Modelling the trustworthiness of the IoT

Jorge Cuellar
Universität Passau
69 PUBLICATIONS 2,199 CITATIONS
SEE PROFILE

Ralf C. Staudemeyer
University of Applied Sciences Schmalkalden
34 PUBLICATIONS 414 CITATIONS
SEE PROFILE

Pavlos Charalampidis
Foundation for Research and Technology - Hellas
18 PUBLICATIONS 148 CITATIONS
SEE PROFILE

Alexandros G. Fragkiadakis
Foundation for Research and Technology - Hellas
78 PUBLICATIONS 1,466 CITATIONS
SEE PROFILE

Deliverable 3.3

Modelling the trustworthiness of the IoT

Editor:	Darío Ruiz, Atos
Deliverable nature:	Report(R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	29-02-2016
Actual delivery date:	30-09-2016
Suggested readers:	IoT application developers, Privacy researchers and Security experts
Version:	1.1
Total number of pages:	119
Keywords:	Trust, Reputation, Reacting Model

Abstract

End-users in Smart Cities and in general in any Internet of Things (IoT) application will encounter many different devices and services of which they have no prior information. They will need to decide whether or not they can trust these devices and services with their information. On the other hand, the owners and administrators of the applications will also have a strong interest in effectively detecting anomalous behaving devices and services in a cost-effective way.

The addition of Reputation Evaluation procedures can enrich IoT Systems through the definition of reactions to those evaluations and the integration with the authorization mechanisms. This way, it is possible to let service clients to decide whether to rely on services with low Reputation Evaluations or define other reactions such as logging the changes in the reputation or warning System Administrators of unreliable devices.

This document introduces a conceptual model for the IoT that allows defining the rules for evaluating Trust and Reputation, the reactions to this evaluation, and the integration of these reactions with the authorization mechanisms of the System.

After defining the conceptual model, the next step is applying this model to the concrete case of RERUM. This document introduces a possible design of how to apply the model to the RERUM architecture and is used as a basis for implementing the POC prototypes of Task 3.4.

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 609094

Impressum

Full project title	Reliable, resilient and secure IoT for smart city applications
Short project title	RERUM
Number and title of work-package	WP3: System & Information Security and Trust
Number and title of task	Task 3.4: Trust modelling
Document title	Modelling the trustworthiness of IoT
Editor: Name, company	Darío Ruiz López, ATOS
Work-package leader: Name, company	Ralf C. Staudemeyer, University of Passau
Estimation of person months (PMs) spent on the Deliverable	

History

Version 1.1	Supersedes version 1.0 of 1st April 2016. This version corrects the restricted dissemination level and declares the document to be public.

Executive summary

There is a pressing need for a simple, yet effective, trust management system for Smart Cities or, more generally, Internet of Things (IoT) applications. Both end users as well as the application owners will be confronted with devices that are malfunctioning, for one reason or the other. Existing trust management systems are either centralized, or are designed for a narrow set of applications, rely on rather complex behavioural models, require to keep a large amount of information that must be analyzed, or contradict basic privacy principles (like to aggregate the data as close to the data collection points, i.e. the devices, as possible).

This document presents a new, efficient, and flexible Trust Model for the IoT and applies it to the RERUM framework. The Trust Model is split into two parts, namely a Trust Evaluation Model and a Reacting Model. The first provides a composite Reputation Evaluation, and the latter defines a set of possible reactions with a generic mechanism to execute them.

The document starts defining the requirements for a generic Trust Model applicable for the IoT and makes an analysis of the most relevant trust approaches, including:

- Social Trust Management,
- Network Trust Management, and
- Plausibility Verification.

This Deliverable focuses on providing a Trust Model for RERUM using the last two.

We are proposing a novel trust evaluation method that is simple enough to be processed close to the data source (as privacy guidelines recommend), even by constrained devices, but strong enough to detect many types of anomalous behaviours. More explicitly, we propose a model (see Section 2.2.4 for more details) that

1. allows for an easy calibration, even by non-experts, to increase the alarm sensitivity or to reduce the number of false positives;
2. does not require large amounts of data being stored in memory;
3. allows the analysis close to the data collection points;
4. is usable for most types of sensors and physical variables (temperature, humidity, light intensity, etc);
5. does not require a complex behavioural model;
6. does not rely on mining large amounts of data;
7. allows the filtering of data streams by eliminating anomalous values;
8. runs with high performance;
9. is effective to detect a large class of anomalous behaviours;
10. is adaptive to the changing situations; and
11. is flexible, allowing the integration of more complex behavioural or data-fusion models.

