibility of the raters. Likewise, the common agencies to which these ratings are communicated for sharing and aggregation are also assumed to behave honestly. We note also that the feedback reporting cannot be automated since human participation is necessary for rating Web services.

2.3.3.4 Summary of strengths and weakness in trust models

Table 1 summarises the strengths and weaknesses of the trust models discussed from the perspective of Aniketos.

Category	***Description and Examples***
Centralised vs. decentralised system	Some trust models and systems are centralised (e.g. eBay) while others are decentralised in a P2P fashion with trust computations performed independently at each node e.g. REGRET [14]. Centralisation and decentralisation both have strengths and weaknesses but the choice of central or distributed system depends on the implementation. Often centralised systems are thought to represent a single point of failure and lack scalability. However, this may be avoided if proper redundancy and clustering is provided. Decentralised systems may be more vulnerable to malicious behaviour due to lack of central control unless adequate countermeasures are in place. Commercial systems such as eBay and Amazon are often centralised. The tradeoffs between centralisation and decentralisation also apply to Aniketos. The Aniketos platform does not require either approach. However, at least partial centralisation of aggregation and management of trustworthiness data may be needed to allow a broad view and easy access to this data.
Subjective vs. objective evaluation	Trust measurement can be based on objective or subjective measurements. For example, some systems rely on monitored QoS attributes e.g. RATEWeb [19], while others use subjective user ratings e.g. eBay. Combinations of both QoS and ratings have also been considered e.g. [59]. Subjective ratings may be vulnerable to abuse by dishonest or malicious users. In Aniketos both subjective (e.g. reputation) and objective (e.g. QoS) data will be used for the trustworthiness evaluation. However, preferences allow adjustments to which type of data is more significant in the evaluation. The need for subjective data arises due to the difficulty in predicting users' perception of the trustworthiness and quality of a service.
Trust context	A trust context specifies what to trust an entity for. A trust model may provide trustworthiness measurements in a specific context e.g. performance, or allow customisation of the context e.g. RATEWeb. The majority of trust models use a general context of trust e.g. SPORAS [18], REGRET, FIRE [16]. eBay and some other systems allow comments which helps users understand the contexts of other users' ratings. In Aniketos, context is customisable through adjusting the significance of categories of metrics.
Requirement for past experience, trustworthiness bootstrapping, and scarce information.	Some trust models attempt to predict trustworthiness even when there is no or scarce information about an entity. REGRET for example requires a minimum number of interactions. Several models provide a default trustworthiness for newcomers which may discourage newcomers in joining (low trustworthiness) or encourage whitewashing (bad entities rejoin with new identities). RATEWeb suggests an extension of the Web services models to support communities. Those communities help determine the trustworthiness of newcomers. Endorsement is another approach used to help improve the trustworthiness level of newcomers. Some work suggests the use of machine learning mechanisms e.g. Hidden Markov model (HMM), to predict trustworthiness in cases of scarce

Category	*Description and Examples*
	information. In Aniketos, in cases where no historical monitoring and feedback data is available; basic trustworthiness evaluation of all services is made using the services' claimed trustworthiness properties in service contracts. However, the credibility (confidence) level will be low depending on the predictability of the service. Runtime monitoring of trustworthiness helps to more credibly (actively) evaluate how trustworthy services and compositions are.
Trust confidence evaluated separately	Trust confidence or credibility may be used together with the trustworthiness score to evaluate how reliable this trustworthiness score is. Several metrics can be used such as quantity and quality of information about the trusted entity and the recency of the information. Fore example, FIRE suggests the use of deviation between rating values and the number of ratings in this evaluation. Some systems do not evaluate trust confidence e.g. SPORAS. Confidence levels will also be used in Aniketos as part of the composite service trustworthiness evaluation.
Rater credibility	Several trust models do not consider credibility and honesty of raters e.g. REGRET, while others provide mechanisms to minimise the influence of dishonest or malicious ratings e.g. using majority opinion as in RATEWeb. Rater credibility countermeasures are considered in Aniketos WP4 as in D4.1. WP2 will implement measures that ensure credibility of ratings as in the guidelines in Section 3.2 particularly the "rating credibility" pattern.
Protection from malicious behaviour	Some trust models attempt to provide protection against several techniques of malicious behaviour such as Sybil attacks (multiple identities), DDoS, self-promotion, collusions. Vulnerability to such attacks may depend on the architecture of the model and the mechanisms used in evaluating trustworthiness. For example Sybil attacks are common in P2P systems. Protection from malicious behaviour and security threats are considered in Aniketos WP4 but WP2 will provide techniques to prevent and counteract those threats as in the patterns and guidelines specified in Section 3.2.
Aggregation of ratings	A trust model may sequentially incorporate new ratings and metrics into the trustworthiness level without maintaining a history of those ratings e.g. SPORAS. More recent models usually consider all ratings based on their recency and credibility e.g. RATEWeb. Although calculating trustworthiness level based on all ratings is more accurate and more flexible, it is computationally more expensive.
Support for recency	More recent ratings are usually more indicative of the trustworthiness of an entity (e.g. service) and hence they should carry more weight than older ratings. Recency (also called temporal sensitivity) is considered in models developed more recently such as FIRE and RATEWeb.
Representation of trustworthiness level	Existing trust models provide varying ways of providing numerical representation of trustworthiness level. For example in FIRE a trust score ranges between -1 and +1 and confidence between 0 and 1. Traditional systems such as eBay; provide a simple sum of ratings which may be any integer value.
Deployment and impact	Several reputation based systems are used on the Web such as on eBay, Amazon, and PageRank (Google Search) with significant role in the success of the organisations. However, many of the advanced systems and

Category	*Description and Examples*
	models are not in commercial operation e.g. FIRE, RATEWeb.
Implementability	Trust models may use existing infrastructures, standards, etc. to support trustworthiness evaluation and utilisation while others require extensions or modifications e.g. RATEWeb requires the use of the concept of communities.
Scalability and performance	Scalability in centralised systems can be provided by clustering and redundancy support. Aggregation of large numbers of ratings to evaluate trustworthiness may affect the performance of a trust management system if a mechanism to deal with this case is not provided.

Table 1: Summary of strengths and weakness in reputation and QoS based trust models

3 Trust from a socio-technical perspective

There are various ways to treat the social in a socio-technical perspective. This chapter complements the technical mechanisms for establishing trust with *user-centric* mechanisms for establishing trust among end-users of composite services. This area aims to help service developers in designing and running systems that are trustworthy.

3.1 User-centric trustworthy service composition

This section takes a user-centred design perspective [60] on Aniketos. The users we focus on are the software designers and developers who will develop and compose services for the Aniketos platform. The central question we address is: how do developers currently choose software components?

3.1.1 Hypotheses for domain-specific trustworthiness factors

A special focus in Aniketos is on how people involved in composing black-box services may draw inferences about trustworthiness. A series of knowledge-rich domain-specific factors will be involved which are not addressed by existing social and cognitive models of trust. A first sketch of likely important factors is:

- *Certificates* that particular security properties have been verified. The issue here is whether the certifying authority, and the tools the authority uses, may be trusted to do the verification. There is also the more superficial issue of whether a seal is used to claim trustworthiness, which is often taken with a grain of salt [43].
- *Time/space complexity* of services used. Although the code is not open to inspection, its complexity may be inferred from performance, e.g., how much time it takes to run a query as a function of input complexity; what impact high service usage has on individual users (using timing information has been proposed as an attack strategy by Kocher [38]).
- *Architecture complexity of the composition.* This will be visible to developers and could well reveal untrustworthy links.
    - *Algorithm complexity.* For instance compare a reputation metric which is only a count, with a metric like EigenTrust [37] which computes the left principal eigenvector of a matrix of local trust values. The higher-level description of what a service does, e.g., as exposed in documentation could conceivably give sufficient details to infer the likelihood of an implementation error.
- *How/when service updates are performed.* There is a trade-off between timely updates which repair security issues and updates which impact on QoS.
- *How well maintained* the service is, e.g., when was the last update; how frequently updated.
- *Transparency of software development process*. How exactly was the component developed, for instance is it open source and can discussions between developers be read?
- *Degree of personal contact with developers.* We already alluded to this above; such contact is likely to affect trustworthiness through similar processes as it does with any human-human contact.
- *Whether formal verification was used.* The formal methods community often asserts that formal methods are unequivocally good. However, even formal proofs of trustworthiness properties need not imply objective trustworthiness. Firstly, the wrong properties, with respect to requirements, may have been proved or important properties omitted. Secondly, the logic, proof procedures, or proof checkers might be unsound. Occasionally logics are shown to contain paradoxes, most famously Russell's paradox, which allow all conjectures, even those which are false, to be proved in a logic by Frege [70]; more recently Girard found a paradox in a logic by Martin-Löf [71], later

versions of which are used in formal provers. Bugs are occasionally found in provers, especially in difficult to write aspects such as keeping track of bound variables [73]. Thirdly, people might not trust a proof they do not understand. The use of formal methods is likely to affect perceived trustworthiness.

- *Quality of documentation.* In order to use a service the interface must be documented. Its quality can therefore allow trustworthiness to be inferred.
- *How often a service has been used.* Related to reputation; a well-used service is likely to signal trustworthiness, e.g., because it is more likely that bugs will have been detected.
- *Security incident history.* Also related to reputation; if a service is plagued with problems then that is likely to have a detrimental effect, though inferences might be affected by how often updates are performed and also how crucial the service is to a composed service.

There is an inherent conflict in all cases between service providers (the trustees), who will be motivated to falsely signal trustworthiness, and service users (the trustors) who will want to ensure they trust only when it is valid to do so. However trustors may be open to various biases, e.g., trust certified services even though they do not know what that certification entails (as discussed above).

3.1.2 Material and methods

After reviewing the literature we sought to investigate empirically how designers and developers think about trust, focussing on the action of choosing a software component. Section 2.1 provided a range of hypotheses for the processes involved. Starting with the guiding predicate, $trust(X, Y, c, \tau, g_x)$, the following entities are involved in the trust inference process:

- a software developer (or developers), X, trusts
- a software component, Y, developed by another developer (or developers) Z
- to perform a sequence of tasks τ
- in context c
- leading (X hopes) to the satisfaction of X's goal g_x.

3.1.2.1 Guiding research questions

As a first broad-brush approach, the following general questions were formulated, driven by the framework introduced above.

1. Since goals are central to the approach of Castelfranchi and Falcone [30], how do software developers and designers describe the goals they are trying to achieve using a software component?
2. Continuing discussions between Aniketos partners, and surveys of the literature, have uncovered a variety of notions of "trust" and "trustworthy" – but what concepts do the terms evoke in software developers?
3. Given the hypothesis that trust in cognitive trustees is qualitatively different to trust in non-cognitive trustees, how often do software developers actually get in contact with the developers of a component? If and when they do, what is the nature of the communication?
4. A range of domain-specific factors are very likely to be important. What factors do developers see as important in the context of choosing software components?
5. Although trust and trustworthiness are multifaceted constructs, not all easily placed on dimensions, is it possible to get assessments on a linear scale of how trustworthy a component is inferred to be?

6. It's likely that more than one software developer is involved in making a decision about trustworthiness. Does the need to debate a decision with others influence how the decision is made?
7. To what degree is trust context-dependent? For instance is it possible for developers to come up with ideas for contexts when a component will and will not be trustworthy? What do those ideas tend to be?
8. What factors cause developers to change their mind about the trustworthiness of a component?
9. To what extent does the importance of a component in a project influence the process by which its trustworthiness is assessed?
10. How does trustworthiness relate to monitoring?
11. How do developers reason about what they ought to do to determine how trustworthy a component is (i.e., the *regulative norms* of their domain)? If they believe they should have done something but for whatever reason were unable to do so, this might give clues for areas where tool support would be beneficial.

3.1.2.2 Method

We designed a Web-based questionnaire to investigate how developers reason about the trustworthiness of software components. Although Section 3.1.1 lists a series of factors we speculate would influence trustworthiness, rather than bias participants, we opted for a semi-structured questionnaire with mostly open questions.

Participants were informed that the survey is "for software designers and developers who have ever had to choose a third-party software component for a project they worked on", and that we are investigating "how people designing and developing software decide whether a software component is trustworthy."

The questions asked were as follows:

1. Consider a time you have chosen to use a software component. What was its function?
2. In what programming language(s) was it written (leave blank if you don't know)?
3. "Trust" and "trustworthiness" have many different meanings. In the context of the software component you have in mind, what exactly did you trust it to do (or not to do)?
4. Did you have any contact with the developers of the software component? [Yes/no.]
5. How did you get in contact?
6. Why did you communicate with the developers? Briefly describe what you discussed.
7. What kind of software did the component become part of?
8. Approximately how many people (including you) worked on the project?
9. Why did you choose to use the selected component and not another component? Briefly describe any reasoning behind the decision.
10. With whom did you speak to help decide whether to use the component? (Leave blank if nobody.)
11. Did you perform any kind of real-time monitoring or checking of the component in the live project?
12. Please describe the real-time monitoring or checking that was performed.
13. Is there a situation or context in which you would not trust the component? [Yes/no.]
14. In what situation/context?

15. How trustworthy do you think the component is? [Rated from 0 = completely untrustworthy to 5 = completely trustworthy on a discrete scale.]
16. What (if anything) would have changed your mind about the trustworthiness of the component?
17. What would the consequences have been if the component had turned out not to be trustworthy? (Try to be as specific as possible.)
18. In retrospect, is there anything you believe you should have done to decide how trustworthy the component was?
19. Do you have any general comments you think might be of relevance?

A series of demographic questions were also asked, results of which will be summarised below.

3.1.3 Participants

A variety of methods were used to try to recruit participants. Software companies were contacted directly, though this was only successful in a couple of occasions. A large online forum used by software developers was asked if they could advertise the study, however they refused as they argued their site is "not an avenue to solicit developers". In the end, most participants came from social networks established by Aniketos. Participants were invited to take part by email. The main selection criterion was that participants were software developers or designers who had ever had to choose a software component. A total of 51 people started to complete the questionnaire. Of those, around half went on to complete the survey; the rest opted out, having completed only their basic demographic details. Table 2 summarizes the sample. Figure 4 shows the distribution of respondents' age (a), years in current position (b), years in software industry/research (c), and team size for the project using the component under discussion (d). Figure 5 shows respondents' country of residence. Around 80% of respondents had "research" or "researcher" as part of their current job title. Many of those had previously worked in the software industry or currently do software related research. The remainder were: a chief technology officer, a senior IT consultant, a software engineer, and a manager.

	Completed		***Opted out***	
N	25		26	
Female *N*	5		4	
Male *N*	20		22	
Median age (*SD*)	33.0	(5.3)	34.5	(8.1)
Median years in current position (*SD*)	4.0	(2.7)	4.0	(5.7)
Median years in software (*SD*)	9.0	(5.1)	11.0	(6.4)

Table 2: Basic demographic details

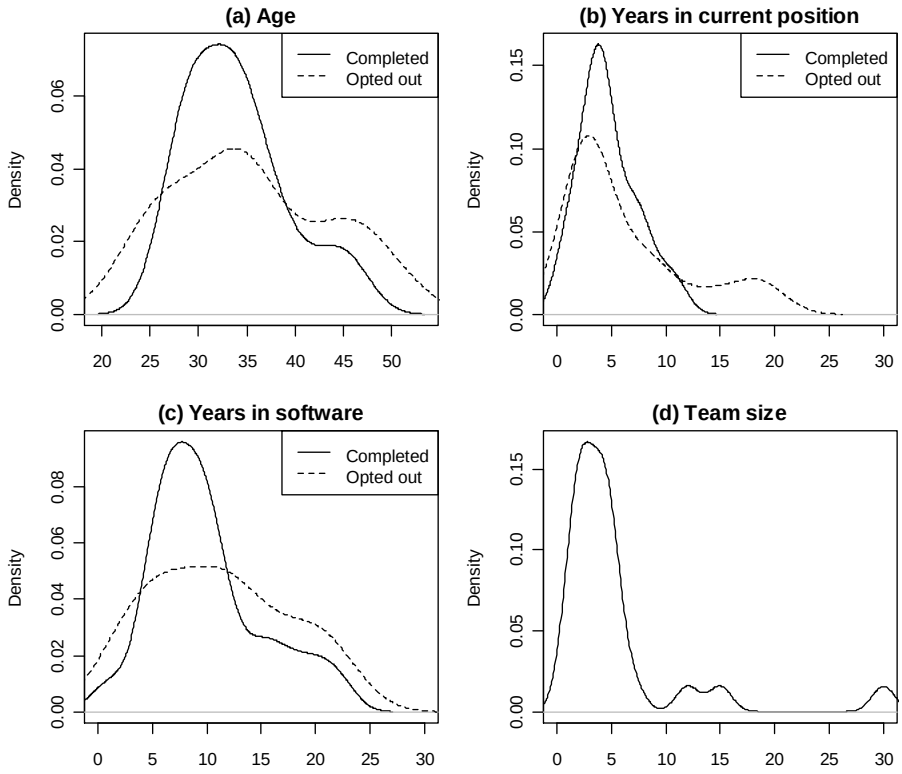


Figure 4: Distribution of respondents' age and years of experience

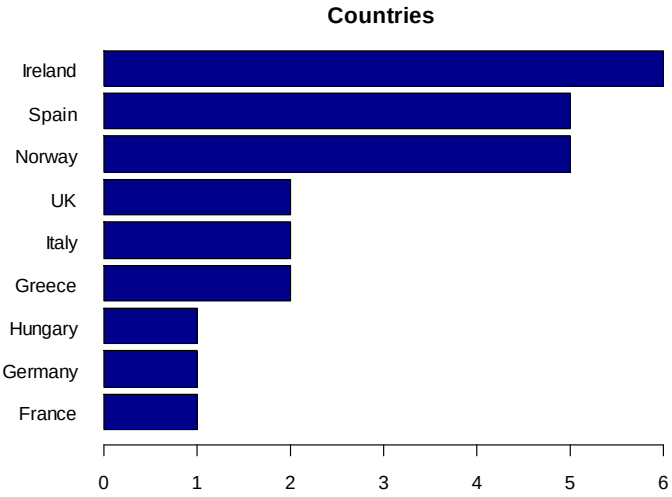


Figure 5: Respondents' current country of residence

3.1.4 Results

3.1.4.1 Software mentioned

A range of software products were covered, including: database and semantic storage applications, banking software, editors, network management software, data analysis software, encryption related products, and simulation software.

3.1.4.2 Trustworthiness factors

Table 3 summarises the main factors to which respondents referred when stating what exactly they trust a component to do. By far the most common responses concerned specific functionality; other

factors were commonly related to "non-functional" requirements, such as reliability, security, and performance.

Factor	*Number of respondents*	*Comments and examples responses*
Functionality	19	Generate (suitably) pseudorandom numbers such that a private key could not be guessed based on poor randomisation "correct implementation of communication standards" "parse and store my data … reliably"
Reliability	6	"only do what I told it to do"
Performance/scalability	6	"time of response" "to be able to cope with a large number of queries"
Security	5	"not modifying data and not disclosing that data to third parties" "Perform the search without affecting the integrity of the data being searched"
Consistency with specification/documentation	4	"Work as expected based on the documentation" "be correctly implemented according to the specification/API" "We did some test in order to see if it really did what it was supposed" "To perform what the tool providers claimed to offer"
Stability	1	"other alternatives were unstable development … prototypes"
Maturity of development	1	"other alternatives were … research prototypes"
Clarity of documentation	1	"it had clear interfaces"
Simplicity	1	One respondent noted that a simpler solution would have been better

Table 3: Factors influencing trustworthiness

3.1.4.3 Processes for inferring trustworthiness

There is some overlap between the processes used to infer trustworthiness and the trustworthiness factors already covered, so here we focus on aspects not already mentioned. See Table 4.

Action	*Comments and examples responses*
Testing	One respondent was very clear of the importance that software "passes all the automated unit tests […] This is true for software we write or purchase, as well." "Rough performance measurements" "Checking input and output results using debugging tools"
Inspecting source code	"… could see how it worked"
Speaking to colleagues	For instance team members, project partners, and boss or project/team leader

Action	*Comments and examples responses*
Considering reputation	"…positive feedback regarding the component on developer forums and mailing lists"
	"Good reputation in the developers community"
	"As a well-established [component] for text searching with a good reputation, we expected it to produce accurate results"
	"The component was widely adopted by the community"
	Cited "in a (peer-reviewed) technical paper"

Table 4: How respondents infer trustworthiness

3.1.4.4 What additional actions did respondents think could have been taken?

Table 5 shows what developers would have done, had they had the time and resources. Testing was again seen as important, as was examining source code. Interestingly monitoring was suggested, which would indicate a lack of trust.

Action	*Comments and examples responses*
More testing	"More consideration of performance"
	"More testing could have been performed (e.g. to check it's read only, and that it provides the correct results)."
	"…some kind of testing with typical injection attacks on the finished product"
	"Interface testing"
	"…memory usage should have been checked"
	"Written tests to assure correct behaviour for some realistic cases."
	"A set of software tests would assist in determining if the component had unintended consequences within it given functional parameters."
Examine source code	"more thoroughly examined the source code"
	"… possibly removing all elements that aren't necessary for our task"
Checked reputation	"I could have looked further for comments/feedback by its users"
Monitoring	"I could have monitored its operation more closely"
Contacted developers	"Contact with the developers"
More time/effort searching/deciding	"Spend more time on investigation, not choose the first component / lib that fit."
	Searched for a simpler solution

Table 5: Additional actions which could have been taken

3.1.4.5 Leaps of faith

One of the questions asked, "How trustworthy do you think the component is?" on a scale from 0 to 5, where 5 denotes completely trustworthy and 0 denotes completely untrustworthy. A central claim of trust theories is that there must always be some uncertainty, otherwise trust would be unnecessary. This is reflected in the responses people gave (see Figure 6), with most respondents giving the assessment 4 out of 5.

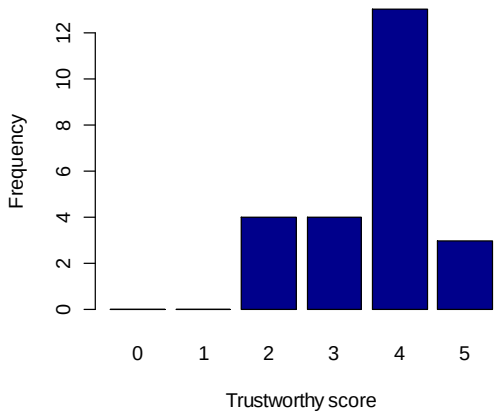


Figure 6: Trustworthiness estimates for selected components
Note: 5 = completely trustworthy, 0 = completely untrustworthy

Several respondents chose to use a component, even though they gave it a trustworthiness score of only 2 or 3 out of 5. It is not clear from most responses why this was, however in one case the component's results were always checked, and it was possible to infer whether the result was correct. For this one case, the existence of a component which produced any result at all was more important than a component which was always trustworthy.

3.1.4.6 Contact with developers

We expected some difference between those who contacted the original developers of a component versus those who did not as in the former case the trustee would be a cognitive agent. Only 8 respondents had any contact with the developers, versus 17 who did not. Contact was via mailing lists and forums (3), email (3), bug tracking systems (1), training courses (1). There was a small but statistically significant ($t(9.9) = 2.3$, $p = .047$; non-parametric Wilcoxon rank sum test $W = 94$, $p = .048$) difference between the trustworthiness score produced by those who had contact with the component developers versus those who did not: those who did have contact gave a slightly lower score (see Figure 7). In the Figure the error bars denote 95% confidence intervals of the means, computed using the t-distribution.

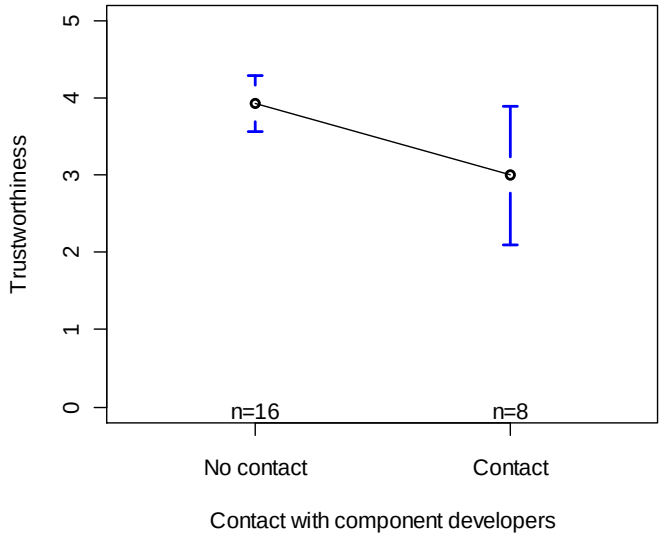


Figure 7: Relationship between whether the developer had contact with the component developers and mean trustworthiness score

There is a plausible explanation for this difference: all of the contact concerned *problems*, e.g.,

- Issues installing software
- Incompatibility between different library versions
- Configuration difficulties

- Errors in documentation
- Performance problems
- Missing functionality
- Asking for added functionality
- Bug reports
- Requiring help modifying code (i.e., because it didn't quite do what was wanted).

The act of contacting the developers tends to be as a consequence of inferred untrustworthiness.

3.1.5 Implications of results for a trust model

3.1.5.1 Trust as a secondary issue

We found evidence that respondents tended to focus on choosing components which seemed to achieve their primary functional goals. This corresponds with results found elsewhere, e.g., [61]

> "Trustors initiate transactions with a primary goal in mind – order the birthday gift that is delivered in time, to pay a credit card bill before incurring a penalty. Discharging the tasks required to reach these goals in a secure manner – i.e. a way that does not put your credentials at risk – is a secondary goal."

3.1.5.2 Suggestions for metrics

One possible trustworthiness metric we hadn't fully appreciated the importance of is some way to communicate the kinds of tests that have been performed. This is unlikely to be an integer but rather a structured set of cases. It might also be helpful to supply unit test parameters and code scaffolds to aid the developer in testing the trustworthiness of a component for him or herself.

One respondent felt so strongly about the importance of testing that he emailed to complain that we hadn't used the term in our survey.

3.1.5.3 Process support

The most common response to the question, "In retrospect, is there anything you believe you should have done to decide how trustworthy the component was?" was "More testing". It seems clear that any tool which aids testing would be welcomed by users.

It is interesting that all contact with third-party developers concerned problems. One thought is that a market place for finding services could also encourage contact when there is no problem. A novel hypothesis – which to our knowledge has not yet been tested – is that this will increase the trustworthiness of components produced.

3.1.5.4 How trustworthy is trustworthy enough?

As 8 participants demonstrated, a high trustworthiness score is not necessary for choosing to use a component. For instance an untrustworthy component might be chosen if it does deliver results, those results are useful – especially when the component runs offline or can be monitored cost-effectively, and where it is possible to tell at run-time whether a correct result was delivered.

3.1.6 Hypothesized model structure of factors influencing trust in a component

One desirable goal is to create a probabilistic model predicting subjective trustworthiness as a function of component cues and actions taken to test the trustworthiness. Note the "subjective" here: it could well be that the model would predict that a component would be inferred to be trustworthy even when it is objectively not – often obviously not, e.g., with respect to guidelines for how to infer trustworthiness. Figure 8 shows a structure that such a model could take, however the final model will depend on what information is available on the Aniketos platform.

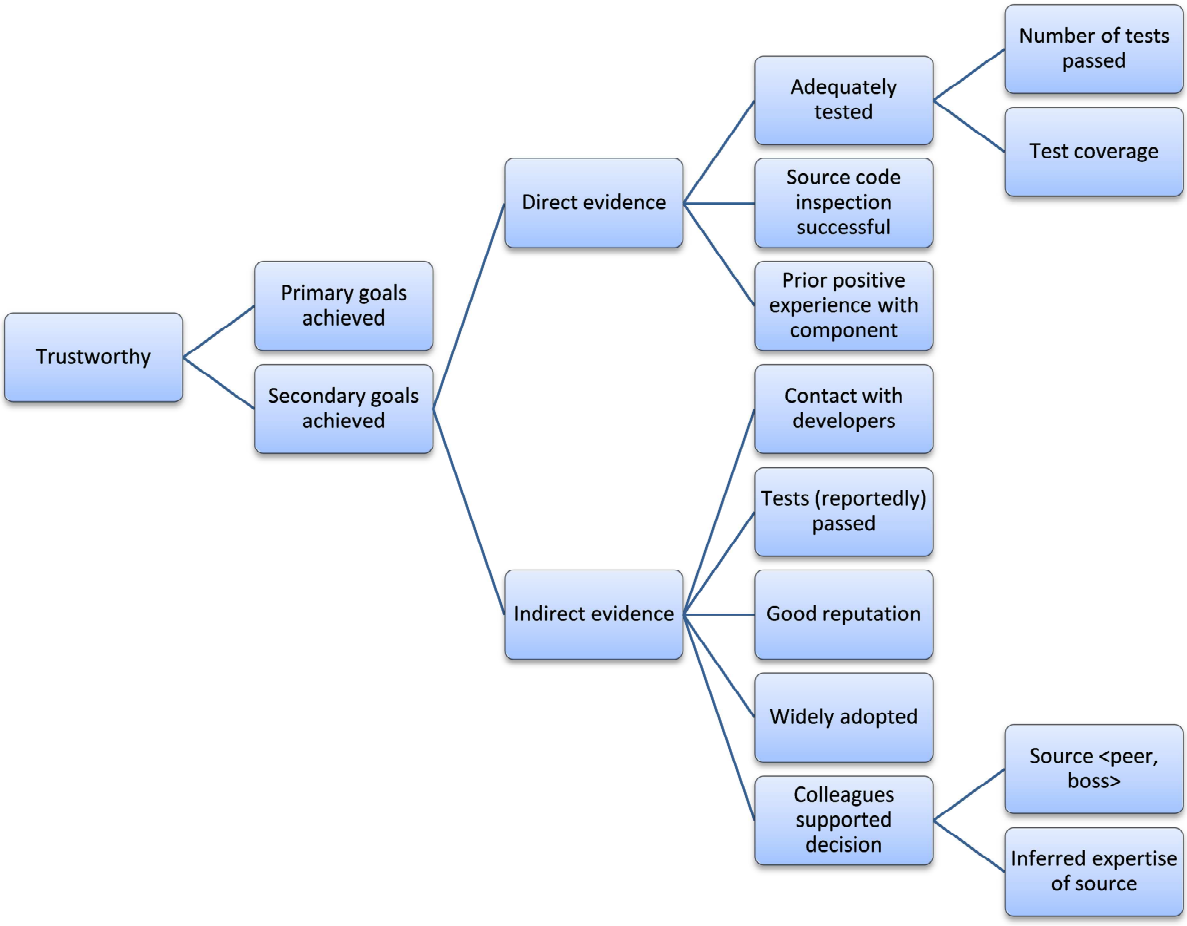


Figure 8: Structure for model hypothesized to predict inferred subjective trustworthiness

In order to make this probabilistic, the conditional probability of each node given its parents needs to be calculated. Then the joint probability distribution, *P(X)*, is given by

$$P(X) = \prod_{n \in Node} P(X_n | X_{Parents(n)})$$

Updating this distribution would be possible given metrics for all the nodes, some of which can be fairly automatically obtained (e.g., the number of automatic tests passed and how widely adopted the component is). Others require a subjective judgement on the part of the developer, e.g., a trustworthiness score from 0 to 5, as we described above.

Decisions are likely to be constrained by issues of, e.g., cost and whether monitoring is possible, suggesting a trustworthiness network also ought to take these factors into account (see Figure 9). So for instance, even though a component might not often be chosen, this need not necessarily imply that it is perceived as untrustworthy.

The question arises of how these sorts of models might be applied to composite services. Again the issue is statistically to infer what factors influence the choice of a component. To do this, nodes would be added to represent the trustworthiness factors of the services of which a composite service is composed. It could well be the case that service consumers are unaware of these factors, so there would be no influence. Or it could be that probabilities are propagated from the weak link in a series of composite services making it possible to diagnose which of the composed services is responsible. The exact details of this need to be refined in light of the technical developments of the Aniketos platform and will depend greatly on what information concerning the services composed to make a composite service is made available to service consumers.

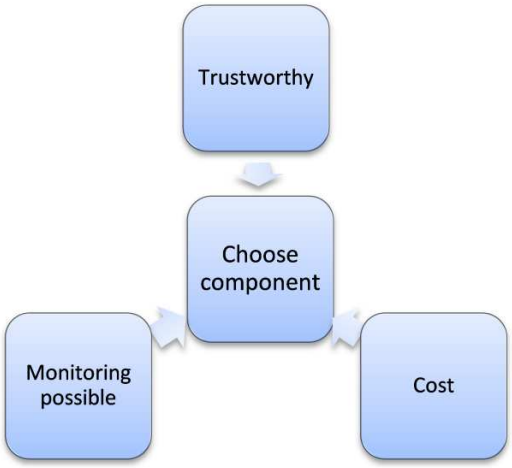


Figure 9: Further higher-level structure, including factors other than trust which might influence the choice of a component

3.1.7 Summary of relevance

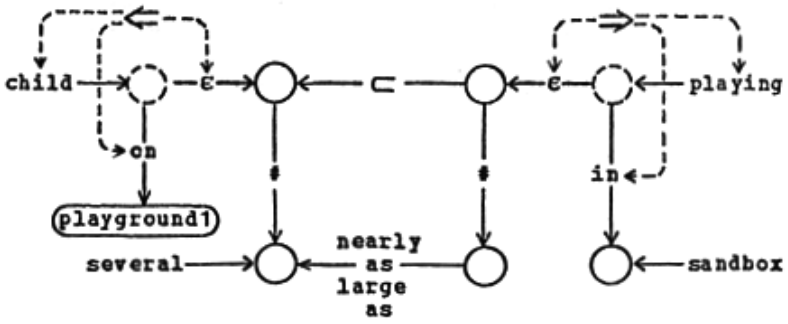
The main contribution of this section is to take seriously software designers and developers as themselves users and to explore socio-cognitive issues which might have a bearing on how they decide whether to trust a software service. The results have implications for how the platform should work. To take an important example, clearly testing – more generally enumerative induction, i.e., trying to verify $P(x)$ for all x by testing instances $P(i_1)$, $P(i_2)$, …, $P(i_n)$ for a finite n – is crucial for software developers. They trust by testing, so they should recognise a process in the Aniketos platform which feels to them like testing. Another issue is that a range of social processes, e.g., trusting someone who is perceived to be an epistemic authority, beyond the technical processes, are important when people choose software components – and hence are likely still to be important when they choose services. The ideal Aniketos platform would enable these processes, where possible helping developers and designers to avoid reasoning biases driven by processes they normally use in non-technical domains.

3.2 Patterns and best practices for developing and providing trustworthy composite service

This section defines and investigates patterns and guidelines for establishing user trust in composite services (including dynamic environments) that can be provided to the service developers to enhance service trustworthiness. The patterns and guidelines can refer to elements or user actions on the Aniketos platform, as well as social structures in the usage context. To facilitate the take-up of these patterns, each is followed by a "target-user". The intention is that these give helpful guidelines for various developers and providers in Aniketos.

3.2.1 Communicating formalisms

Help explain the meaning of formal properties with positive and negative examples	
Context	Many of the tools proposed for Aniketos require the use of formal properties, e.g., (i) for input to verification tools; (ii) to understand what properties have been verified.
Problem	Formalisms are often difficult to understand, even for professional developers. Although this difficulty cannot be removed altogether, it seems likely that aids can be provided, especially to help those with little experience with formal methods.