To achieve this, we extend well-established methods for calculating adaptive mean estimators (using exponential smoothing) to more interesting parametric and non-parametric statistics like the standard deviation, quantiles, characteristic “jumps”, etc. In addition, we use them to detect abnormal sensor or network behaviour on the fly. This forms the basis of our proposed solution.

The Trust Evaluation Model consists of two main steps: The first step consists in several observers providing a trust rating for each incoming data stream according to a set of evaluation rules expressed as an expert system in the language CLIPS (C Language Integrated Production System). For example we can calculate a trust rating for single data values using signatures. In RERUM we assume that Smart Objects can generate signed data values. Based on the verification outcome, an observer can estimate a corresponding trust rating.

The second step provides a composite Reputation Evaluation that takes into account the different trust ratings, and creates a joint evaluation. This joint evaluation is not a single value, but a set of different evaluation values according to several evaluation criteria defined in CLIPS rules.

We have considered the following options for a Reacting Model:

- Logging alerts,
- Warning administrators,
- Disabling services, and
- Warning requesters on low Reputation Evaluations.

Technically, the reactions associated to changes in the Reputation Evaluations can be implemented for instance by:

- Manually Defined Reactions,
- XML Based Event Condition Action (ECA) languages, and
- Database based ECA languages.

The proposed mechanism for executing the reactions is based on the definition of an interface that receives as input the Reputation Evaluation that was provided by the Trust Evaluation Model or any refined event derived from it. This mechanism is able to execute any action defined by developers for the system as long as they comply with this interface.

The design of the integration of the model with RERUM is carried out following the flow of the information involved in the evaluation of the reputation, that is:

- Interception of RERUM Data,
- Collection of data from external services,
- Managing Context Data,
- Processing of Reputation,
- Correlating Reactions to Reputation Alerts,
- Processing Reactions, and
- Incorporating the Reputation Access Profiles to the Authorization Process.

Additionally, the document shows how the reputation of the user can feed the authorization of the users to improve it by following the mechanism of ‘chain of filters’ already defined in Java Enterprise Edition JEE. In short, a Proof of Concept (POC) evaluator for the reputation of the users puts that information in an HTTP header that is later available for the authorization components, which can make use of them in the access policies.

For the trust Reputation Evaluation at the network level, we provide the example of the design of a hierarchical multi-layer framework for evaluating the trust-ratings of devices in an IoT network. Several parameters are considered for the trust-ratings of devices and the goal is to identify misbehaving or malicious devices against a number of attacks and faults. For example, we consider the *packet drop rate* and the *packet modification rate*, for computing the misbehaviour rate of each device, which is a type of trust rating. This rate is computed by each node for all of its one-hop neighbours, so one report (per node, per neighbour) is created, and transmitted to the Reputation Manager in order to identify the reputation of the device according to pre-defined rules. After all reports are collected, the Reputation Manager fuses them using the D-S algorithm, to estimate the probability that supports the malicious nature of a node. We improve the performance of D-S, in terms of its conflict, by applying an outlier detection method before fusion, so conflict substantially decreases.

List of authors

Company	Author	Contribution
ATOS	Darío Ruiz López	Overall Editor Contributed to sections: • Executive Summary • 1.1 Objectives of this document • 1.3 Structure of this document • 2.1 Trust Model Requirements • 2.3 Overall structure of the trust model • 3.5 Processing of Reputation Main contributor to sections: • 2.5 Reacting Model • 3.1 Overall integration of Trust Evaluation Model in RERUM Architecture • 3.2 Interception of RERUM data • 3.3 Interception of external data • 3.4 Managing context data • 3.6 Correlating reactions to data events • 3.7 Processing of reactions • 3.8 Incorporating the Reputation Access Profiles to the Authorization Process • 3.9 Processing Reputation for Users at Service Level
SIEMENS	Jorge Cuellar, Ricarda Weber, Prabhakaran Kasinathan, Santiago Suppan	Contributed to sections: • Executive Summary • 1.1 Objectives of this document • 1.3 Structure of this document • 2.1 Trust Model Requirements • 2.3 Overall structure of the trust model Main contributor to sections: • 2.2 Analysis of the possible trust approaches for IoT • 2.3 Overall structure of the Trust model • 2.4 Trust Evaluation Model (except 2.4.4 Trust for single data values) • 3.5.Processing of reputation