Non-solutions	*Expressing the property in natural language.* Consider the following example (slightly rewritten from an example in [62]): "If a rule exists affirming of every member of the class A the property B then […] whenever a specific object, x, that is a member of A is encountered it can be inferred that *x* has the property B…" This is supposed to express a simple formula: $$(\forall x \in A\ B(x))\ \&\ y \in A\ \rightarrow\ B(y)$$ One can imagine how only slightly more complex expressions would be rendered completely incomprehensible in natural language. Using natural language clearly removes the benefits of the logical expression and does not improve matters. *Using only diagrammatic notations.* Diagrammatic notations can, but do not necessarily help either, e.g., consider the following Schubert diagram:  This supposedly expresses that several children are in the playground and most of them are playing in the sandbox (example from [64]; see for further discussion).
Solution	Provide illustrative positive and negative example instances of a property, i.e., examples which are true and examples which are false. Importantly these should be presented together with the formal properties to emphasise that the examples are only examples and do not perfectly characterise the full space of possibilities.
Examples	$$\forall x, y.\, x < y \rightarrow A(x) \leq A(y)$$ Positive examples: A = <1,2,3,4,5>; A = <1,4,5,7,8> Negative examples: A = <5,4,3,2,1>; A = <1,2,3,4,1> $$\forall x, y, z.\, trusts(x, y) \& trusts(y, z) \rightarrow trusts(x, z)$$ Positive example: $trusts(wife, husband), trusts(husband, mistress), trusts(wife, mistress)$ Negative example: $trusts(wife, husband), trusts(husband, mistress), \neg trusts(wife, mistress)$
Target user	Developers of formal tool support in Aniketos

Explaining universal proofs with examples instances	
Context	Many of the tools produce a proof of properties
Problem	Proofs produced by automatic provers are also difficult to understand as they are not intended for human consumption. Even mathematicians tend to dislike completely formal proofs, e.g., [63] "Presenting proofs in [formal, automatically checkable] style has never taken off within the mathematical community. Instead, mathematicians write *rigorous* proofs, i.e. proofs in whose soundness the mathematical community has confidence, but which are not [formal]."
Non-solution	Use informal proof. There is no evidence that developers currently write non-formal proofs. Also such proofs could not be produced automatically, eliminating any

	benefits of tool support.
Solution	Provide example instances to demonstrate what has been proved.
Examples	Property: x+y = y+x Examples: 2+3 = 3+2; 5+6 = 6+5; …
Target user	Developers of formal tool support in Aniketos

3.2.2 Certificates

Explain scope of certificate	
Context	Many companies produce certificates that a piece of software is trustworthy.
Problem	Consumers of the certificate are often unaware exactly what has been verified.
Non-solution	Expose the processes used to test/verify the software. This is often not possible because, e.g., (a) doing so would imply losing a competitive advantage, (b) the technical details might not be understandable to the client.
Solution	Provide examples of the properties which have been demonstrated to hold (this could include the proof that an undesirable property holds) and also examples of those which have not been tested/verified.
Example	"Injection attacks were run for the most commonly detected SQL-based zero-day exploits. No testing was done of code related to the network protocol." Properties *P* and *Q* were tested however *R* was not tested. (Note: this does not mean that *P* and *Q* were true and *R* was false, rather that nothing is known about *R*.)
Target user	Security engineers who certify services

3.2.3 Trustworthy service offerings and trustworthy communication

Authenticity of the used service	
Context	When using a service, the consuming party (e.g., end user, service developer of a composite service) should be able to prove the authenticity of the consumed service.
Problem	Many (successful) attempts to attack the privacy or integrity of service consumers are based on a dishonest service that pretends to be the demanded (honest) service.
Non-solutions	Relying only on the domain name (DNS system) as means for providing an authentic URI for a service. Use of self-signed certificates without a secure and authentic communication of the properties of the key used for the creation of the certificate.
Solution	Provide a SSL enabled URI for the service that uses a cryptographic certificate which is signed by a well-known and trusted certification authority (e.g. Verisign). To increase the usability for end-users, the public key of the root-certificate of this certification authority should be in the set of default certificates used by Web browsers or the underlying platform used for the client development. A strong encryption scheme shall be used.
Limitation	This pattern only covers the authenticity of the service, i.e., in general the user has still to trust the service. Note in particular that an authentic service can still be dishonest with respect to other security or trust properties (e.g., forwarding confidential data to a third party).
Example	An Example for a service that implements this patterns are services offering software downloads (e.g., Linux distributions or software patches from commercial vendors). While the software supplier wants to guarantee the origin (authenticity) of the software, he does not necessarily need to authenticate the users.
Target user	Service developers and service providers

Authenticity of the user of a service	
Context	When using a service, the offering party (e.g., service developer) should be able to prove the authenticity of the service consumer.
Problem	Many Web service offerings (e.g., electronic banking) require the authentication of the consuming party.
Non-solution	Using HTTP Basic Authentication or Login/Password-based solutions over an unsecured communication channel.
Solution	Solutions that can be implemented for most standards environments without requiring additional hardware are: 1. Using an SSL enabled URI for the service together with SSL client certificates 2. Using a secure channel (e.g., encrypted using SSL) with HTTP Basic Authentication 3. Using a secure channel (e.g., encrypted using SSL) with application specific login/password-based authentication. In all three solutions, a strong encryption scheme shall be used. In case stronger means of authentication (e.g. two-factor authentication) are required, additional steps need to be taken in to account. Examples for techniques providing a stronger means of authentication are: • Generating one time transaction numbers (TAN) on the server that are sent via an independent communication channel (e.g., as SMS to the user's mobile phone) to the user. The user then has to insert a fresh TAN for each transaction. • Using hardware-generated one time passwords as, e.g., provided by RSA's SecurID solutions (http://www.rsa.com/SecurID).
Limitation	The solutions that are not based on additional hardware tokens, respectively, one time passwords or TANS, (i.e., the first three solutions) are prone to replay attacks (e.g., the credentials can be stolen using Trojans or key-loggers).
Example	An example service type that requires the authentication of end users are Web shops.
Target user	Service developers and service providers

Secure and authentic communication between a service and consumer of a service	
Context	Many service offerings require both the secure communication and the mutual authentication between the service and its consumer.
Problem	Many service offerings (e.g., electronic banking) require the exchange of confidential data between the service and its consumer. Therefore, the communication should ensure the secrecy and authenticity of the exchanged data/information.
Non-solution	Using SSL with a weak encryption scheme.
Solution	Combine the solutions of pattern 3.2.1and 0, i.e. using an SSL enabled URI service that uses a cryptographic certificate which is signed by a well-known and trusted certification authority (e.g. Verisign) either with • SSL client certificates, • HTTP Basic Authentication, or • application specific login/password-based authentication. A strong encryption scheme shall be used. For applications requiring that likely targets for replay attacks (e.g., Web applications providing financial transactions services such as e-banking or ones that provide

	access to internal networks) hardware-based one-time credentials shall be used.
Limitation	See pattern 3.2.1 and 3.2.3.2; in particular, solutions that do not use one-time credentials are prone to replay attacks.
Example	E-banking, i.e., Web services offered by banks that allow for executing transactions on bank accounts require, usually, a two-factor authenticated and secure channel.
Target user	Service developers and service providers

Warn about reputation-based biases	
Context	People are influenced by the reputation of organisations which often leads them to trust the quality of a service, in a particular situation, based on more general cues about trustworthiness.
Problem	Occasionally trust can become so routine that obvious cases of untrustworthiness are undetected. For example, a study showed that people trust a major search engine's ability to rank the relevance of results, even when presented with (experimentally manipulated) specific cases where it provides less optimum solutions [85]. Although participants showed behaviour consistent with being aware of the problem, e.g., in terms of eye fixation patterns, they still selected the highest-ranked results.
Non-solution	It might be argued that organisations should take advantage of such biases to increase the use of their services; however a problem with this approach is that if specific deficiencies are exposed, then reputation could be later affected.
Solution	Emphasise the benefits and drawbacks of any services. Typically a service will have to have some merit to be used at all.
Examples	Advise users on the possible limitations of the rank algorithm when used for automation so that they have a mental model of when not to trust. Provide estimates of the uncertainty that a particular hit is relevant.
Target user	Developers of the trustworthiness computation engine

Information about individual trust metrics	
Context	For service developers it is important to have detailed information on the individual service characteristics or factors that have been measured and combined for elaborating the trustworthiness level of the service, rather than having only a single computed value or result.
Problem	As developer trust in a service or a service developer has a cognitive aspect to be considered and certain characteristics may have a higher impact on developers' trust than others, depending on the context. Service developers may prefer to have all the information related to how the trustworthiness is calculated, and have the result decomposed with respect to the individual measures taken on several factors.
Non-solution	Provide the algorithm used to calculate the trustworthiness value, so developers can understand the impact of certain characteristics on the final computed value.
Solution	Provide the algorithm and the break down result by each relevant characteristic so that developers can understand better each of the measures taken to compute the result.
Limitation	Some factors are measured at runtime, thus not all the individual measures might have sense at design-time. Additionally for components where no information yet exists (first use with the Aniketos platform) measures are also null in the beginning.
Examples	Advice developers on the calculations performed to reach a certain service trustworthiness value; provide the algorithm for calculating reputation but also the different measures taken on the involved factors.
Target user	Service developers

Ratings credibility	
Context	User ratings and reputation can help establish and maintain trust between service consumers and their provider.
Problem	Anyone can rate a service whether they have used it or not. This can reduce the credibility of the ratings due to possibility of multiple ratings for the same transaction and easy access for users with malicious intentions such as boosting or slandering the reputation of a particular service.
Non-solution	Reduce the effect of received bad and malicious ratings by only filtering or manipulation.
Solution	Permit only per business transaction rating and use authentication mechanisms to ensure that only registered users are allowed to rate a service. Other mechanisms such as filtering can further improve the credibility of ratings.
Example	eBay allows a rating per user transaction. All users must be authenticated in order to carry out a transaction.
Target user	Developers of the trustworthiness prediction and monitoring systems including Aniketos.

3.2.4 Service implementation

Checking of user input	
Context	Many service implementations are not checking all user supplied input. Unexpected user input may result in security vulnerabilities.
Problem	Many Web applications that accept user input are using this user input for generating new Web sites or store this input internally, e.g., in a relational database.
Non-solution	Directly handling user input, e.g., storing user input directly in a database without checking it properly beforehand.
Solution	All input received from (potential) untrusted source, e.g., users, needs to be checked carefully and all dangerous parts need either to be removed or encoded safely. For example, input that is stored in an SQL database needs to replace every occurrence of a single quote (') in user input by two single quotes ('') to form a valid SQL string literal (other encodings might be required additionally). Moreover, for preventing SQL injections attacks there are also other countermeasures such as using prepared SQL statements. Similarly, output needs to be encoded as well to prevent, e.g., Cross-site-Scripting attacks. Overall, for the source code of each shipped or deployed software component it should be validated that all user input and all output is correctly encoded. Such validation can be done completely manually or supported by tools such as static code analysers.
Limitation	The encoding of the input and output must be done for each use separately, i.e., there is no general encoding routine that prevents all attacks that can be caused by improper encoding of input or output data.
Example	Examples for attacks that are caused by improper input or output encodings are SQL injection attacks, Cross-site-scripting (XSS) attacks, or cross-site-request-forgery.
Target user	Service developers and certification authorities (based on code audits).

Debug code in deployed service	
Context	Many service implementations contain debug code, i.e., code that is only useful

	during the development of the service, that may reveal unwanted functionalities or security vulnerabilities.
Problem	During the development of software, developers often introduce debugging code that helps to debug problems (e.g., additional output) in the software or simplify the development (e.g., circumventing authorization checks). The former, for example, may reveal additional information such as version numbers of libraries that help attackers. The latter may disable authentication or authorization checks at all, i.e., introducing so called back-doors.
Non-solution	Shipping or deploying the service implementations that include debugging code.
Solution	Remove (or at least put them into comments) all debug code before shipping or deploying a service. Overall, for the source code of each shipped or deployed software component it should be validated that all debug code is removed. Such validation can be done completely manually or supported by tools such as static code analyzers
Example	Backdoors are one example for vulnerabilities that can be caused by shipped debug code.
Target user	Service developers and certification authorities (based on code audits)

3.2.5 Contracts

Explain scope of contract (Derived from [65])	
Context	As Web service trustworthiness is a measure of whether the service will perform as intended, the service contract that explains exactly the service intended behaviour (functionality and non-functional aspects of the service) should be shown by the Aniketos platform.
Problem	The service contract has an abstract description of what the service does and more concrete description of how and where the service can be accessed. The abstract description describes the actual physical interface (API) that service consumers connect to. The technical details of the functionalities and other implementation and communication aspects of the service are needed for service consumers to locate and use the service at runtime. These technical details are often complex and not really explanatory for the service consumer, and sometimes not accessible at all.
Non-solution	Show in the Aniketos platform the complete service contract.
Solution	The Aniketos platform should specify contract metadata in a format that service providers and consumers can understand.
Example	Contract should have a summary in natural language that explains what relationship the trustworthiness level is applicable to, that includes: - Service provider: Service provider ID and data. - What: API - How: Binding-related definitions - Where: Service Ports addresses (a URL, an email address, or some other type of transport-specific address) - Policies: Integrated in what, how and where parts.
Target user	This pattern should be used by the Aniketos platform developers in order to implement the solution in the platform.

Link to actual contract

Context	For some services, it might be important to understand the details of the service in order to really trust the service.
Problem	The summary of the service contract is not enough for the developer to trust the service for some security features (e.g. non-disclosure of data to 3rd parties).
Non-solution	Showing the complete service contract file in the Aniketos platform might result in a cluttered interface that does not help the consumer in understanding the actual content of the contract.
Solution	The Aniketos platform should show a summary of the service description in a natural and easy to understand language and provide, when possible, a URL link to the actual contract file, so the developer can consult it if further clarifications are needed.
Example	http://mydomain.com/myservicecontract...
Target user	This pattern should be used by the Aniketos platform developers in order to implement the solution in the platform.

Link to actual socio-technical security model of the service	
Context	For some services, it may be important to have the description of high level requirements of the service, to understand the trust relationship the contract applies to.
Problem	The summary of the service contract is not enough for the developer to trust the service for some security features (e.g. non-disclosure of data to 3rd parties).
Non-solution	Show in the Aniketos platform the complete service contract.
Solution	When possible, the Aniketos platform should provide a URL link to the service model created with the socio-technical security modelling language
Example	http://mydomain.com/myservicegoals...
Target user	This pattern should be used by the Aniketos platform developers in order to implement the solution in the platform.

Provider and services contract traceability	
Context	For composite services it might be important to have the traceability of the service contract with the individual components' contracts.
Problem	A composite service contract might not be enough for the service developer to understand the implications on a type of behaviour because it refers to other contracts (the ones of its components).
Non-solution	Show in the Aniketos platform the complete service contract.
Solution	When possible, the Aniketos platform should provide a URL link to the individual service contracts as well as the composite service contract.
Example	http://mydomain.com/myservicecontract... http://mydomain.com/myserviceComponent1contract... http://mydomain.com/myserviceComponent2contract... [etc.]
Target user	This pattern should be used by the Aniketos platform developers in order to implement the solution in the platform.

4 Aniketos trust requirements

This chapter analyses functional, non-functional and architectural requirements based on WP1 tasks T1.3 ("Aniketos platform requirements and scenarios") and T1.4 ("Overall architecture of Aniketos platform") that are relevant to both WP2 in establishing and maintaining trust in composed services and to WP1 in defining the socio-technical security modelling language and tool (T1.2). Traceability to the high-level requirements listed in D1.2 ("First Aniketos architecture and requirements specification") was performed but not documented herein.

The requirements presented here represent a "complete" set of requirements as expressed by potential "end-users" of the design-time and runtime tools. For this work, the role of "end-user" was mainly held by the industrial partners of Aniketos. The requirements are "complete" in the sense that they were not restricted by project-related constraints such as available budget, resources and time to develop the specified tool; they were written with a "let's dream anything is possible" spirit. The objective is not therefore to obtain a full coverage of these requirements within the timeframe of Aniketos, but to have a global understanding of the ultimate goal that is pursued. Evaluation, through WP6 and WP7, will allow us to identify which requirements were covered. Non covered requirements will be assessed in the exploitation plan, as an input to construct the technical exploitation roadmap.

As indicated in the chapter's title, trust is central in the following requirements. However, trust is handled differently according to the engineering phases (and therefore the tools) being specified. For the requirements engineering phase (and its related socio-technical security modelling language and tool as defined in task T1.2), trust is mainly handled under its cognitive acceptation; the specification work performed herein has made extensive use of the state of the art presented in Section 2.1.1. For the software design and development engineering phases (and their related trust management components), trust is mainly handled under its trustworthiness acceptation with metrics that can be computed; the related specification rests upon the work presented in Section 3.

4.1 Definitions

4.1.1 Operational organisation

The overall Aniketos stakeholders have been classified in D1.2. The Aniketos stakeholders involved with the trust management system/software are:

- Aniketos authority: this stakeholder maintains the software and services part of the Aniketos platform, of which trust is a foundation;
- Aniketos platform contributor, i.e. a user able to enrich the Aniketos platform with meaningful content: this stakeholder must understand the trust mechanisms that underlie the Aniketos platform so as to contribute usefully;
- Service end-user: this stakeholder uses the trust management system/software to select a service, and provides feedback that nourishes the reputation mechanisms;
- Service provider: this stakeholder must understand the trust mechanisms that underlie the Aniketos platform to ensure that the provided service is selected by service end-users;
- Service designer (sub-category of service developer): this stakeholder uses the trust management system/software to elicit the security needs and security requirements;
- Service implementer (sub-category of service developer): this stakeholder needs to understand the trust mechanisms so as to include those mechanisms in his/her own services;
- Service composer (sub-category of service developer): this stakeholder uses the trust management system/software to discover trustworthy candidates for service composition;
- Certifying authority (sub-category of third-party): this stakeholder needs to understand the trust mechanisms so as to be able to certify them.

4.1.2 Functional capability

4.1.2.1 Concept

In Aniketos the trustworthiness prediction module predicts the current trustworthiness of a service based on the trust relationship and trust model(s) in use. It will also be able to estimate the overall trustworthiness of a composite service based on the suggested composite assembly. The module can provide those predictions to other modules during both the service design time and runtime states. On the other hand, the trustworthiness monitoring module enables runtime monitoring of trustworthiness based on Aniketos mechanisms and metrics adequate for the runtime state. If changes in trustworthiness defy the required levels in a contract, then the affected parties are notified using the notification module. This module invokes the trustworthiness prediction module to retrieve trustworthiness predictions that may be affected by runtime changes and threats. Further details on module interactions can be found in D1.2.

4.1.2.2 Basic functions

The basic functions of the trust management system are:

- Trust modelling: this capacity is a design-time only function; it allows the elicitation and capture of trust relationships and derivation of security requirements;
- Trustworthiness prediction: this function computes the trustworthiness of a service based on the trust relationship and trust model(s) in use. It is also able to estimate the overall trustworthiness of a composite service based on the suggested composite assembly; it can be used both at runtime and at design-time;
- Trustworthiness monitoring: this function enables runtime monitoring of trustworthiness based on the mechanisms and metrics defined according to the trust model.

The three functions of the trust management system/software have a structuring effect on the remainder of this chapter, as the requirements are grouped and coded according to the functions.

4.1.3 States and modes

Organising requirements according to states and modes is common practice in industry, but probably rarely seen in research projects. States and modes allow for the specification of requirements that depend on the system/software's state and/or mode, with a particular focus on operational states and degraded modes. As integrity is a priced value in the Aniketos project, describing system/software states and modes seemed highly relevant, even if it was in a simplified way.

The states and modes have a structuring effect on the remainder of this chapter, as the requirements are grouped and coded according to these states (cf. Sections 4.2.1 and 4.2.2) and modes. In addition, interface and quality factors are usually specified; due to the research nature of this project, these latter requirements (cf. Sections 4.2.3 and 4.2.4) have been kept to a minimum.