UNI PASSAU	Ralf C. Staudemeyer, Henrich C. Pöhls	Contributed to sections: • Executive Summary • 1.1 Objectives of this document • 1.2 Structure of this document • 1.3 Position within the Project • 2.3 Overall structure of the Trust model • 2.4 Trust Evaluation Model (integrity) • 3.5 Processing of Reputation • 4.0 Conclusions Main contributor to section: • 2.4.4 Trust for single data values
FORTH	Alexandros Fragkiadakis, Pavlos Charalampidis, Elias Tragos	Contribution to Section 3.10: Trust Reputation Evaluation for Devices at the network level

Table of Contents

List of Figures

List of Tables

Abbreviations

CA: Certification Authority

CDF: Cumulative Density Function

CLIPS: C List Processing: Programming Language similar to LISP

COAP: Constrained Application Protocol

CPU: Central Processing Unit

DARE: EU financed research project

DBMS: Database Management System

DFMT: Data Fusion and Mining Trust

DOLCE: Declarative programming language used in the RERUM component Stream Processor

DoS: Deny of Service attack

DOW: Description of Work document

DPT: Data Perception Trust

D-S: Dempster-Shafer

DTCT: Data Transmission and Communication Trust

ECA: Event-Condition-Action

ECA-DL: Event-Condition-Action Definition Language

ECDSA: Elliptic Curve Digital Signature Algorithm

ENISA: European Union Agency for Network and Information Security

EPL: Event Processing Language

ETX: Expected Transmission count

G: Generality

GUI: Graphical User Interface

GVO: Generic Virtual RERUM Object

GW: Gateway

HVTI: Human Trust Interaction

HW: Hardware

IEEE: Institute of Electrical and Electronics Engineers

IoT-A: Internet of Things . Architecture Project

IT: Identity trust

JEE: Java Enterprise Edition

JESS: Rule Engine for the Java Environment

JSON: Java Script Object Notation

JVM: Java Virtual Machine

KNN: k-th Nearest Neighbour (algorithm)

LAN: Local Area Network

MBR: MisBehavior Rate

MW: MiddleWare

OSTM: Overall Service Based Trust Metric

PDR: Packet Drop Rate

PEP: Policy Enforcement Point

PIP: Policy Information Point

PKI: Private Key Infrastructure

PMR: Packet Modification Rate

POC: Proof Of Concept

PP: Privacy preservation

PRP: Policy Retrieval Point

QIoTS: Quality of IoT services

QoS: Quality of Service

RPC: Remote Procedure Call

SOA: Service Oriented Architecture

SP: Service Provider

SSR: System Security and Robustness

SQL: Structured Query Language

STM: Service Trust Metric

SMC: Secure Multiparty Computation

TCG: Trusted Computing Group

TPM: Trusted Platform Module

TRA: Trust and Reputation Architecture

TRD: Trust Relationship and Decision

VANET: Vehicular Ad-Hoc Networks

VE: Virtual Entity

WSN: Wireless Sensor Networks

XACML: eXtensible Access Control Markup Language

XLST EXtensible Stylesheet Language

XML: eXtensible Markup Language

XSL: Extensible Stylesheet Language

XSLT: Extensible Stylesheet Language Transformations

Definitions

In the literature, there is no consensus regarding the terms related to trust. We use the following terminology, but the user is advised that the terms are used in a different way by different authors. In particular, the term “trustworthy” is used to describe an intrinsic, non-verifiable property of an entity, that is, the ability, benevolence, and integrity of a trustee.

Risk: Risk is the possibility of losing some value or asset, or of missing a goal, if an uncertain unwanted event happens. Risk has two different dimensions: the likelihood (or *probability)* of something bad happening and the resulting impact (or *cost*) the event has in terms of losses. In formulas the risk value is often the product of them: likelihood times impact. In a multi-party environment (like IoT) one single unwanted event may imply different risk values for different parties, depending on the cost that the event has for each of them.

Trustworthiness: An entity or service is *trustworthy* if there is absolutely no risk in using this service, or engaging in transactions with this entity, or relying on the values provided by the service. Notice that a service is trustworthy in a certain point in time if in the future it will definitely behave according to the expectations of the system and the other (non-malicious) entities of the system. Expectations include in particular adherence to service level agreements, protocol specifications, and privacy and security policies of the system.