4.1.3.1 States

The trust management system/software has two states:

- design-time operational state;
- runtime operational state.

4.1.3.2 Modes

The trust management system/software has two modes:

- normal operational mode;
- failed mode.

Considering that Aniketos is a research project, reduced operational modes (in which only a subpart of the functions are available) are not considered. If Aniketos were to go operational, reduced operational modes should be carefully studied.

4.1.3.3 Mode transitions

4.1.3.3.1 From operational to failed mode

The detection of the failure of any one of the functions will cause the system/software to enter in the failed mode.

4.1.3.3.2 From failed to operational mode

In case all of the functions become available again after a failure, the system/software is allowed to go to normal operational mode.

4.1.4 Editorial practices

The following editorial practice has been followed in the writing of the requirements:

- for the normative requirements the operative verb "shall" is used;
- for the recommended requirements the operative verb "should" is used;
- for the optional requirements the operative verb "may" is used.

System requirements are composed of three parts:

- a unique identifier,
- a title,
- the requirement body, which describes the function, the capacity(ies) or the parameter(s).

The requirement unique identifier is composed as follows:

- the ANIKETOS keyword;
- a function-code which can take the following values: TM for "Trust Modelling", TD for "Trustworthiness Determination" and TN for "Trustworthiness moNitoring";
- a unique sequence number within the function-code, with a range from 001 to 999.

It is worth noting that the sequence numbers run through the design-time and runtime sections, contributing to giving a functional coherency throughout the system/software states.

4.2 Characteristics[5]

4.2.1 Design-time operational state

Normal Operation mode – *Trust modelling – Model management*		
ID	**Name**	**Description**
ANIKETOS-TM-001	*Model initialisation: study*	The design-time tools ***shall*** organise all the design-time work in a coherent set called "study".
ANIKETOS-TM-002	*Model initialisation: study identification*	A study ***shall*** be identified by: (i) a name; (ii) a creation date; (iii) a responsible author.

[5] Characteristics of the system / software defined through requirements.

ANIKETOS-TM-003	*Model initialisation: study general description*	The design-time tools ***should*** allow for a general description of the study in textual format.
ANIKETOS-TM-004	*Model maintenance: model states*	The design-time tools ***shall*** allow for the capture of an overall model state, including at least a "working" state and a "validated" state.
ANIKETOS-TM-005	*Model maintenance: model validated state*	The design-time tools ***shall*** prohibit any modification to a model in the "validated" state unless the state is compromised.
ANIKETOS-TM-006	*Model maintenance: state transition from working to validated*	The design-time tools ***shall*** require user confirmation and logging (identification and timestamp) to transition the model from the "working" state to the "validated" state.
ANIKETOS-TM-007	*Model maintenance: state transition from validated to working*	The design-time tools ***shall*** require user confirmation and logging (identification and timestamp) to transition the model from the "validated" state back to the "working" state.
ANIKETOS-TM-008	*Model maintenance: model state and consistency*	The design-time tools ***shall*** prohibit the model to be set to the "validated" state if it is inconsistent (cf. consistency requirements).

Normal Operation mode – *Trust modelling – Capture of entities*		
ID	**Name**	**Description**
ANIKETOS-TM-009	*Capture: actors*	The design-time tools ***shall*** allow for the capture of actors, where an actor[6] is an implementation independent unit of responsibility that performs actions to achieve an effect that contributes to a

[6] According to NAF [87], "*In order to be able to allocate responsibilities in a coherent and non-conflicting manner to actual persons, the responsibilities need to be captured first, irrespective of the actual persons or executing parties. The complete set of these responsibilities needs to cover all of NATO's internal and external obligations. In order to fulfil his obligations, an actor has a certain information need. By linking these information requirements to the responsibility instead of to an individual, flexibility and applicability of information requirements become independent of an incidental organisational structure. The structure (e.g. for a new mission) can change, but as soon as the responsibilities are allocated to individuals at certain locations, the information requirements are known. Due to the linkage of these information requirements to responsibilities, the conditions (e.g. timeliness of information, availability, etc.) for fulfilling the requirements also become clear and stable. […]Anything actors do needs to contribute to the goals of the organisation. This provides a means to validate whether all responsibilities and tasks of the actors indeed contribute to at least one goal of the organisation, and whether all goals are taken care of by the responsibilities and corresponding tasks of actors. The operational objectives are enduring, they do not change easily over time, and e.g. the ultimate goal of NATO has not changed since the alliance was established*." Even though the above text is extracted from the NATO architecture framework, it provides a good stable foundation for any architecture, including service oriented architectures such as the ones developed for and by Aniketos.

Normal Operation mode – *Trust modelling – Capture of entities*		
ID	**Name**	**Description**
		desired end state. *Note: the set of responsibilities will be defined through goals to achieve.*
ANIKETOS-TM-010	*Capture: agents, instances of actors*	The design-time tools ***may*** allow for the capture of particular instances of actors, hereafter called agents.
ANIKETOS-TM-011	*Capture: actor description*	The design-time tools ***shall*** allow for the capture of a short textual description of the actors.
ANIKETOS-TM-012	*Capture: social structure purpose*	When an actor represents a social structure, the design-time tools ***should*** allow for the capture of a short textual description of its purpose.
ANIKETOS-TM-013	*Capture: non-participant stakeholders*	The design-time tools ***shall*** allow for the capture of non-participant stakeholders.
ANIKETOS-TM-014	*Capture: non-participant stakeholder description*	The design-time tools ***shall*** allow for the capture of a short textual description of the non-participant stakeholders.
ANIKETOS-TM-015	*Capture: context facts*	The design-time tools ***shall*** allow for the capture of facts (e.g. legal and regulatory references) that may influence the overall risk management process.
ANIKETOS-TM-016	*Capture: context events*	The design-time tools ***shall*** allow for the capture of events (typically, elements of the climate / situation) that may influence the overall risk management process.
ANIKETOS-TM-017	*Capture: assets*	The design-time tools ***shall*** allow for the capture of the assets of the actors[7], in terms of goals to reach, and supporting resources.
ANIKETOS-TM-018	*Capture: asset depository*	Assets ***shall*** be associated to their depositories (i.e. the actors that are or feel accountable for the assets). *Note: thus, an asset may be associated to multiple depositories.*
ANIKETOS-TM-019	*Capture: resource owners*	The design-time tools ***shall*** allow for the capture of resource owners.
ANIKETOS-TM-020	*Capture: resource multiple-ownership*	The design-time tools ***should*** allow for the capture of multiple owners for a given resource.

[7] Not to agents or non-participant stakeholders.

Normal Operation mode – *Trust modelling – Capture of entities*		
ID	**Name**	**Description**
		Note: this multiple-ownership may be the result of the grouping of multiple resources into an aggregated resource; in this case, resource decomposition should allow for identifying a unique owner for each sub-resource.
ANIKETOS-TM-021	*Capture: resource types*	The design-time tools ***shall*** allow for the differentiation between intangible and tangible resources.
ANIKETOS-TM-022	*Capture: resource marking, security-irrelevance*	The design-time tools ***shall*** allow for the marking of a resource as being irrelevant with respect to security needs. *Note: This requirement concerns only sub-resources. Indeed, if a top-level resource is security-irrelevant, it should be suppressed; thus there is no reason to mark it.*
ANIKETOS-TM-023	*Capture: resource decomposition*	The design-time tools ***shall*** allow for an AND/OR decomposition of a resource in sub-resources, under the condition that it has not been marked as security-irrelevant.
ANIKETOS-TM-024	*Capture: tangible resource decomposition*	The AND/OR decomposition of a tangible resource ***shall*** always be a tree-decomposition.
ANIKETOS-TM-025	*Capture: intangible resource decomposition*	The AND/OR decomposition of an intangible resource ***shall*** be of any type of non-cyclic graph.
ANIKETOS-TM-026	*Capture: multi-dimensional resource decomposition*	The AND/OR decomposition of an intangible resource ***should*** support multiple orthogonal decompositions, e.g. temporal (states), structural (contents).
ANIKETOS-TM-027	*Capture: asset types*	The design-time tools ***shall*** allow for the differentiation between primary assets and supporting assets[8].
ANIKETOS-TM-028	*Capture: strategy to meet a goal*	The design-time tools ***should*** allow for the capture, in textual format, of the overall strategy to meet a goal.
ANIKETOS-TM-029	*Capture: goal marking, self-capability*	The design-time tools ***shall*** allow for the marking of a goal of an actor as being fully fulfilled by that sole actor.
ANIKETOS-TM-030	*Capture: goal marking, security-irrelevance*	The design-time tools ***shall*** allow for the marking of a goal of an actor as being irrelevant with respect to security needs.

[8] Primary vs. supporting applies to assets, whilst tangible versus intangible applies to resources only, however both concepts have similarities (to be further specified).

Normal Operation mode – *Trust modelling – Capture of entities*		
ID	**Name**	**Description**
		Note: this may happen when a high-level goal is decomposed in sub-goals, some of which are security critical, whilst others are security agnostic.
ANIKETOS-TM-031	*Capture: goal decomposition*	The design-time tools ***shall*** allow for an AND/OR decomposition of a goal in sub-goals, under the condition that it has not been marked (as a self-capability or as security-irrelevant).
ANIKETOS-TM-032	*Capture: collaboration to achieve a goal*	The design-time tools ***shall*** allow for the capture of the collaborative nature of a goal, highlighting all the actors potentially involved in the fulfilment of the goal; such calls are hereafter called "collaborative goals" irrespective of the fact that the collaboration is enacted. *Note 1: the collaborative nature of a goal is a high-level view of that goal in the sense that this goal can be decomposed in sub-goals that can each be individually realised by different actors.* *Note 2: unlike for delegation, authorisation to collaborate for the fulfilment of the goal is not subsumed by a collaboration potential.*
ANIKETOS-TM-033	*Capture: link between primary assets*	The design-time tools ***shall*** allow for the capture of a "need" (or "means-end") link between a goal and an intangible resource. *Note: the "need" link does not convey any flow information; whether the resource linked to a goal is accessed, consumed, produced, modified or destroyed is irrelevant; the link semantics only conveys the fact that this goal intrinsically cannot be achieved without that resource being available.*
ANIKETOS-TM-034	*Capture: delegation*	The design-time tools ***shall*** allow for the capture of a goal delegation from one actor to another actor.

Normal Operation mode – *Trust modelling – Capture of security needs*		
ID	**Name**	**Description**
ANIKETOS-TM-035	*Capture: implicit actor authenticity*	A goal assignment to its depository (or depositories) ***shall*** subsume actor authenticity. *Note: actor authenticity is the property*

Normal Operation mode – *Trust modelling – Capture of security needs*		
ID	**Name**	**Description**
		of the actor of being genuine and being able to be verified and/or trusted.
ANIKETOS-TM-036	*Capture: implicit resource access authorization*	A "need" link between a goal and an intangible resource ***shall*** subsume access privileges granted to the actor responsible for the goal achievement over the intangible resource.
ANIKETOS-TM-037	*Capture: delegation implicit authorization*	Goal delegation ***shall*** subsume delegation authorization. *Note: goal delegation authorization relates to the legitimacy of the interaction.*
ANIKETOS-TM-038	*Capture: implicit goal integrity*	Goal specification ***shall*** subsume goal integrity for all goals that are not marked as security irrelevant. *Note: goal integrity is the quality that a goal has when it is attained or sustained in an unimpaired manner, free from unauthorized manipulation.*
ANIKETOS-TM-039	*Capture: availability security need*	The design-time tools ***shall*** allow for the capture of a primary asset (i.e. goal or resource) availability security need.
ANIKETOS-TM-040	*Capture: availability expression*	An availability security need ***may*** be expressed through a ratio or as plain text.
ANIKETOS-TM-041	*Capture: least-privilege security need*	The design-time tools ***shall*** allow for the capture of a goal least-privilege security need. *Note: the semantics conveyed is that, while achieving a goal, the involved actors shall have access only to the necessary resources for that goal, and nothing else.*
ANIKETOS-TM-042	*Capture: need-to-know security need*	The design-time tools ***shall*** allow for the capture of an intangible resource need-to-know security need. *Note: the semantics conveyed is that a resource with a need-to-know security property shall only be accessible to the actors involved in the achievement of those goals to which it is linked by a "need" link.*
ANIKETOS-TM-043	*Capture: intangible resource privacy security need*	The design-time tools ***shall*** allow for the capture of an intangible resource privacy security need. *Note: the intangible resource privacy security need expresses that the intangible resource must remain known only to its owner(s).*

Normal Operation mode – *Trust modelling – Capture of security needs*		
ID	**Name**	**Description**
ANIKETOS-TM-044	*Capture: goal privacy security need*	The design-time tools ***should*** allow for the capture of a goal privacy security need. *Note: the goal privacy security need expresses that third parties should not be aware that that goal is being pursued; this may imply that the involved parties (e.g. through delegation) remain unknown to third-parties.*
ANIKETOS-TM-045	*Capture: non-repudiation security need*	The design-time tools ***shall*** allow for the capture of a non-repudiation security need between an actor and a goal. *Note: non-repudiation will assure that the actor is provided with proof of goal completion and that the collaborating parties, if any, are provided with proof of the actor's identity, so neither can later deny having contributed to the achievement of the goal.*
ANIKETOS-TM-046	*Capture: accountability security need*	The design-time tools ***shall*** allow for the capture of an accountability security need between an actor and a goal. *Note: the accountability security need expresses that the depository (i.e. actor) is accountable for that goal, regardless of any delegation he will make.*
ANIKETOS-TM-047	*Capture: non-repudiation versus accountability security needs*	The non-repudiation and accountability security needs ***shall*** be mutually exclusive. *Note: accountability is considered to include authenticity and non-repudiation.*
ANIKETOS-TM-048	*Capture: anonymity security need*	The design-time tools ***shall*** allow for the capture of an anonymity security need between an actor and a delegated goal.
ANIKETOS-TM-049	*Capture: non-repudiation versus anonymity security needs*	The non-repudiation and anonymity security needs ***shall*** be mutually exclusive.
ANIKETOS-TM-050	*Capture: accountability versus anonymity security needs*	The accountability and anonymity security needs ***shall*** be mutually exclusive.
ANIKETOS-TM-051	*Capture: data integrity security need*	The design-time tools ***shall*** allow for the capture of an intangible resource integrity security need.
ANIKETOS-TM-052	*Capture: data location security need*	The design-time tools ***shall*** allow for the capture of a required data location for each intangible resource. *Note: this requirement is specifically*

Normal Operation mode – *Trust modelling – Capture of security needs*		
ID	**Name**	**Description**
		designed to cope with cloud-computing platforms with which it is not always clear where data will be stored and/or processed.
ANIKETOS-TM-053	*Capture: separation of duty security need*	The design-time tools ***shall*** allow for the capture of the separation of duty security need.
ANIKETOS-TM-054	*Capture: consent security need*	The design-time tools ***shall*** allow for the capture of the consent security need.
ANIKETOS-TM-055	*Capture: correction right security need*	The design-time tools ***shall*** allow for the capture of the correction right security need.
ANIKETOS-TM-056	*Capture: written-purpose security need*	The design-time tools ***shall*** allow for the capture of the written-purpose security need.
ANIKETOS-TM-057	*Capture: staff agreement security need*	The design-time tools ***shall*** allow for the capture of the staff agreement security need.
ANIKETOS-TM-058	*Capture: auditability security need*	The design-time tools ***shall*** allow for the capture of the goal auditability security need.
ANIKETOS-TM-059	*Capture: security need motivation*	The design-time tools ***shall*** allow for the capture in textual format of the motivation (justification) behind each security need.

Normal Operation mode – *Trust modelling – Capture of trust*		
ID	**Name**	**Description**
ANIKETOS-TM-060	*Capture: trust relations*	The design-time tools ***shall*** allow for the capture of trust relations between actors, expressed at the level of a goal, meaning that the trustor trusts the trustee for the fulfilment of that specific trustum (i.e. goal). *Note: the trust is subsumed to extend to the resources linked to the goal.*
ANIKETOS-TM-061	*Capture: distrust relations*	The design-time tools ***should*** allow for the capture of distrust between actors, expressed at the level of a goal, meaning that the trustor distrusts the trustee for the fulfilment of that specific trustum (i.e. goal). *Note: if this requirement holds, then any requirement hereafter valid for trust relations also holds for distrust relations.*
ANIKETOS-TM-	*Capture: absence of trust*	The design-time tools ***may*** allow for the capture of the absence of trust relations

Normal Operation mode – *Trust modelling – Capture of trust*		
ID	**Name**	**Description**
062	*relations*	between actors, expressed at the level of a goal, meaning that the trustor does not explicitly trust the trustee for the fulfilment of that specific trustum (i.e. goal). *Note: if this requirement holds, then any requirement hereafter valid for trust relations also holds for the absence of trust relations.*
ANIKETOS-TM-063	*Capture: absence of distrust relations*	The design-time tools ***may*** allow for the capture of the absence of distrust relation between actors, expressed at the level of a goal, meaning that the trustor does not explicitly distrust the trustee for the fulfilment of that specific trustum (i.e. goal). *Note: if this requirement holds, then any requirement hereafter valid for trust relations also holds for the absence of distrust relations.*
ANIKETOS-TM-064	*Capture: trust or distrust relation*	The design-time tools ***shall*** refuse the establishment of multiple types of trust relations (i.e. trust, distrust, absence of trust, absence of distrust) between actors, expressed at the level of the same goal, except for the couple {absence of trust, absence of distrust}. *Note: the occurrence of multiple antagonist relations between actors is possible if different goals are concerned.*
ANIKETOS-TM-065	*Capture: cognitive trust*	The design-time tools ***shall*** allow for the capture of trust relationships only between at least one cognitive actor and some other entity or entities (whether actor, service or resource, cognitive or non-cognitive).
ANIKETOS-TM-066	*Capture: trustworthiness*	The design-time tools ***shall*** allow for the capture of trustworthiness as an absolute property of an entity, whether actor (cognitive or non-cognitive), goal or resource. *Note: any entity having a trustworthiness property will be called hereafter a "trustworthy entity", irrespective of the trustworthiness value(s).*
ANIKETOS-TM-	*Capture: reputation*	The design-time tools ***shall*** allow for the capture of reputation as one viewpoint

Normal Operation mode – *Trust modelling – Capture of trust*		
ID	**Name**	**Description**
067		on trustworthiness.
ANIKETOS-TM-068	*Capture: certification*	The design-time tools ***shall*** allow for the capture of certification as one viewpoint on trustworthiness.
ANIKETOS-TM-069	*Capture: temporal trust*	The design-time tools ***should*** allow for the capture of a trust relationship restricted in time and duration.
ANIKETOS-TM-070	*Capture: contextual trust*	The design-time tools ***should*** allow for the capture of a trust relationship restricted to a specific context, in particular environmental context.

Normal Operation mode – *Trust modelling – Basic consistency checking*		
ID	**Name**	**Description**
ANIKETOS-TM-071	*Consistency checking compulsory triggering*	The design-time tools ***shall*** search and raise inconsistencies before changing the model state to "validated".
ANIKETOS-TM-072	*Consistency checking optional triggering*	The design-time tools ***should*** search and raise inconsistencies upon designer request.
ANIKETOS-TM-073	*Inconsistencies: irrelevant actor*	The design-time tools ***shall*** raise an inconsistency error if an actor has not been assigned at least one asset (goal or resource).
ANIKETOS-TM-074	*Inconsistencies: leaf goal decomposition error*	The design-time tools ***shall*** raise an inconsistency error if a leaf goal of a goal-decomposition has not been marked (as a self-capability or as security-irrelevant).
ANIKETOS-TM-075	*Inconsistencies: collaborative goal decomposition warning*	The design-time tools ***shall*** raise an inconsistency warning if a "collaborative goal" has not been decomposed. *Note: being collaborative gives a potential for delegation, but does not impose it.*
ANIKETOS-TM-076	*Inconsistencies: collaborative goal decomposition error*	The design-time tools ***shall*** raise an inconsistency error if a "collaborative goal" has been decomposed but none of its sub-goals have been delegated.
ANIKETOS-TM-077	*Inconsistencies: least-privilege warning*	The design-time tools ***shall*** raise an inconsistency warning if an actor has more than one goal, of which at least one has a least-privilege security need. *Note: achieving many goals whilst ensuring the least-privilege security need can be performed with appropriate data segregation, but promises to be difficult.*

Normal Operation mode – *Trust modelling –Consistency checking with the mainstream system engineering*		
ID	**Name**	**Description**
ANIKETOS-TM-078	*Mapping: concepts*	Depending on the selected mainstream system engineering framework, the actor and asset (goal and resource) concepts and their relations ***shall*** be mapped to their corresponding concepts and relations[9].
ANIKETOS-TM-079	*Inconsistencies: confidentiality issue*	Depending on the selected mainstream system engineering framework, the design-time tools ***should*** raise an inconsistency error if the information flow, as described in the mainstream system engineering model, contradicts one of the confidentiality security needs (i.e. least-privilege, need-to-know and privacy).

Normal Operation mode – *Trustworthiness determination – Computational analysis support*		
ID	**Name**	**Description**
ANIKETOS-TD-001	*Trustworthiness: computation*	The design-time tools ***shall*** allow for the computation of trustworthiness as a function of different trustworthiness viewpoints (e.g. reputation, certification).
ANIKETOS-TD-002	*Trustworthiness: configurable computation*	The function allowing for the computation of trustworthiness based on different trustworthiness viewpoints ***shall*** be user-configurable to allow for different importance to be given to the different trustworthiness viewpoints.
ANIKETOS-TD-003	*Trustworthiness: incremental composition*	The design-time tools ***shall*** allow for the computation of composite trustworthiness when composing trustworthy entities (whether actor, service or resource) at design-time.
ANIKETOS-TD-004	*Trust: fusion*	Upon designer request, the design-time tools ***shall*** compute the fusion of trust relationships.
ANIKETOS-TD-005	*Trust: dilution*	Upon designer request, the design-time tools ***shall*** compute the dilution of trust relationships.

[9] The mapping ***shall*** be performed for a set of architecture frameworks as part of D6.2.

Normal Operation mode – *Trustworthiness determination – Discovery support*		
ID	**Name**	**Description**
ANIKETOS-TD-006	*Discovery support*	Trustworthiness properties ***shall*** be made available to support trust-based delegation discovery.

Normal Operation mode – *Trustworthiness monitoring* - Not applicable in this state and mode.

Failed Operation mode – *Trust modelling* - Not applicable in this state and mode.

Failed Operation mode – *Trustworthiness determination* - Not applicable in this state and mode.

Failed Operation mode – *Trustworthiness monitoring* - Not applicable in this state and mode.

4.2.2 Runtime operational state

Normal Operation mode – *Trust modelling* - Not applicable in this state and mode.

Normal Operation mode – *Trustworthiness determination*		
ID	**Name**	**Description**
ANIKETOS-TD-007	*Validation of a certificate of a service*	The verification module ***shall*** validate the certificate of a certain service upon request.
ANIKETOS-TD-008	*Validation of the certificates of a composite service*	The verification module ***shall*** validate the certificates of a composite service upon request.
ANIKETOS-TD-009	*Determining trustworthiness level*	The run-time tools ***shall*** determine the trustworthiness level of a service or a service composition. This **may** include reputation of the service (and/or service provider).

Normal Operation mode – *Trustworthiness monitoring*		
ID	**Name**	**Description**
ANIKETOS-TN-001	*Monitoring confidentiality*	The verification module ***shall*** monitor the confidentiality of the data flow within a service composition.
ANIKETOS-TN-002	*Monitoring privacy*	The verification module ***shall*** monitor the privacy of the data flow within a service composition.
ANIKETOS-TN-003	*Monitoring integrity*	The verification module ***shall*** monitor the integrity of a service or a service composition.
ANIKETOS-TN-004	*Monitoring trustworthiness level*	The run-time tools ***shall*** monitor the trustworthiness level of a service or a service composition.
ANIKETOS-	*Monitoring contracts*	The run-time tools ***shall*** monitor the

Normal Operation mode – *Trustworthiness monitoring*		
ID	**Name**	**Description**
TN-005		fulfilment of the service contracts (be the service composite or not).
ANIKETOS-TN-006	*Notification about confidentiality*	The verification module ***shall*** provide information about confidentiality of a service upon request.
ANIKETOS-TN-007	*Notification about privacy*	The verification module ***shall*** provide information about the privacy of information flow of a service or a service composition upon request.
ANIKETOS-TN-008	*Notification about integrity*	The verification module ***shall*** provide information about the integrity of information flow of a service or a service composition upon request.
ANIKETOS-TN-009	*Notification about trustworthiness level*	The run-time tools ***shall*** provide information about the trustworthiness level of a service or a service composition upon request.
ANIKETOS-TN-010	*Notify of changes in confidentiality*	The verification module ***shall*** notify if the confidentiality of a service has changed.
ANIKETOS-TN-011	*Notify of changes in privacy*	The verification module ***shall*** notify if the privacy of a communication channel is violated.
ANIKETOS-TN-012	*Notify of changes in integrity*	The verification module ***shall*** notify if the integrity of an information flow is violated.
ANIKETOS-TN-013	*Evaluation of trustworthiness level of a service*	The runtime-tools ***shall*** make the evaluation of trustworthiness level of a service based on the monitored data (to be defined) over service execution.
ANIKETOS-TN-014	*Contract status report*	A service contract has an attribute specifying its state. Possible states might be: draft, proposed, approved, invalidated (e.g. due to design-time changes), failed (e.g. due to runtime non-compliance). The run-time tools ***shall*** set the status of the contract.
ANIKETOS-TN-015	*Trustworthiness level feedback*	The monitor trustworthiness module ***shall*** provide the feedback of the evaluated trustworthiness level based on the monitored data over service execution.
ANIKETOS-TN-016	*Trust reputation*	The runtime-tools ***shall*** evaluate the service reputation to be included in calculation of the trustworthiness level of the service. Service provider reputation ***should*** be linked to the service reputation (relationship will be provided by the trust models in use).

Failed Operation mode – *Trust modelling* - Not applicable in this state and mode.

Failed Operation mode – *Trustworthiness determination*		
ID	**Name**	**Description**
ANIKETOS-TD-010	*Unavailability of the contract negotiation module*	If the contract negotiation module ***is not*** available, then the trustworthiness determination module (or the contract monitoring module) immediately reports a trust violation and checks the last known trustworthiness status of the service. If the last known status is still valid (e.g., not older than t_1), it reports this status. Otherwise, it reports a violation. After t_2, trustworthiness determination module (or the contract monitoring module) checks the status of the service again. If this fails, an error is reported. Note that the concrete values for t_1 and t_2 must be defined in the SLA.
ANIKETOS-TD-011	*Unavailability of the verification module*	If the verification module ***is not*** available, then the trustworthiness determination module (or the contract monitoring module) immediately reports a trust violation and checks the last known trustworthiness status of the service. If the last known status is still valid (e.g., is not older than t_1) it reports this status. Otherwise, it reports a violation. After t_2, trustworthiness determination module (or the contract monitoring module) checks the status of the service again. If this fails, an error is reported. Note that the concrete values for t_1 and t_2 must be defined in the SLA.

Failed Operation mode – *Trustworthiness monitoring*		
ID	**Name**	**Description**
ANIKETOS-TN-017	*Unavailability of the contract*	If the contract of any of the service components is not available, the run-time tools ***shall*** inform about this fact to aware that the trustworthiness level of the service might be compromised.
ANIKETOS-TN-018	*Unavailability of the notification module.*	If the notification module ***is not*** available, then the run-time tools ***shall*** make a back log of the monitored data in order to be able to send this information when the notification module restarts.

4.2.3 External interface requirements

External is to be understood as external to trust management: this includes interfaces internal to the Aniketos project. External interfaces for the trust management modules are specified in D1.2. This specification has been checked against the requirements listed above.

Note (as expressed in D1.2): There may be a link between the community support module and the trustworthiness prediction module if the trustworthiness information can be provided at runtime by Aniketos authority or a third party. This depends on the trust model to be developed in WP2.

4.2.4 Quality factors

No requirement has been set on trust management system extensibility or scalability. This will however need to be assessed later if industrial exploitation looks promising.

4.3 Documentation

4.3.1 System and software documentation

Please refer to D12.1 Aniketos Project Quality Handbook for the requirements in the trust management system documentation because it has to comply with the minimum quality levels of the results of the project.

Documentation		
ID	**Name**	**Description**
ANIKETOS-DOC-001	*User manuals*	All software trust modelling components ***shall*** be provided with a user manual in English.
ANIKETOS-DOC-002	*Installation manuals*	All trust modelling software components ***shall*** be provided with an installation manual in English.
ANIKETOS-DOC-003	*Trust modelling methodology*	The design-time trust modelling software component ***shall*** be provided with a methodology to elaborate the models, written in English.

4.3.2 Patterns and guidelines for establishing trust

Please refer to Section 3.2 for possible patterns and guidelines for establishing developer trust in composite services.

5 Aniketos multidimensional trust in composite services

This chapter describes the mechanisms and techniques that will be considered in Aniketos for managing service trustworthiness. It includes the definition of trust that will be adopted in Aniketos as well as the detailed descriptions of the models, metrics, and mechanisms and how they can work together to establish and maintain trust.

5.1 Trust definition in Aniketos

As seen previously in Chapter 2, trust is a concept that *lies at the intersection of several domains, including sociology, psychology, law, economics, ethics, and computer science* [46], hence both social and technical aspects of trust should be considered, as it is based on socio-cognitive and technical relationships.

When it comes to how Aniketos can support the service developers and service consumers in their trust relationships to the services (and service providers), it is important to find an interpretation of trust relevant for this project. This can be a bit difficult by just looking at the term "trust" in isolation. The domain model found in D1.2 contains an overview of key concepts of Aniketos and their relations for our problem domain. It is imperative for the project to get to a definition of service trustworthiness that can be computed by the Aniketos platform and serves as aid to the developers and the consumers in their interaction with the services.

In the following, we explain the **trust relationship** definition adopted in Aniketos for both service composition and service provisioning contexts. The main distinction between them resides in the actors and whether the service trustworthiness is being predicted (design-time) or monitored (run-time). Another important difference to note is the format the trust object takes; either the offered service provider security contract or the negotiated service security agreement, respectively.

5.1.1 Trust relationship in service composition

In Aniketos, a trust relationship needs to be established between a service developer (i.e. the trustor) and each of the component services (i.e. the trustee) when creating a composite service (creation of a composition plan at design time) The object of the trust (i.e. the trustum) is formalised in a service provider security contract (or, for short, security contract or contract template) which includes the set of security properties the provider wishes to offer to the service consumer of the component at stake. The strength of this relationship is inspired by the *predicted* service property trustworthiness (Table 6).

Therefore, for service composition, for the *X* trusts *Y* relationship described in Section 2.1.1 in Aniketos we consider the following parts:

***X* trustor**	***Y* trustee**	**in context *C***	**to do task τ trustum**	**for achieving *X*'s desired goal g_x**
Service developer	Service	Specified by the service contract	Service provider security contract	Successfully create a trustworthy composed service
Service developer	Service provider	Specified by the service contract	Service provider security contract	Successfully create a trustworthy (composed) service

Table 6: Trust relationship in service composition

Note also that usually, when creating a composite service, the service developer trusts not only the service components but also their providers, being these either individual persons or whole

organizations. The trust of the developer in that service will perform as expected is closely linked to the trust in the provider, i.e. one influences the other. The precise relationship between both will be described by the trust mechanisms in Section 5.2.

5.1.2 Trust relationship in service provisioning

As Aniketos also supports service provisioning, a trust relationship needs to be established also in this context between a service consumer (i.e. the trustor) and a service provider (i.e. the trustee). During service provision the consumer uses (consumes) the service so its composition plan is deployed. As opposed to service composition, the object of the trust (i.e. the trustum) is formalised in the service security agreement (the mutually agreed set of security properties the customer wishes to be satisfied and the provider wishes to offer by their engagement). In this case, the strength of the trust relationship is inspired by the *monitored* service property trustworthiness (Table 7).

When the execution of the service component is adherent to its security contract during service runtime, trust level is maintained. In case there are policy violations then punishments in the form of trustworthiness level reduction (of the service component or service provider) take place.

Since a service provider can provide many services, associating trustworthiness to only that entity might not be enough; some trustworthiness value should be linked to the specific services as well (Table 7).

***X* trustor**	***Y* trustee**	**in context *C***	**to do task τ trustum**	**for achieving *X*'s desired goal g_x**
Service consumer	Service	Specified by the service security agreement	Service security agreement	Successfully consume (a set of) service functionality
Service consumer	Service provider	Specified by the service security agreement	Service security agreement	Successfully consume (a set of) service functionality

Table 7: Trust relationship in service provisioning

Note that service end-users are a type of service consumer. Other service consumers are service mediators, which mediate in a service provision, because they consume a service and usually enrich it somehow for further providing it, so they also play the role of service provider.

5.1.3 Trustworthy services

The main challenge of Aniketos is to build trustworthy services. The notion of a *trustworthy service* at the most basic level can be taken as a service satisfying some minimum security requirements, most notably attestation and authorisation of service endpoints, and the use of secure communication channels between them [86], but also involves a judgement about how likely that service is to perform as claimed.

As explained in previous sections, and as pointed out Mohammad et Al [90]: "Trust is a social aspect that is hard to define formally. Trust is relative, there is no absolute trust."

Still we need to come up with an interpretation of service trustworthiness that is valid for Aniketos, because we still need to use the services that have proved they have a minimum level of acceptance, even if we know they may fail occasionally.

The definition from NSA (as quoted by Ross Anderson in [91]) also goes in this direction: "a trusted system or component is one whose failure would break the security policy, while a trustworthy system or component is one that won't fail."

In Aniketos, **trustworthiness of a service** relates to the *security decision with respect to extended investigations to determine and confirm qualifications, and suitability to perform specific tasks and responsibilities* (as cited in [49]). Therefore, service trustworthiness is something that can be computed, measured or cognitively estimated in order to evaluate to what degree a service should be trusted. Service trustworthiness is associated to metrics (see below), measuring one or a set of properties of the service.

Therefore, the **trustworthiness level of a service** in Aniketos is a combination of both:

- **Cognitive trust** of the user (service consumer or service developer) on the service that is based on reputation derived from:
  - previous experiences of the user (direct trust),
  - recommendations given by a user, or by a community of users, on the service (indirect trust) and
  - other particular and unpredictable subjective metrics/aspects that are of important for the user for evaluation of trust.
- **Non-cognitive trust** is associated to objective and measurable properties of the service (see trust metrics). Depending on the type/nature of the service some metrics might apply and some others not.

The Aniketos platform will provide a means to support service trustworthiness evaluation and monitoring (non-cognitive trust) and will help the end users in the process of trust decision (cognitive trust) when developing or consuming services.

When monitoring a service, Aniketos service consumers will get some information on whether the service is performing as expected with regards to its security behaviour. In this case the trustworthiness level evaluated by the Aniketos platform is calculated from a set of trust metrics obtained during the execution of the service and influences the notifications the consumer gets.

When creating the service, Aniketos service developers will be informed on a predicted trustworthiness level of the service with respect to whether it has the ability to perform as expected, especially in relation to its security behaviour. The set of trust metrics computed for this does not need to be exactly the same as during service consumption. Therefore, two different trustworthiness levels will be associated to a service, one at design time (predicted) and one at run-time (monitored).

5.1.4 Trustworthiness metrics

The goal of having a computed or evaluated level of service trustworthiness in the Aniketos platform means that we should be able to define a set of attributes on certain service properties (characteristics) that influence its trustworthiness. For each of the attributes we should also be able to define metrics (the so called “trust metrics” or better “trustworthiness metrics”) which combined provide a measurable quantity of the service trustworthiness. Such metrics should be defined in a way that can be measured somehow (e.g. a reputation level, the strength of evidence, or the economical value of a promise/guarantee, etc.).

In the quality model of the ISO/IEC 9126 Software engineering -- Product quality standard [66] presented in Figure 10, six main internal (related to interim products) and external (related to the behaviour of the code when executed) quality characteristics that influence a software product quality are defined (the upper part of Figure 10). Each characteristic is further subdivided into sub-characteristics (the lower part of Figure 10) and each sub-characteristic can be measured by internal or external metrics of the attributes that influence that particular sub-characteristic.