Trust-rating is a preliminary estimator of the trustworthiness of a service, data or entity. It describes the degree of trustworthiness according to the criteria of a particular “observer”*,* which has directly monitored a service, a data stream or an entity, based on some rules to detect abnormal behaviour of that entity. Administrators and other privileged users may also enter trust-ratings, based on internal or external information (like vulnerability bulletins, etc). One service may have different observers and therefore also different trust-ratings, for different reasons. Those *reasons* are expressed as conditions for the trust-rating rules.

Reputation is an aggregated estimator of trustworthiness, based on the trust ratings of several observers, and therefore based on different criteria. Note that the values "trust-rating" and "reputation" are related to the probability of a risk, but not necessarily directly related to the amount of risk itself, as the risk also depends on the impact (*cost)* of an event for a given party (reputation of an entity is independent of the *cost* of risk of trusting the entity).

Trust-value (or "trust"): A party has to rely on services, data, or other parties or entities, according to his context and purposes and in order to complete his goals. If he has some choices here, to use one service or another one, his decision may depend on economic calculations, but also on his risk perception of the choice implications. Also in cases where he has no choices, he may decide to add additional risk-control measures or countermeasures in case of a problem. In any case, his *trust* on a service is a measure of the confidence he has, that the risk of using this service is acceptable.

Trustor: A party who trusts a service, data, or other parties or entities in order to complete his goals.

Trustee: The party whom the trustor relies, or the party that the trustor wants a trust evaluation of, in order to know if he can trust it.

Trust Model: The set of rules and processes used by a system and the interested entities to answer the trust question: "should entity A trust entity B for a given purpose"? The trust model is composed of mainly three parts: (1) the *processing model*, which describes how the rules are evaluated, based on which events or information, how the rules are written and by whom; (2) the *rule language*, which describes which *generic rules* can be expressed, and (3) the *concrete rules*, used by a particular system in a concrete situation to evaluate a trust question or to react to a given trust/reputation state.

1 Introduction

In general, any IoT system will encounter the difficulty that some sensors or devices may be malfunctioning and providing erroneous measurement values. Moreover, compromised or corrupted services may also provide incorrect responses to the users of the service. It is therefore prudent to provide a reputation value for values or services, so that consumers are able to choose, if there is some redundancy in the data or services, or take further reactions or precautions. This is especially interesting in any IoT environment, because the consumer services will possibly know very little about the service invoked (in a non-IoT system the services and the applications consuming them often belong the same service, or know each other, or have long-lasting trust relationships). For this reason, it is important that the IoT systems are able to assess the reputation of their services and to take reactions based on that assessment, such as disabling non-trusted devices or warning about low Reputation Evaluations.

1.1 Objective of this Document

The main objective of this document is to produce a conceptual Trust Model for the IoT that allows the definition of mechanisms for assessing the reputation of provided services and the definition of corresponding reactions to be taken as a consequence of that assessment, including the warning of applications consuming services of low reputation.

The work carried out in Task 3.4 has three parts:

1) Development of a model for the trustworthiness of information exchanged in the Internet of Things.
2) Integration design of the Trust Model within RERUM.
3) Implementation of a basic proof of concept (POC) prototypes for:
    a) Trust Evaluation Model,
    b) Basic Reaction to the results of the Model, and
    c) Incorporation of the Reputation Evaluation of Users to the Authorization Process.

This document also demonstrates how the reputation of the users could be added to RERUM to enrich the authorization process. However, this prototype is conceived for demonstrating how to integrate a user reputation with the authorization instead of how to calculate that reputation, which is out of the scope of the task. For that reason the evaluator for the user reputation is a trivial one. The description of this POC is presented in Chapter 3.9, too.

1.2 Structure of this Document

Chapter 2 explains the Trust Model as proposed by RERUM for the IoT. This model is the result of the research carried out in Task 3.4. We first define the generic IoT-requirements of the trust model. Later, we explore different trust approaches and analyse the possible alternatives to our model. In Section 2.3 we present an overview and a brief introduction of the different types of components of the trust model. In the then following two sections we cover the two parts of our Trust Model: the Trust Evaluation Model and the corresponding Reacting Model. The first explains how the model analyses the data to produce the trust and Reputation Evaluations and the latter explains possible options to react to the Reputation Evaluation and incorporate it to the authorization mechanisms.

Then, Chapter 3 explains how the invented Trust Model can be applied to RERUM. We start with covering the overall integration of Trust Evaluation Model in the RERUM Architecture. Here we provide an overview of the trust, reputation and reactor components and explain how they fit into the RERUM architecture. The subsequent Sections 3.2 to 3.8 follow the processing of the data. Section 3.2 explains how data is intercepted and Section 3.3 covers how data from external services is obtained and used to feed the components of the Trust Evaluation Model. Then, Section 3.4 explains how components implementing the Trust Model can access data from other RERUM components. In Section 3.5 we explain how the components that process the Trust Evaluation Model work. Sections 3.6 and 3.7 cover the correlation and processing of reactions as defined in the Reacting Model. The incorporation of Reputation Access Profiles to the Authorization Process is covered in Section 3.8. The next Section 3.9 explains how the reputation of users can be incorporated to the authorization process. The final Section 3.10 goes into details on evaluating the reputation of devices at network level.

1.3 Position within the Project

In this section, we set the results of T3.4 in the RERUM context, highlight the innovations, and explain the relation to the RERUM use cases.

1.3.1 Relation with other Tasks and WPs

This deliverable D3.3 is the result of Task 3.4, which is part of Work Package 3. Inputs are D2.5 providing the reference architecture of RERUM and D3.1 contributing the authorization components of RERUM. The reference architecture is used when explaining how to integrate the proposed IoT model in RERUM. The D3.1 document is used as a reference of how RERUM authorization components work.

As an output, this deliverable complements D3.1 because it upgrades the authorization components with the ability of taking the results of the Reputation Evaluation of the services into account. However, the most important output of this deliverable is for the scientific community. It is the Conceptual Trust Model for the IoT, whose research innovations are described in the following.

1.3.2 Innovations in the Document

The definition of a trust model is not innovative per se. There is plenty of previous work about how to calculate trust ratings and reputations; and as well the integration with a reaction model is not innovative per se. Existing trust management systems are either simplistic, or designed for a particular application. They may depend on rather complex and expensive behavioural models, require large amounts of information that must be kept and analysed, or contradict basic privacy principles (for example to aggregate the data as close to the data collection points, i.e. the devices, as possible).

This document presents a new, efficient, and flexible Trust Model for the IoT and applies it to the RERUM framework. The Trust Model is split into two parts: a Trust Evaluation Model, which provides a composite Reputation Evaluation, and a Reacting Model, which describes how the System reacts to that Reputation Evaluation.

To our better knowledge, the innovations in this document are:

- The definition of a Trust Model specific to the IoT.
- A flexible, general-purpose, effective, high-performance, privacy preserving, on-the-fly ("streaming mode") Trust Evaluation Model with 11 advantages:
  - allows for an easy calibration, even by non-experts, to increase the alarm sensitivity or to reduce the number of false positives;
  - does not require large amounts of data being stored in memory;
  - allows the analysis close to the data collection points;
  - is usable for most types of sensors and physical variables (temperature, humidity, light intensity, etc);
  - does not require a complex behavioural model;
  - does not rely on mining large amounts of data;
  - allows the filtering of data streams by eliminating anomalous values;
  - runs with high performance;
  - is effective to detect a large class of anomalous behaviours;
  - is adaptive to the changing situations and
  - is flexible, allowing the integration of more complex behavioural or data-fusion models.
- Definition of a set of generic reactions that are applicable to IoT.
- Integration of the evaluation of the reputation with the authorization process.
- Ability to calculate trust metrics and the reputation of an IoT device based on statistics both for the network and data reliability.

1.3.3 Relation with the Use Cases

The trust model depicted in Chapter 2 provides a way for IoT systems such as RERUM to adapt to the level of trustworthiness of the services via the definition of a series of distinct reacting rules. This adaptation includes both disabling untrusted services and warning about low Reputation Evaluations. Thus, the trust model introduced in this document foster Contribution 10: Integration of ABAC in IoT with specific business data contained in the request.

Chapter 3 provides the design for integrating the model in RERUM. Additionally, it provides a design for Contribution 9: Enrich authorization process with Reputation Evaluation and Contribution 22: 6LoWPAN Multicast.

As the following figure shows, Task 3.4 received input from the system architecture and D3.2. More specifically, it took into account the Authorization mechanisms that were initially described in D3.1 and further refined in D3.2. With this, it also provides a POC prototype to be used in the pilots.