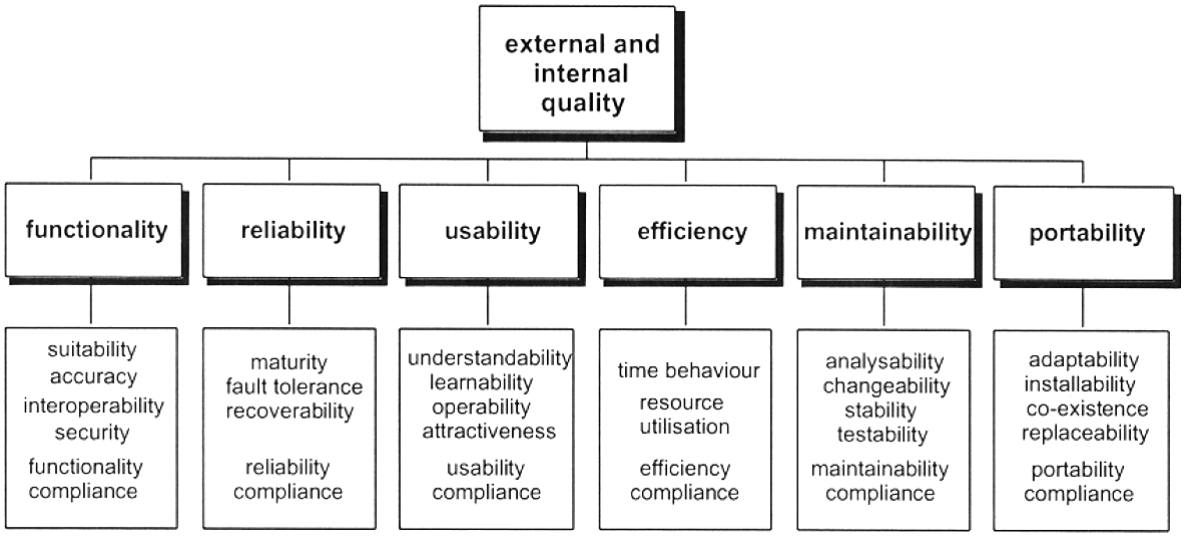


Figure 10: Quality model for external and internal quality (source: ISO/IEC 9126)

The ISO/IEC 9126 standard defines also the quality in use characteristics (see Figure 11) which relate to the user view of product quality in specific environments and contexts of use.

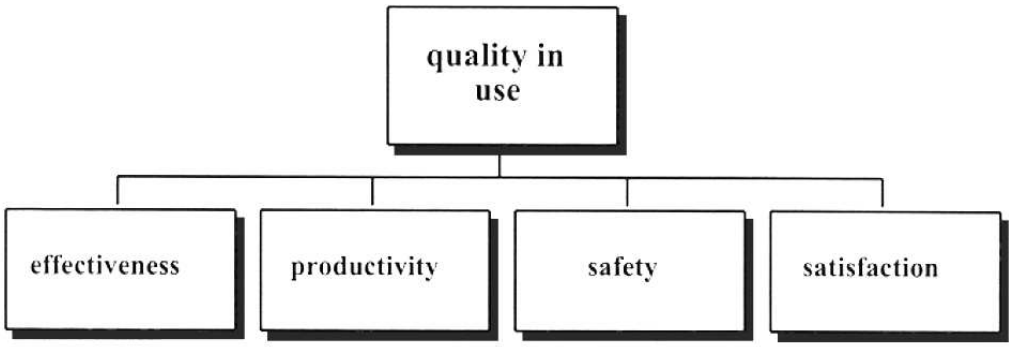


Figure 11: Quality model for quality in use (source: ISO/IEC 9126)

In Aniketos we will identify which of these characteristics and sub-characteristics (properties) will be considered as part of the trust models we will develop, which have influence on the trustworthiness level of the service.

Considering the three viewpoints described of the ISO/IEC 9126 quality model (internal, external and in use), the set of metrics that serve to inform the service trustworthiness computation varies depending on which stage in the service lifecycle we are in. Additionally, depending on the type or nature of the service some metrics might apply and some others not.

The service properties that we would like to consider in Aniketos for evaluating the service trustworthiness are the following:

a. Authentication (the service is authenticated, its identity and authenticity verified). This will be managed by the Identity Management service of the Environment.
b. QoS aspects (see Section 5.2.2) including:
    1. Performance: measured through response time attribute.
    2. Reliability: measure through rate of successful completion of a service execution.
    3. Availability: measured through uptime attribute.
    4. Security: the following attributes may be considered:
        a. Encryption: indicating secure exchange of messages using encryption.

b. Authentication: indicates whether authentication mechanisms are used.
c. Privacy: indicates which parties can have access to the operations and messages.

5. Cost: the execution cost of a Web service.

c. Reputation of the service and service provider.. Domain general factors for reputation-based models include (see Section 6):

- Certificates
- Time complexity /space /algorithm /architecture complexity
- Quality and frequency of updates
- Software development process
- Personal relation to the developers
- Formal verification of trustworthiness properties
- Quality of documentation
- Frequency of usage of a service
- Security incident history

Eventually, other subset of metrics related to domain-specific trustworthiness factors identified in Section 3.1.1 could also be included. This subject is still under consideration due to the difficulties the implementation of some metrics could bring into the platform construction in the project time frame.
The challenge resides obviously in defining a methodology to measure such metrics and the elaboration of trust models that combine these attributes, some objective (e.g. lines of code), some perhaps subjective (e.g. quality of documentation), and some that could either increase or decrease trustworthiness depending on the trustor's point of view (e.g. cost).

5.2 Trust models and mechanisms

This section describes Aniketos models, metrics, techniques and mechanisms for establishing and maintaining trust in composite services. The models take into account the service compositionality.

5.2.1 Contract-based trust and security

The Security-by-Contract-with-Trust paradigm (which serves also as a basis for some of the security analysis techniques developed within WP3, see [D31] for details), S×C×T for short, has been recently proposed [67, 68] as a unique framework for managing both security and trust at application execution time. One of the main differences between the Security-by-Contract paradigm and the Security-by-Contract-with-Trust approach is that the code of the application also plays a central role in the S×C×T workflow, depicted in Figure 12. Indeed, the function "check-evidence" is replaced by a trust module that according to the adherence of the application to its contract[10] updates the level of trust of the application itself. It is interesting to note that updating the level of trust of the application implies updating the level of trust of the provider of the application since the device receives the couple application-contract from the provider.

[10] In contrast to ANIKETOS project, in this work we do not distinguish between "agreement template", as the list of policies which are guaranteed by the provider, and "contract", as a list of policies application provider and application consumer 'agree' on. It is not important to highlight this difference in scope of the work done, because we do not consider explicitly the selection of the most suitable application, but only focus on analysis of a selected application.

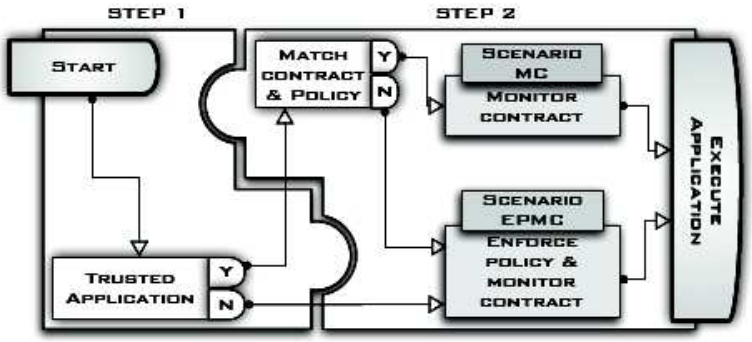


Figure 12: The S×C×T application deployment workflow

The basic idea is the following one: let us consider having a device and let us suppose we want to run an application on it, developed by possibly unknown developers. As in the S×C paradigm, we assume that contracts and policies are specified through the same formalism. According to Figure 12, the code is downloaded with its contract. The level of trust is checked, this means that we measure the level of trust that the code adheres to its contract. If the check fails, the code is considered untrusted, so on one side the policy is enforced in order to guarantee security issues, on the other side the contract is monitored in order to log the contract violations. If the monitored execution does not violate its contract the level of trust is upgraded, otherwise it is downgraded. Otherwise, if the code is trusted, the compliance between the contract and the policy is checked in order to understand if the application can be executed without any enforceable mechanisms running on it.
Going into more detail, the application lifecycle consists of the following steps:
Step 1- Trust Assessment: Each downloaded mobile application comes with a given recommendation rate, which allows the trust module of the user device to decide if the application can be considered as trusted or not.
Step 2- Contract Driven Deployment: According to the trust measure, the security module decides if just monitoring the contract or both enforcing the policy and monitoring the contract going into one of the scenarios described in Step 3.
Step 3-Contract Monitoring vs. Policy Enforcement Scenarios: Depending on the chosen scenario the security module is in charge of monitoring either the policy or the contract and saving the execution traces (logs). Indeed, we have two sub-cases representing the Contract Monitoring (CM) and the Policy Enforcement & Contract Monitoring (PECM) configurations:

- **Contract Monitoring Scenario (CM).** The monitoring/enforcement infrastructure is required to monitor only the application contract. Indeed, under these conditions, contract adherence also implies policy compliance. If no violation is detected then the application worked as expected. Otherwise, we discovered that a trusted party provided us with a fake contract. In more detail, the contract monitoring works according to the following strategy depicted in Figure 13a. The execution trace is kept in memory. When a signal arrives, its consistency with respect to the monitored contract is checked. If the contract is respected then its internal monitoring state is updated and the operation is allowed, and a good behaviour is logged (i.e. contract respected). Otherwise, if a violation attempt happens, a security error occurs, and a bad feedback is trigged (i.e. contract violation), and the system switches from contract monitoring to policy enforcement configuration in order to guarantee that the security policy is satisfied. Since an instance of the policy is always present, this operation does not imply a serious computational overhead.

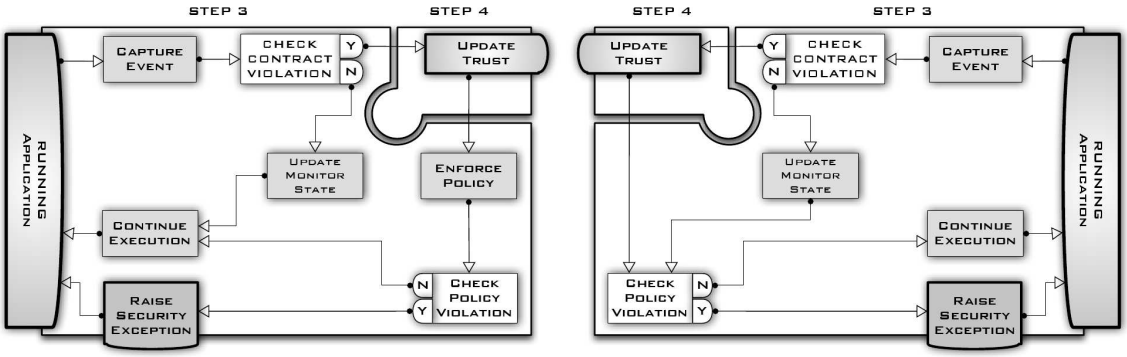


Figure 13: Workflows for CM (a) and PECM (b) scenarios

- **Policy Enforcement & Contract Monitoring Scenario (PECM).** Since the contract either declares some undesired behaviour or comes from an untrusted source, the policy enforcement is turned on. Similarly to a pure enforcement framework, our system guarantees that executions are policy-compliant. However, monitoring contracts during these executions can provide useful feedback for better tuning the trust vector. Hence, in this scenario, both the policy enforcement and the contract monitoring are active. Indeed, the contract monitoring receives event signals from the executing code and keeps trace of the execution trace. When a signal arrives, its consistency with respect to the monitored contract is checked. If the contract is respected then its internal monitoring state is updated and the operation is allowed, and a good behaviour is logged (i.e. contract respected). Otherwise, if a violation attempt happens, a security error occurs and a violation feedback is logged for the trust module. The policy enforcer is only in charge of following the execution of the application and whenever it attempts to violate the security policy of the device the enforcement mechanism halts the execution in such a way that the security policy is satisfied. This configuration is activated on a statistical basis (Figure 13b).

Let us notice that, in both the previous scenarios, contract monitoring plays a central role. Indeed, a contract violation denotes that a trusted provider released a fake contract.

Step 4-Trust Feedback Inference: Finally, the trust module parses the S×C×T produced logs and infers trust feedback.

5.2.1.1 Certifying Service Implementations

Both, the Security-by-Contract paradigm and the Security-by-Contract-with-Trust paradigm rely on a certification that proves that an actual service implementation adheres to its security or trust objectives, i.e. its contract. In principle, the following techniques can provide such a certificate:

- *Random test case generation (RTG),* i.e. the generation of random test cases that are used for checking if a implementation fulfils its contract or crashes (fuzz testing).
- *Specification-based test case generation (STG),* i.e. the generation of test cases (sequences of system operations) based on the service contract that can be used to validate that an implementation fulfils its contract.
- *Static code analysis (SCA),* i.e. techniques that statically (without actually execution) analyse the source code (less preferable: byte code) of a service using formal techniques such as abstract interpretation.
- *Manual code inspection* (MCI) or a manual code review may result in accredited certification, e.g., following Common Criteria.
- *Formal verification* (FV), of a service implementation using techniques such as model-checking or (interactive) theorem proving. As such techniques require a detailed formal specification together with a formal semantics of the programming languages used, the formal verification is very time consuming and, therefore, expensive.

These techniques differ (see Figure 14) both in the amount of human expertise and work required for applying them as well as in the level of assurance they can give for the absence of security and trust related problems of the analyzed service.

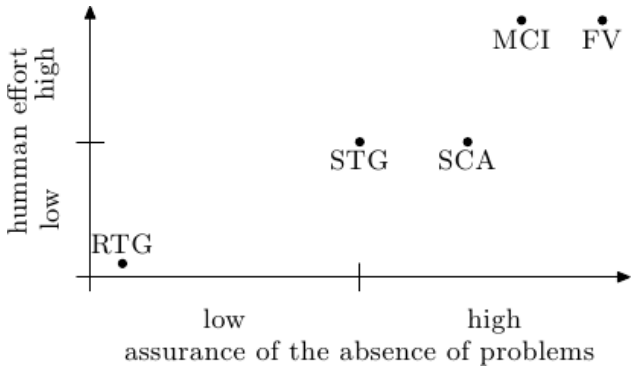


Figure 14: Comparsion of the different techniques for certifying service implementations.

As static code analysis provides a high level of assurance while only requiring a moderate level of human effort and expertise, we focus on applying static code analysis techniques for ensuring the security and trustworthiness of services within the Aniketos framework.

Static code analysis is a method for finding (semi-) automatically potential security and trust problems in service implementations without actually executing the implementation. Analysing a implementation that requires the execution of the implementation is called *dynamic program analysis*. The simplest form of a static code analysis is a textual search (e.g. similar to the Unix utility grep) for "dangerous commands" such as commands that allow for directly executing arbitrary system commands. Of course, such a text search that ignores the context in which a command is used is usually not sufficient. On problem with such simple approaches is the generation of many false positives, i.e., such methods report a large number of potential problems that are no problem because there is no possibility for an attacker to exploit the use of a "dangerous commands". Therefore, state of the art static code analysis tools try to take the context in which a potential dangerous programming construct is used into account.

Static code analysis approaches (see [75, 77] for an overview of such approaches) for analysing security and trust properties usually work on abstractions of the source code such as control flow graph (i.e. a compact representation of the set of possible execution path of a method) or call graphs (i.e. a representation of the potential control flow between different methods). Based on those graphs, the (potential) dataflow of a program can be analysed. The dataflow analysis gives detailed information on how data flows from the source (e.g. user input) to the various sinks (e.g. output to the user or storing the data in a database) or potential dangerous commands. For each of these flows, the static analysis tools needs to check, if the required security and trust checks (e.g. requiring authorization, checking credentials, etc.), respectively, counter-measures (such as input sanitation) are taken or not. Tsipenyuk et al. [76] present a taxonomy of the different security and trust problems that are usually tackled by static code analysis approaches. Chess and Jacob [78] give a detailed overview of static code analysis for security and trust related properties, thus, the interested reader is referred to [78].

Within Aniketos, we will focus on the development of static code analysis techniques that exploit the semantics of the analyzed implementation (this is in contrast to most of the state of the art tools which usually only analyse the implementation on a purely syntactical level). The use of formal analysis techniques on a semantically level should help to reduce the number of reported false positives (i.e., reported problems that, after a manual inspection, are classified as not being a trust or security issue). Moreover, within Aniketos we will concentrate on the development of static analysis techniques for trust and privacy properties - this, as well, extends the state of the art which focuses on the detection of

exploitable security problems such as SQL injections. The underlying formal techniques will be aligned with the techniques used for analyzing service compositions (see [D31] for details).

In general, Security-by-Contract-with-Trust is an important part of ANIKETOS platform (e.g., see D2.2 and D2.3). Although it is used in ANEKETOS as a separate module, it benefits from and contributes to trustworthiness computation. First of all, SxCxT needs reliable trust values (e.g., determined by methods described in Section 2.3.3) in order to make a decision whether security properties must be enforced. As it has been discussed above, if an application comes from a trusted source only contract must be monitored. This idea is highly important in the context of services, where direct access to invoked services is often limited and some properties cannot be checked. Secondly, SxCxT model shows how trust values are updated. The application which behaves according to the contract increases its reputation, while violation of a contract is punished by reducing reputation (see Section 5.2.2.3). Finally, as it has been mentioned SxCxT could benefit from usage of certificates which increase the trust in correct implementation of declared policies (see Section 2.3.2).

5.2.2 QoS and reputation

This subsection includes Aniketos mechanisms for quantitive trust computation based on user feedback and reputation in the service user communities and on QoS attributes. QoS and reputation are important to determine the trustworthiness of a component service and the expected trustworthiness of a composition of those services. Section 2.3.3 investigated existing work in this area and how Aniketos can benefit from their strengths and weaknesses. As discussed in that section several service QoS attributes and their classifications have been identified in the literature. The following are the categories of QoS that will be considered:

- Performance: measured through response time attribute.
- Reliability: measure through rate of successful completion of a service execution.
- Availability: measured through uptime attribute.
- Security: the following attributes may be considered:
    - Encryption: indicating secure exchange of messages using encryption.
    - Authentication: indicates whether authentication mechanisms are used.
    - Privacy: indicates which parties can have access to the operations and messages.
- Cost: the execution cost of a Web service.

These categories are used as a preliminary set and can be extended or modified as required. The ratings of a component service are generated from the metrics data by the trustworthiness monitoring module. Some service attributes may be evaluated without runtime monitoring such as cost and security properties. Therefore, the attributes may be classified into two nonexclusive sets of design-time and runtime category of attributes.

5.2.2.1 Business process models

The calculation of the trustworthiness of the composite Web service depends on the way the composite service is constructed. Component Web services may be invoked in a business process in one or more of path constructs that include the following basic and other commonly supported constructs:

- *Sequence:* services are invoked sequentially one after the other.
- *Parallel with synchronisation (AND split/AND join):* two or more services are invoked in parallel and their outcome is synchronised. All services must be executed successfully for the next task (service) to be executed.
- *Loop or iteration:* a service is invoked in a loop until a condition is met. We assume that the number of iterations or its average is known at the time of composition or it follows a particular distribution.

- *Exclusive choice (XOR Split/XOR join):* a service is invoked instead of others if a condition is met. We assume that the probability of each alternative service is to be invoked is known at the time of composition.
- *Discriminator (AND split/OR join):* two or more services are executed in parallel but no synchronisation of the outcome of their execution.
- *Unordered sequence:* multiple services are executed sequentially but arbitrarily.

These and several other possible patterns are investigated in Workflow Patterns Initiative [74]. In WP3, formal techniques for analyzing security properties of business processes (representing service compositions) will be developed (see [D31] for details).

5.2.2.2 Computing QoS and reputation of service compositions

The computation assumes that the QoS, reputation, other metrics are available for component services. Each constituent of the business process will require its trustworthiness computation technique. Note that for the security attributes we follow the weakest link approach where a service composition is as secure in relation to a particular security attribute as its component service with the weakest metric of that security attribute.

$$\min_{s \in S} Sec(s)$$

where *Sec* is a rating of a security attribute.

The aggregation functions per process construct for each of the proposed QoS attributes and for the reputation are as follows (see Table 1 for summary):

Sequence: For a sequence of services the cost and response time are the sums of respective values of component services. Reputation, reliability, and availability are the product of the values of constituent services respectively.

Parallel: The cost of the execution of multiple concurrent services is the sum of their cost. The response time however is equal to the longest response time since the next step cannot proceed until all parallel services have successfully executed. Reputation, reliability, and availability are the product of the values of constituent services respectively.

Loop: The reputation and QoS attributes of a loop construct of n iterations of a service *s* is the same as a sequence construct of n copies of *s*.

Exclusive Choice: each service *s* in the alternative services set *S* has a probability $\rho(s)$ that it will be executed and $\sum_{s \in S} \rho(s) = 1$. For all attributes in this construct the aggregation is the sum of corresponding attributes of component services multiplied by its probability.

Discriminator: the cost of a discriminator construct is the sum of costs of the services. The response time is equal to the minimum response time since the first service completes marks the start of the next task. The construct only fails if all constituent services fail. Hence, reliability $p_{discriminator} = 1 - \prod_{s \in S}(1 - p(s))$ and similarly for reputation and availability.

Unordered sequence: Reputation and QoS attributes in unordered sequence is dealt with in the same way as sequence constructs.

Construct	***Reputation (r)***	***Response time (t)***	***Reliability (l)***	***Cost (c)***
Sequence	$\prod_{s \in S} r(s)$	$\sum_{s \in S} t(s)$	$\prod_{s \in S} l(s)$	$\sum_{s \in S} c(s)$
Parallel	$\prod_{s \in S} r(s)$	$\max_{s \in S}\{t(s)\}$	$\prod_{s \in S} l(s)$	$\sum_{s \in S} c(s)$
Loop	$r(s)^n$	$n \cdot t(s)$	$l(s)^n$	$n \cdot c(s)$

Construct	***Reputation*** ***(r)***	***Response time*** ***(t)***	***Reliability*** ***(l)***	***Cost*** ***(c)***
Exclusive choice	$\sum_{s \in S} \rho(s) \cdot r(s)$	$\sum_{s \in S} \rho(s) \cdot t(s)$	$\sum_{s \in S} \rho(s) \cdot l(s)$	$\sum_{s \in S} \rho(s) \cdot c(s)$
Discriminator	$1 - \prod_{s \in S} (1 - r(s))$	$\min_{s \in S}\{t(s)\}$	$1 - \prod_{s \in S} (1 - l(s))$	$\sum_{s \in S} c(s)$
Unordered Sequence	$\prod_{s \in S} r(s)$	$\sum_{s \in S} t(s)$	$\prod_{s \in S} l(s)$	$\sum_{s \in S} c(s)$

Table 8: Aggregation functions per process construct

5.2.2.3 Calculation of the overall trustworthiness

Service ratings can be generated based on QoS and reputation metrics. A service rating measures how a metric adheres to or violates service contract specifications (or more generally acceptable quality values). Aggregation of the metrics for a service composition is discussed in the previous section. The mechanisms for the aggregation of the metrics, generating ratings and the calculation of the trustworthiness are implemented in D2.2.

For a service in general (i.e. both component and composite) the overall trustworthiness in Aniketos may be calculated from its ratings using a number of approaches such as:

- Exponential moving average of the ratings:

  $$T = \propto \cdot R + (1 - \propto) \cdot T$$

  where T is the trustworthiness score, R is the new rating, and $\propto$ is the rate of adoption of new ratings ($0 \leq \propto \leq 1$) which can depend on the recency of last calculation of trustworthiness and the new rating's priority.
- Weighted average of the existing ratings. Weighting can be based on some criteria such as recency of each rating and the priority of the type of that rating. This approach allows more dynamic customisation of weights but is not as scalable.

Subsection 2.3.3.4 discussed the credibility of ratings. A credibility or confidence value indicating certainty about the trustworthiness evaluation can be calculated based on the quantity and quality of information available about a service. For example, significant variations between the values of service ratings results in reduction of the credibility. This may be calculated using the following formula:

$$c_\sigma = 1 - \frac{\sum_{i \in n} \omega_i \cdot |S - s_i|}{\sum_{i \in n} \omega_i}$$

where c_σ is the ratings deviation credibility, w_i is the weight of the rating s_i, and $|S - s_i|$ is the absolute difference between s_i and S is the trustworthiness score.

5.2.2.4 Eliciting user feedback

A variety of methods can be used to elicit trust estimates, e.g., discrete scales with 5 to 10 points or a visual analogue scale, where (in principle) infinitely many points are available between two extreme positions. Questions can ask people to:

- rate how much they agree (e.g., from "completely disagree" to "completely agree") with a statement about a service, such as how well it satisfies the service's advertised specification;
- score a service on some reputation-relevant quality, such as usability;
- rate how sure they are about their rating.

The number of items used affects the possible variability in responses. For instance a single 5-point discrete item clearly offers only 5 possibilities, so it may be difficult to distinguish between services. Two 5-point items provide 25 levels. In general a scale comprised of *p*-point scales and *i* items will have p^i different values. In practice, fewer variations will arise if item responses are correlated, for instance if they ask the same question in only superficially different ways, or (more interestingly) if they tap into a single underlying latent construct. For an extreme example, 100 items with 5-point responses which perfectly correlate will lead to only 5 possibilities. However theory-driven between-item correlation can be useful as it can be used to average out random measurement error. Principle components analysis is commonly used to detect such correlations and to extract a value representing the shared variance of a collection of items which tap into the same construct.

Care must be taken distinguishing between ratings of a quality and epistemic certainty about a rating of a quality. When people are presented with a scale concerning a probability, 50% is sometimes used to express "don't know" [83], which is incorrect. Knowing that an event has .5 probability is informative. Consider a repeatedly flipped coin for which we know *P*(heads) = .5. We know the coin will eventually land heads – around half the time over a long number of trials. If we have no information about the probability, that would be expressed as an interval, *P*(heads) ∈ [0,1]. The coin could systematically land heads or systematically land tails – the interval expresses that we don't know anything about its likely behaviour. Similar problems occur in the domain of ratings of trust leading to reputation metrics. For instance rating trust in an interval from 0 to 1 leaves ambiguous what zero denotes; it could mean either complete distrust or no trust, i.e., complete uncertainty about trust [84].

5.2.2.5 Understanding and reasoning about uncertain information

Inferring the trustworthiness of a service in Aniketos based on a sample of opinions about reputation and data on quality of service (QoS) is an uncertain inference problem. People are generally not very good at solving such problems. In one infamous study, only around 20% of medical students and qualified doctors were able to solve a simple probability problem concerning diagnosis [81]. There are inter-individual differences in performance on a range of reasoning tasks, many but not all related to general intelligence [80]. Also how information is represented can have an effect on how well people can reason about it, including the diagnosis problem which presented so many difficulties [82].

Reputation ratings typically involve the reduction of many individual ratings to one or more summary values, for instance to the mean rating. Doing so removes information about uncertainty in the measurement, so for instance a service with a mean rating of 4 out of 5 might not be statistically significantly different from a service rated 3 out of 5, leading to a service being unfairly chosen over another. Even where differences are statistically significant, they may not be practically meaningful. A difference in mean rating between 4.666 and 4.667 is an extreme example.

If there are qualitatively different groups of raters, e.g., due to a mixture of experienced and novice developers, then this may be reflected in multimodal distributions (see Figure 15 for a simulated example based on 100 draws from a mixture of beta distributions *Be*(8,1) and *Be*(1,8), rescaled from [0,1] to [0,5] and discretized), in which case taking a single summary value is misleading and may not summarise either of the groups. In the example, there are two modes, one at 0 at the other at 5. The mean is 2.5 and median 3, where few values lie.

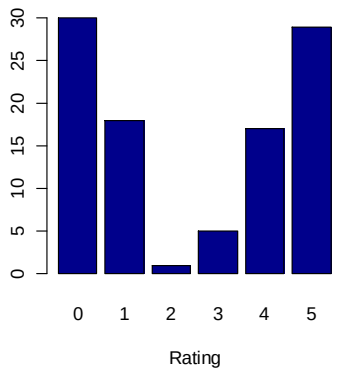


Figure 15: Example of bimodal reputation distribution

There will be correlations between ratings for separate services, for instance if users rate more than one service or if a composite service repeatedly contributes QoS data about one of its services. This means that the data points will not be independent, introducing a bias unless the error structure is correctly modelled.

Although QoS and usage data are more objective than self-reported reputation data, the problem of inferring from sample-to-population remains. A difference between services in terms of recorded QoS does not imply that the services differ in their ability to provide an equally good QoS. The problem of correlation structure rears its head here too. For instance a single composite service can unduly influence the reputation or QoS data of the services it composes if

- the composite service is used more frequently than other composite services;
- a contextual factor, such as the quality of a network connection, is the (unknown) cause of a reported poor trustworthiness or poor QoS.

5.2.2.5.1 Count data

The simplest instance of count data consists of pairs of positive and negative cases, representing, e.g., how many times a request to use a service has and hasn't been successful. It might be compelling to represent these data as only percentages; however doing so removes information about uncertainty due to noise.

For instance, suppose service *S1* was rated positively 70% of the time and *S2* 60% of the time. If these percentages were based on 20 ratings, 10 for each services, the 95% Confidence Interval (CI) of the *difference* in the population percentages ranges from −42% to +62%, so we cannot reliably infer which service has a better rating. If there are 200 ratings, then the difference ranges from −4 to +24%, which is in favour of *S2*, but still includes 0, i.e., the data are consistent with no difference in the population. With 2000 ratings, the difference 95% CI is 6% to 14%, so now we can be surer that *S1* is better than S2.

The situation is similar for data concerning how often each of *N* services has been used. For instance suppose there are three services, *S1*, *S2*, and *S3*.

- *S1* has been used 6 times
- *S2* has been used 3 times
- *S3* has been used 3 times

The probability of this distribution of responses, given the null hypothesis that in the population usage of all the services is equal, is around 0.6. The percentages of usage were kept constant (50%, 25%, 25%) but the number of observed instances increased to demonstrate the effect on *p*-values – see Table 9. The $\chi^2(df = 2)$ distribution was used for all but the first example (which was computed by simulation with 2000 replicates). By normal conventions, the example with 48 observations in total (24,12,12) is unlikely given the null, so we could justify choosing *S1* over the other two services.

S1	*S2*	*S3*	*Probability of sample distribution, given equal usage in population*
6	3	3	0.61
12	6	6	0.22
18	9	9	0.11
24	12	12	0.05
30	15	15	0.02
36	18	18	0.01

Table 9: Effect of (hypothetical) number of service uses on ability to infer population differences in services

Recommendation: present integer counts of both successes and failures. If percentages are used, then include also the 95% CIs (see, for instance, [79], for a comparison of methods) and the number of observations.

5.2.2.5.2 Interval data

Rating data is often on a bounded scale, e.g., from 0 to 5, or – especially for visual analogue scales – 0 to 100. Typically such data are analysed as if they were Gaussian distributed in which case the 95% CI for the mean is computing using the *t*-distribution,

$$[\text{mean} + t(df = N\text{-}1, quantile = .025) \times SE, \text{mean} + t(df = N\text{-}1, quantile = .975) \times SE]$$

Or equivalently, since the distribution is symmetrical,

$$\text{mean} \pm t(df = N\text{-}1, quantile = .975) \times SE$$

where N is the number of observations and SE is the standard error of the mean, $\frac{SD}{\sqrt{N}}$. The *t*-distribution may also be used to infer population differences between ratings based on the sample using a *t*-test. An approximation for the 95% CI is given by the normal distribution, in which case the intervals are the mean ± 1.96 × *SE*, which is equivalent to the t-distribution as *df* tends to infinity.

Discrete ratings cannot have a Gaussian distribution; however in practice the statistics are sufficient. For border cases, treating bounded rating data as if it were Gaussian may give inaccurate intervals as the distribution is skewed. For instance simulating responses on a 0 to 5 scale from 100 users (based on 100 draws from the beta distribution, *Be*(8,1), rescaled from [0,1] to [0,5] and discretised) gives the distribution in Figure 16 which is far from the shape of a Gaussian.

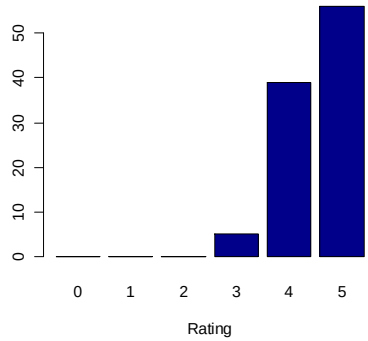


Figure 16: Example of skewed reputation distribution

Recommendation: present rating data in a way which highlights variability and uncertainty in the estimation of the statistic chosen (e.g., mean/median).

6 Conclusion

This chapter examines the coherence and integration of the mechanisms in managing trust in service compositions and aligns these techniques with the architecture of the Aniketos platform given in D1.2.

In Chapter 4, we discussed which requirements as identified in D1.2 are relevant for the choice of suitable trust models. A key result of the requirements analysis is the need for different trust models using different definitions of trust. In Chapter 2, this deliverable describes several models for trust and trustworthiness. Focusing on the introduced models, all utilize different trust interpretations. So, a wide range of trust aspects (e.g. technical vs. social trust) is covered. One goal within the Aniketos project is the combination of these trust definitions to provide a comprehensive overview of trust in the service context. The benefit of the combination is a measurement of trust that relies on different trust interpretations and can be used to validate the trustworthiness of a service.

The goal of all trust management approaches is the reliable answer to the question if a service or a service composition is trustworthy. This answer is determined by the actual behaviour of a composite service combined with the evaluation of the behaviour of its component services in the past.

To evaluate the past behaviour of a service, the architecture of the Aniketos platform as described in D1.2 provides the trustworthiness prediction module. This module computes the current trustworthiness of a service based on the trust relationship and trust models in use. Thereby, this computation utilizes information about the past behaviour of a service. It also estimates the overall trustworthiness of a composite service based on a given composition plan.

The trustworthiness level is a ratio including various trust metrics and scalable results of trust mechanisms. The described models therefore provide mechanisms and metrics to estimate a scalable level of a well-defined trust definition.

The application of the trustworthiness prediction module is necessary at design-time as well as at runtime. During design time, it provides information about the trust of a service or a service composition based on gathered data. During runtime, the actual behaviour of a service composition influences the results of the computed trust level.

So, the trustworthiness prediction during runtime relies on the monitoring of a service during runtime. This task is provided by the monitor trustworthiness module. As described in the architecture of the Aniketos platform in D1.2, this module enables runtime monitoring of trustworthiness based on the mechanisms and metrics defined according to the trust model. If changes in trustworthiness defy the required levels in a contract, then the affected parties are notified by the notification module. This module also invokes the trustworthiness prediction module to inform about changes of the trust level of a service composition.

In this deliverable we have considered two different types of trust models which offer a different point of view on trust. In Chapter 3, we have examined user-centric trust and on the other hand we have discussed trust relying on technical aspects. Therefore, we differentiate between cognitive models taking a user-centric perspective and non-cognitive models of trust which have a technical perspective.

As mentioned above, cognitive models take a user-centric perspective. This means that they incorporate user' experiences into mechanisms for establishing trust among users. This approach states that trustworthiness is highly influenced by the perception of the user and so defines trust as a result of non-technical factors. This type of model bases the trust level of a service on the perception of the users.

Social and cognitive models as refined in Chapter 2 and 5 focus on the trust perception of services developers and services designers. As these people are composing black box services, this may draw inferences about trustworthiness. Therefore, this deliverable identified a first draft of domain-general factors to study their influence on trust, trustworthiness, and reputation of service developers and designers. These factors are listed with a short explanation in Table 10.

Domain general factor	Critical influences on trust, trustworthiness, or reputation
Origin and quality of certificates	Is a certifying authority trustworthy?
Time and space complexity	Does the performance of a service influence its trustworthiness?
Architecture complexity	Does the composition reveal untrustworthy links?
Algorithm complexity	Does a complex metric to compute a trust level has an influence on the user validation?
Frequency and quality of updates	Are the updates on a regular basis? If a new security problem arises, are there timely updates? How is the quality?
Software development process	Is a software open source? Is its development process documented?
Personal relation to the developers	Does a personal contact to the developers influence the level of trust?
Formal verification of trustworthiness properties	Does the application of formal methods influences the level of trustworthiness?
Quality of documentation	Is the interface of a service well-documented?
Frequency of usage of a service	Is the service well-known to a user?
Security incident history	Is a service known for security incidents?

Table 10: Domain general factors

Reputation-based trust models rely on the idea that the history of a trustee can reflect its future trustworthiness. This history is recorded based on user experiences and can be expressed as a scalable rating of the past. Three main models are described:

- Online reputation models for products and services
- Models for P2P and multi-agent systems for peers
- Models for services

In reputation-based trust models, the trustworthiness of service depends on its history. A user will trust a service if the past behaviour shows that this service can be trusted.

A second type of a cognitive model analyzes the usage of patterns and guidelines for trust in composite services. Patterns and guidelines propose a set of instructions to achieve a certain security objective. In this deliverable, we address only a special kind of patterns. All presented patterns focus on tasks related to technical aspects. This means that a certain set of instructions can be implemented on a technical level and its achievement can be checked.

Non-cognitive trust models provide mechanisms for establishing trust among entities and verifying the identity of participating entities and their credentials and characteristics. These are described in Chapter 2. The mechanisms used in these models validate the application of security or other technical properties. Focusing on policy-based trust models, the trustworthiness of a service is analyzed on a technical level. This means that the trustworthiness of a service relies on the fulfilment of certain specifications of the behaviour of a service. To validate if a service behaves in a predefined way, it is necessary that the specification can be validated automatically to simplify the check.