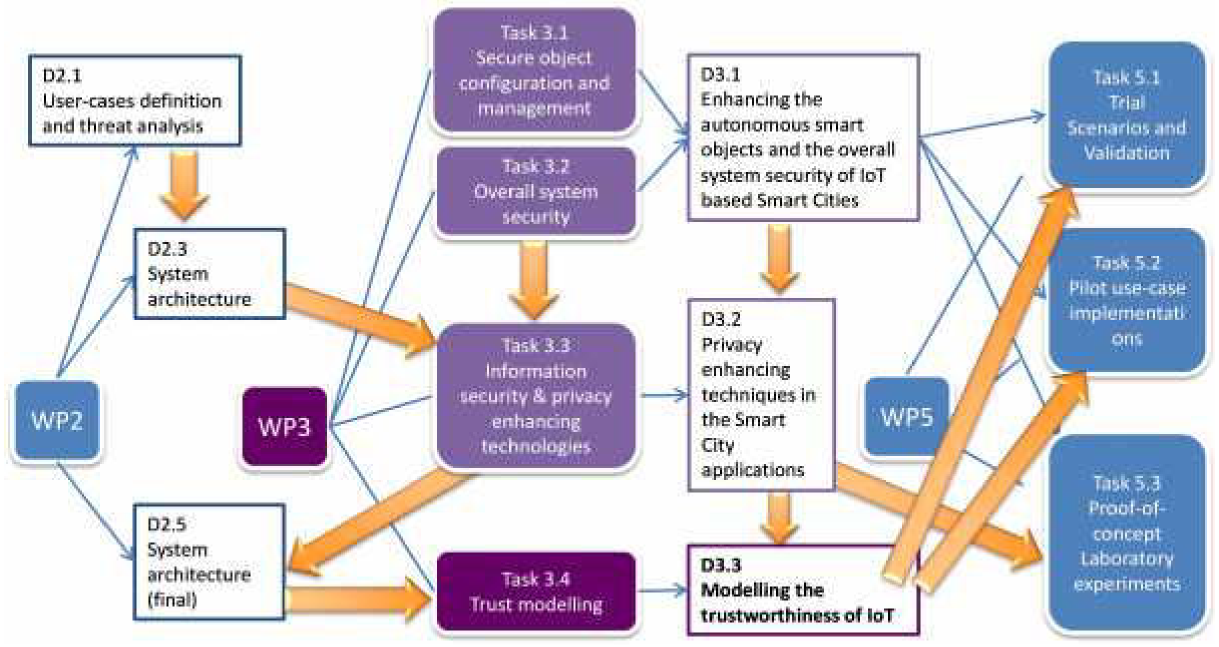

D2.1
User-cases definition and threat analysis
D2.3
System architecture
WP2
D2.5
System architecture (final)
Task 3.1
Secure object configuration and management
Task 3.2
Overall system security
Task 3.3
Information security & privacy enhancing technologies
WP3
Task 3.4
Trust modelling
D3.1
Enhancing the autonomous smart objects and the overall system security of IoT based Smart Cities
D3.2
Privacy enhancing techniques in the Smart City applications
D3.3
Modelling the trustworthiness of IoT
WP5
Task 5.1
Trial Scenarios and Validation
Task 5.2
Pilot use-case implementations
Task 5.3
Proof-of-concept Laboratory experiments

Figure 1: Position of T3.4 / D3.3 within RERUM

1.4 Trust Management Topics in IoT

According to [IOT-A D4.2], information about entities should be used to estimate the trustworthiness of an entity (either on an absolute scale or relative to other entities). *Trustworthiness* is a rather abstract construct: it relates to the degree to which an entity (in our case particularly a device or a service) is acting "correctly", honestly and reliably, so that there is no risk in trusting it. For any observer it is impossible to decide with certainty if an entity is trustworthy, that is, if this entity *will act correctly in the future*: an entity may behave “correctly”, without creating any suspicion in order to gain good reputation and then mount a complex attack. Although trustworthiness can't be directly measured, it is necessary to obtain estimates. According to [Zheng 14], the main building blocks are:

- **Trust (-value)** relates to **confidence** of a user (data stream consumer) in the reliability, integrity, security, dependability, ability, and other characteristics of an entity. The factors to be taken into account here include not only the past behaviour of the party to be trusted, but essentially also the needs of the party that must decide on granting trust and possibly other factors.
- **Reputation** is a measure derived from direct or indirect knowledge or experiences on earlier interactions of entities (“multiple observers”) and is used to assess the level of trust one may be willing to put into an entity.
- **Trust metrics** (or as we call them "*trust-ratings*") allow deriving and computing a trust value of a given IoT component based on selected indicators. A trust-value then is a kind of trustor-personalised trust rating.
- **Observers** (or witnesses) look out for such trust indicators and produce trust-ratings. Observers may be parts of services or separate processes.

[Yan 08] and also [Yan 11] differentiate five factors influencing trust, based on the properties of the trustee (the party to be trusted), the trustor (the party that deliberates whether to have confidence in the trustee) and their context:

- **Trustee's objective properties:** Competence; ability; security (confidentiality, integrity, availability); dependability (reliability, maintainability, usability, safety); predictability; timeliness; (observed) behaviours; strength; privacy preservation.
- **Trustee's subjective properties:** Honesty; benevolence; goodness.
- **Trustor's objective properties:** Assessment; a given set of standards; trustor's standards.
- **Trustor's subjective properties:** Confidence; (subjective) expectations; subjective probability; willingness; belief; disposition; attitude; feeling; intention; faith; hope; trustor's dependence and reliance.
- **Context (incl. purpose):** Situations entailing risk; structural risk; domain of action; environment (time, place, involved persons), purpose of trust.

Most trust/reputation models generally follow five generic steps as discussed in [Marti 06]. The context is the following: a user or service, the trustor, needs to choose one entity to perform a certain transaction, so he wants to see which candidate partner (trustee) deserves more trust. [IOT-A D4.2] recommends that any standard model should be designed having these 5 steps or components in mind.

1. **Observation:** Gathering behavioural information about the system's entities: The first component of a reputation system is responsible for collecting information on the behaviour of nodes, which will be used to determine how "trustworthy" they are. Using established network identities, a reputation system protocol collects information about the behaviour of a given node in previous transactions. This information is collected individually from each node querying them, or is provided proactively by all nodes collating together their experiences. Information sources may be for instance own personal experience, experience of pre-trusted nodes, directly or transitively, or a global reputation system.
2. **Scoring:** Scoring and ranking entities: (sensors, or even their measurements, in the case of IoT-A): Once the nodes' past history has been collected and properly weighted, a reputation score is computed for each node, either by an interested agent, a centralized entity, or by many nodes collectively.
3. **Selection**: Once an agent has computed reputation ratings for the nodes interested transacting with it, it must decide which, if any, to choose. This is especially true if multiple peers offer the same service, but also if the services are ranked closely, based on whether the peer's reputation rating is above or below a set selection threshold.
4. **Transaction**: When a service has been chosen, out of the possible candidates, the trustor performs the transaction with (or uses the service offered by) the chosen trustee.
5. **Rewarding** and **punishing** entities: Reputation systems can be used to motivate peers to positively contribute to the network and/or punish adversaries who try to disrupt the system.

Finally, the trustor, after performing his transaction with his chosen service, should assess his experience, producing a new .observation (called feedback).

As [IOT-A D4.2] points out, redemption of past malicious entities who have improved should be possible in a reputation system, and a utility score may complement the reputation score, also to give newcomers without a reputation score an opportunity to participate in the system.

[Zheng 14] considers that a holistic trust management system should evaluate many different aspects, including data perception trust (DPT), identity trust (IT), privacy preservation (PP), Data transmission and communication trust (DTCT), System security and robustness (SSR) trust, privacy-preserving data fusion and mining trust (DFMT):, trust of users (UT), and the trust of IoT applications (or Quality of IoT services, QIoTS).

RERUM has already addressed security, robustness and privacy aspects in separate deliverables ([RD3.1] and [RD3.2]). Therefore in the context of this trust-focused deliverable these topics are not stressed. Thus, security-centric properties like data transmission and communication trust (DTCT), privacy preservation (PP), System security and robustness (SSR), and Identity trust (IT) are regarded as out of scope for this deliverable. This includes the trust one is to put into the credentials of a party, like for instance managed by a public key infrastructure (PKI). The specific RERUM use case ”smart cities” needs to be dealt with, thus defining a generic trust management system and thus achieving generality is not central aim of RERUM.

Services offered within the RERUM smart cities context heavily rely on the sensed data of the environment, traffic, movement, etc. Their reliability and trustworthiness is essential. Therefore RERUM focuses on data perception trust (DPT) and the correctness and plausibility of sensed data and the corresponding data streams. Privacy and security aspects of data fusion and mining are regarded as out of scope in this deliverable, (but related work is included in the RERUM security and privacy deliverables).