Nevertheless which specification is demanded by a service depends on the usage scenario where this certain service is applied. Using a secure channel to broadcast financial data requires other specifications than upload holiday pictures. Moreover, there may be additional aspects influencing the demanded specifications:

- legal restrictions (compliance)

- service level agreements of a company
- personal security goals of a user

In the context of this deliverable, we have studied policy-based trust models to ensure trustworthiness on a technical level.

This means that a service is considered trustworthy if certain technical specifications are provided. The user of a service defines the specifications of acceptable behaviour and describes these in a policy. Trustworthiness in the context of a policy-based model means that a consumer's policy has to be fulfilled by the agreement template of a service, e.g. the agreement template is matching the policy. So, a service behaves according to the specified requirements of a user and this matching provides a contract between user and service provider defining the behaviour of this service.

This guarantee relies on two conditions. First, the agreement template has to be truly fulfilled by the service. This means that a service has to match the specifications its developer has been promised. This requires an additional check. Within the Aniketos project, we plan to check this kind of matching by static analysis. Second, the policy of a user has to be evaluated. This evaluation of security policies can be given by a trusted third party based on security standards as described in Section 2.3.2.

Cognitive and non-cognitive models aim to establish trust by users towards composite services, but focus on different perspectives of trust. Cognitive models see a trustworthy relationship among users as a perception gained by the involved parties. The quality and reason of these perceptions are studied within these models. This user-centric perspective is complemented by the non-cognitive models to broaden the view of the characteristics of a trustworthy relationship. These models provide validation that a service fulfils defined requirements. This guarantees a certain well-defined behaviour of a service. The combination of non-cognitive as well as cognitive models in the Aniketos project offers a wide perspective of various characteristics which can be taken into account when establishing a trust relationship.

References

[1] Aniketos EU FP7 project, http://aniketos.eu/home, last accessed July 2011

[2] R. Henning, “Security service level agreements: quantifiable security for the enterprise?”, In Proc. of the 1999 Workshop on New security paradigms, pp. 54–60. ACM Press, 2000

[3] V. Casola, A. Mazzeo, N. Mazzocca, and M. Rak, “A SLA evaluation methodology in Service Oriented Architectures”, In Proc. of the 1st Workshop on Quality of Protection., Milan, Italy, 2005. Springer-Verlag

[4] L. Krautsevich, A. Lazouski, F. Martinelli, and A. Yautsiukhin, “Risk-based usage control for service oriented architecture”, In Proc. of the 18th Euromicro Conf. on Parallel, Distributed and Network-Based Processing, IEEE Computer Society Press, 2010

[5] ISO27001, Available via http://www.iso27001security.com, last accessed July 2011

[6] CRAMM (CCTA risk analysis and management method), http://www.cramm.com/

[7] E. Johansson and P. Johnson, “Assessment of enterprise information security - an architecture theory diagram definition”, In Proc. of the 3nd Annual Conf. on System Engineering Research, March 2005

[8] ISO/IEC, Common Criteria for Information Technology Security Evaluation, Common Criteria Project Sponsoring Organisations, ed. 2.2, 2004, available via http://www.commoncriteriaportal.org

[9] W. List, “The common criteria – good, bad or indifferent?”, Information Security Technical Report, v.2, no.1, pp. 19–23, 1997

[10] SSE-CMM (Systems Security Engineering Capability Maturity Model), Carnegie Mellon University, version 3.0, June 2003, http://www.sse-cmm.org/docs/ssecmmv3final.pdf,

[11] H. Koshutanski, F. Martinelli, P. Mori, L. Borz, and A. Vaccarelli, “A fine grained and x.509 based access control system for globus”, In OTM, pp. 1336–1350, Springer, 2006

[12] M. Colombo, F. Martinelli, P. Mori, M. Petrocchi, and A. Vaccarelli, “Fine grained access control with trust and reputation management for globus”, In OTM Conf., pp. 1505–1515, 2007

[13] H. Koshutanski, A. Lazouski, F. Martinelli, and P. Mori, “Enhancing grid security by fine-grained behavioral control and negotiation-based authorization”, Int. J. Inf. Sec., v.8, no.4, pp. 291–314, 2009

[14] J. Sabater and C. Sierra, “REGRET: A reputation model for gregarious societies”, In Proc. of the 4th Int. Workshop on Deception, Fraud and Trust in Agent Societies, in the 5th Int. Conf. on Autonomous Agents (AGENTS’01), pp. 61–69, Montreal, Canada, 2001

[15] L. Xiong and L. Liu, “PeerTrust: Supporting Reputation-based Trust for Peer-to-Peer Electronic Communities”, IEEE Transactions on Knowledge and Data Engineering (TKDE), vol. 16, no. 7, pp. 843–857, July 2004

[16] T. Huynh, N. Jennings, and N. Shadbolt, “FIRE: An integrated trust and reputation model for open multi-agent systems”, In Proc. of the 16th European Conf. on Artificial Intelligence, 2004

[17] R. Zhou and K. Hwang, “Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing”, IEEE Transactions on Parallel and Distributed Systems, vol. 18, no. 4, pp. 460–473, 2007

[18] G. Zacharia and P. Maes, “Trust Management Through Reputation Mechanisms”, Applied Artificial Intelligence J., vol. 14, pp. 881–907, 2000

[19] Z. Malik and A. Bouguettaya, “Trust Management for Service-Oriented Environments”, Springer, 1st ed., 2009

[20] M. Maximilien and M. Singh, “Agent-based trust model involving multiple qualities”, In Proc. of 4th Int. Autonomous Agents and Multi Agent Systems, 2005

[21] M. Maximilien and M. Singh, “Toward autonomic Web services trust and selection”, In Proc. of 2nd Int. Conf. on Service Oriented Computing, 2004

[22] W. Conner, A. Iyengar, T. Mikalsen, I. Rouvellou, and K. Nahrstedt, “A trust management framework for service-oriented environments”, In Int. World Wide Web Conf., 2009

[23] S. Kamvar, M. Schlosser, and H. Garcia-Molina, “The EigenTrust algorithm for reputation management in P2P networks”, In Proc. of the 12th World Wide Web Conf., Budapest, 2003

[24] L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank citation ranking: Bringing order to the web”, Technical Report, Computer System Laboratory, Stanford University, 1998

[25] A. Abdul-Rahman and S. Hailes, “A distributed trust model”, In Proc. New security Paradigms Workshop, Cumbria, United Kingdom, 1997

[26] T. Baumgartner, M. Heinrichs, A. Vonlanthen, U. Fischbacher, and E. Fehr, “Oxytocin shapes the neural circuitry of trust and trust adaptation in humans”, Neuron, vol. 58, pp. 639–650, 2008

[27] I. Bohnet and R. Zeckhauser, “Trust, risk and betrayal”, J. of Economic Behavior and Organization, vol. 55, pp. 467-484, 2004

[28] J. F. Bonnefon, D. Longin, and M. H. Nguyen, “A logical framework for trust-related emotions”, Electronic Communications of the EASST, 2009

[29] E. Bouwers, J. Visser, C. Lilienthal, and A. van Deursen. “A Cognitive Model for Software Architecture Complexity”, In Proc. of the 2010 IEEE 18th Int. Conf. on Program Comprehension (ICPC '10). IEEE Computer Society, Washington, DC, USA, pp.152-155, 2010

[30] C. Castelfranchi and R. Falcone, “Trust Theory: A Socio-cognitive and Computational Model”, Chichester, UK: John Wiley and Sons Ltd, 2010.

[31] E. Fehr and S. Gächter, “Altruistic punishment in humans”, Nature 415, pp.137–140, 2002

[32] E. Fehr and B. Rockenbach, “Detrimental effects of sanctions on human altruism”, Nature, vol. 433, pp.137–140, 2003

[33] V. Gallese and A.Goldman, “Mirror neurons and the simulation theory of mind-reading”, Trends in Cognitive Sciences, vol. 2, no. 12, pp. 493–501, 1998

[34] A. Giddens, “The Consequences of Modernity”, Stanford University Press, 1991

[35] W. Hasselbring and R. Reussner, “Toward Trustworthy Software Systems”, Computer, vol. 39, no. 4, pp. 91–92, 2006

[36] F. Heider and M. Simmel, “An experimental study of apparent behaviour”, The American J. of Psychology, vol. 57, no. 2, pp. 243-259, 1944

[37] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, “The EigenTrust Algorithm for Reputation Management in P2P Networks”, In Proc. of the 12th Int. World Wide Web Conf., 2003

[38] P. Kocher, “Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems”, In N. Koblitz, editor, Advances in Cryptology (CRYPTO '96), Santa Barbara, California, vol. 1109 of LNCS, pp. 104–113. Springer, 1996

[39] M. Kosfeld, M. Heinrichs, P. Zak, U. Fischbacher, and E. Fehr, “Oxytocin increases trust in humans”, Nature, vol. 435, pp. 673–676, 2005

[40] G. Möllering, “Trust: Reason, Routine, Reflexivity”, Oxford: Elsevier, 2006

[41] M. A. Nowak, “Five rules for the evolution of cooperation”, Science, vol. 314, no. 5805, pp. 1560-1563, 2006

[42] K. Oatley and P. N. Johnson-Laird, “Towards a cognitive theory of emotion”, Cognition and Emotion, vol. 1, pp. 29-50, 1987

[43] J. Riegelsberger, and M. A. Sasse, “Trustbuilders and trustbusters: The role of trust cues in interfaces to e-commerce applications”, 1st IFIP Conf. on e-commerce, e-business, e-government, pp. 17–30, 2001

[44] C. Tennie, U. Frith, and C. D. Frith, “Reputation management in the age of the world-wide web”, Trends in Cognitive Sciences, vol. 14, no. 11, pp. 482-488, 2010

[45] O. E. Williamson, “Calculativeness, Trust, and Economic Organization”, J. of Law and Economics, vol. 36, pp. 453-486, 1993

[46] B. Alcalde, et al., “Towards a decision model based on trust and security risk management”, in Proc. of the 7th Australasian Conf. on Information Security , Australian Computer Society, Wellington, New Zealand, vol. 98, pp. 61-70, 2009

[47] S. Jones and P. Morris, “TRUST-EC: Requirements for Trust and Confidence in E-Commerce”, Technical Report EUR 18749,1999

[48] B. Alcalde, and S. Mauw, “An algebra for trust dilution and trust fusion”, in 7th Workshop on Formal Aspects in Security and Trust, Springer-Verlag, 2009

[49] Wheeler, A. and L. Wheeler, Security Glossary, http://www.garlic.com/~lynn/secure.htm last accessed May 2011

[50] F. Martinelli, and M. Petrocchi. “On relating and Integrating Two Trust Management Frameworks”, In VODCA 2006, ENTCS vol. 168, pp. 191-205

[51] F. Martinelli, and M. Petrocchi. “A Uniform Framework for Security and Trust Modeling and Analysis with Cripto-CCS”, In ICS 2006, ENTCS vol. 186, pp. 85-99

[52] A. Jøsang, L. Gray, and M. Kinateder. Analysing topologies of transitive trust. In FAST, Tech. Rep.IIT TR-10/2003, pp. 9–22, 2003

[53] N. Li, W. H. Winsborough, and J. C. Mitchell, “Distributed credential chain discovery in trust management”, J. of Computer Security, vol. 1, no. 11, pp. 35–86, 2003

[54] F. Martinelli. Analysis of security protocols as open systems. Theoretical Computer Science, vol. 290, no. 1, pp. 1057–1106, 2003.

[55] E. Chang, T. S. Dillon, F. K. Hussain. “Trust and reputation relationships in service-oriented environments” 3rd Int. Conf. on Information Technology and Applications, vol. 1, pp. 4-14, 2005

[56] A. Jøsang; R. Ismail; C. Boyd. “A Survey of Trust and Reputation Systems for Online Service Provision”, Decision Support Systems vol. 43, no. 2, 2007

[57] A. Carroll, M. Juarez, J. Polk, T. Leininger, “Microsoft Palladium: A Business Overview”, Microsoft white paper, 2002

[58] J. McGibney, D. Botvich, “A Trust Overlay Architecture and Protocol for Enhanced Protection against Spam", In Proc. 2nd Int. Conf. on Availability, Reliability, and Security, Vienna, p. 749-756, 2007

[59] S. Hwang, H. Wang, J. Tang, J. Srivastava, “A probabilistic approach to modeling and estimating the QoS of web-services-based workflows”, Information Sciences, vol. 177, no. 23, 2007

[60] D. A. Norman and S. W. Draper, “User Centred System Design”, Lawrence Erlbaum Associated, 1986

[61] J. Riegelsberger and M. A. Sasse. “Ignore These At Your Peril: Ten principles for trust design”, In Proc. Trust 2010: 3rd Int. Conf. on Trust and Trustworthy Computing, 2010

[62] M. Ford, “Two modes of mental representation and problem solution in syllogistic reasoning”, Cognition, vol. 54, pp. 1-71, 1995

[63] A. Bundy, M. Jamnik, and A. Fugard, “What is a proof?” Philosophical Transactions of The Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 363, no. 1835, pp. 2377-2391, 2005

[64] D. Winterstein, “Using Diagrammatic Reasoning for Theorem proving in a Continuous Domain”, PhD Thesis, University of Edinburgh, 2004

[65] T. Erl, A. Karmarkar, P. Walmsley, H. Hass, L. U. Yalcinalp, K. Liu, D. Orchard, A. Tost, and J. Pasley, “Web service Contract Design & Versioning for SOA”, Prentice Hal, 2009.

[66] ISO/IEC 9126 Software engineering - Product quality standard http://www.iso.org/iso/catalogue_detail.htm?csnumber=22749

[67] P. Greci, F. Martinelli, I. Matteucci, “A framework for contract-policy matching based on symbolic simulations for securing mobile device application”, In ISoLA, pp. 221–236, 2008

[68] G. Costa, N. Dragoni, A. Lazouski, F. Martinelli, F. Massacci, I. Matteucci, “Extending security-by-contract with quantitative trust on mobile devices”, In Proceeding of 4th Int. Conf. on Complex, Intelligent and Software Intensive Systems, Krakow, Poland, pp. 872–877

[69] C. Dellarocas, “The Digitization of Word-of-Mouth: Promise and Challenges of Online Reputation Mechanism”, Management Science, vol. 49, no. 10, 2003

[70] A. D. Irvine, “Russell's Paradox”, The Stanford Encyclopedia of Philosophy, 2009, http://plato.stanford.edu/archives/sum2009/entries/russell-paradox/

[71] T. Coquand, “An analysis of Girard's paradox”, Proc. of the IEEE Symposium on Logic in Computer Science, pp. 227–236, 1986

[72] E. W. Dijkstra, “On anthropomorphism in science”, EWD936, 1985, http://www.cs.utexas.edu/users/EWD/transcriptions/EWD09xx/EWD936.html, last accessed July 2011

[73] L. C. Paulson, “Isabelle: The next 700 theorem provers”, In P. Odifreddi, editor, Logic and Computer Science, Academic Press, pp. 361-386, 1990

[74] Workflow Patterns initiative, http://www.workflowpatterns.com, last accessed July 2011

[75] S. Heckman and L. Williams, “A systematic literature review of actionable alert identification techniques for automated static code analysis”, Inf. Softw. Technol. vol. 53, no. 4, pp. 363-387, 2011

[76] K. Tsipenyuk, B. Chess, and G. McGraw, “Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors”, IEEE Security and Privacy vol. 3, no. 6, pp. 81-84, 2005

[77] M. Pistoia, S. Chandra, S. J. Fink, and E. Yahav, “A survey of static analysis methods for identifying security vulnerabilities in software systems”, IBM Systems J., vol. 46, no. 2, pp. 265-288, 2007

[78] Brian Chess and Jacob West, Secure Programming with Static Analysis, Addison-Wesley Professional, 1st ed., 2007

[79] R. G. Newcombe, “Interval Estimation for the Difference between Independent Proportions: Comparison of Eleven Methods”, Statistics in Medicine, vol. 17, pp. 873–890, 1998.

[80] K. E. Stanovich, R. F. West, and M. E. Toplak, “Individual differences as essential components of heuristics and biases research”, In K. Manktelow, D. Over, & S. Elqayam (Editors), The science of reason: A festschrift for Jonathan St. B. T. Evans, pp. 335-396, New York: Psychology Press, 2011

[81] W. Casscells, A. Schoenberger, and T. Graboys, “Interpretation by physicians of clinical laboratory results”, New England J. of Medicine, vol. 299, pp. 999-1000, 1978.

[82] U. Hoffrage, G. Gigerenzer, S. Krauss, and L. Martignon, “Representation facilitates reasoning: what natural frequencies are and what they are not”, Cognition J., vol. 84, pp. 343-352, 2002

[83] B. Fischhoff and W. B. D. Bruin, “Fifty–fifty = 50%?”, J. of Behavioral Decision Making, vol. 12, pp. 149–163, 1999.

[84] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins, “Propagation of trust and distrust”, In Proc. of the 13th Int. Conf. on World Wide Web (WWW '04). ACM, New York, NY, USA, pp. 403-412, 2004.

[85] B. Pan, H. Hembrooke, T. Joachims, L. Lorigo, G. Gay, and L. Granka, “In Google We Trust: Users’ Decisions on Rank, Position, and Relevance”, J. of Computer-Mediated Communication, vol. 12, pp. 801-823, 2005

[86] Z. Jianwu, L. Mingsheng, and L. Hui, "On achieving trustworthy SOA-based Web Services," In 2006 Int. Conf. on Security and Management, SAM'06, Las Vegas, NV, USA, pp. 341-347, 2006

[87] NATO Architecture Framework, Version 3, Annex 1 to AC/322-D(2007)0048

[88] A. Fugard, N. Pfeifer, and B. Mayerhofer, “Probabilistic theories of reasoning need pragmatics too: modulating relevance in uncertain conditionals”, J. of Pragmatics, vol. 43, pp. 2034-2042, 2011

[89] J. Henrich et al, “Economic man in cross-cultural perspective: Behavioral experiments in 15 small-scale societies”, Behavioral and Brain Sciences, vol. 28, pp. 795-855, 2005

[90] M. Mohammad and V. Alagar, “TADL - An Architecture Description Language for Trustworthy Component-Based Systems”, In Proc. of the 2nd European Conf. on Software Architecture, Springer-Verlag: Paphos, Cyprus. pp. 290-297, 2008

[91] Anderson, R., Security Engineering: A Guide to Building Dependable Distributed Systems. Wiley, 2nd ed., 2008