In the course of addressing data perception trust and data fusion and mining trust (so far as the resulting content data streams of those fusion and mining processes are concerned), RERUM also needs to deal with the corresponding vertical topic of trust relationship evaluation and trust decision making (TRD). When evaluating trust of data streams, RERUM takes into account the quality of service criteria raised by the trustors as well as their context to help them meet their personalised IoT quality of service (QIoTS) needs and purpose of data use.

In this deliverable we discuss the specification of a trust-provisioning system, but not the *securing* of trust management system, which is a large topic of its own. Attacks on a trust management system may involve false recommendations by malicious nodes. The trust-computing protocol should be resilient against such false recommendations, which the system requires robustness against. Attacks against the trust management system could for instance be:

- Falsification of ratings, leading to Denial-of-Service for unjustified poor reputation, or to abuse maliciously acquired good reputation [Marti 06]))
- (Distributed) Denial-of-service, not only against the availability of the reputation system, but also for collusive good- and bad- false reputation building (see also [IOT-A D4.2])
- Collusion or Sybil attacks (see [13]): an adversary initiates a disproportionate number of malicious peers in the network. Each time one of the peers is selected as a service provider, it provides a bad service, after which it is disconnected and replaced with a new peer identity)
- Abuse of a well achieved reputation (getting a very good reputation and then abusing it; see also [IOT-A D4.2])
- Abuse of newcomer status (misbehaving, leaving the system and restarting as newcomer, whitewashers, as they are called in [Marti 06]) [IOT-A D4.2]
- Abuse of a good reputation in transactions of minor relevance for transactions of major relevance. Impact on reputation should correspond to the relevance of the transaction [IOT-A D4.2]
- Hijacking and abuse of orphaned accounts (accounts that are not used actively any more) of good reputation.

For instance, [Saied 13] presents a context-aware trust management system for the IoT that assigns dynamic trust scores to cooperating nodes according to different contexts (e.g. status of the assisting node) and different functions (e.g. for which service the assistance of that node was required). Special attention is paid on how much confidence should be put in a report provider. A quality of recommendation score is assigned to each node reflecting its trustworthiness when rating other nodes and adjusted after each interaction during the learning phase, so the trust management system becomes resilient to targeted attacks.

2 Trust model

This chapter is about defining a model that evaluates the trust and reputation on data, services and devices. For this reason, it contains Section 2.4 devoted specifically to the mechanisms to perform such an evaluation. This section studies the data used in IoT from both a reliability and integrity perspective.

From a reliability perspective, the Trust Evaluation Model takes into account the ability to analyse the data from the point of view of several observers, each of them being able to produce their own trust ratings according to their own criteria and for different purposes. These trust ratings can be generated for evaluating trust at data stream, device and service level. The trust ratings from individual observers are then fused together to produce a combined Reputation Evaluation (also at data stream, device and service level) for several evaluation criteria, too. The evaluation criteria for producing the Reputation Evaluation should take into account the specific reasons that lead to different trust ratings.

From an integrity perspective, the signature associated to the data is also checked when evaluating the trust rating by each observer. This contributes to further refine the trust of the generated data and the producing services and devices. (Correctly signed values can be untrusted because a sensor is malfunctioning, but incorrectly signed values should have a low trust rating. The same is true for values that according to the security policies *should* be signed but are not).

The set of rules of each observer for evaluating the trust rating for each purpose at the different levels comprise the Trust Evaluation Model. (Even so, there are going to be data consumers that may have a new purpose for using existing data streams not taken into account by the observers). In that case they have to add a new observer for their needs). The set of rules for combining the different trust ratings in a joint Reputation Evaluation comprise the reputation model. Together, the Trust Evaluation Model and the Reputation Model, make up the Trust Evaluation Model.

Besides analysing the reliability and integrity of the data, when it comes to incorporate a trust model to anything, including an IoT environment, it is important to take a step back to have a holistic view of the system and take into consideration why it is necessary to evaluate trust and reputation and what this evaluation will be used for. In other words, once the reputation value is calculated, how the system could react to that evaluation and how it is defined? For that reason, Section 2.5 is devoted to explain the mechanisms for defining and deciding such reactions.

For this reason, the Reacting Model section will define mechanisms for creating rules that allow the specification of access policies which support distributed heterogeneous access control and access profiles. These access policies will be based on the type of sensor, their role in the application and the reputation indicator. Note this support will usually be oriented to *providing advice for accessing services* more than for disabling them. This way, the Reputation Evaluation (indicator) can be used at the moment of the authorization to allow accessing only trusted services, enabling the easy way to share information between platforms that also improves its robustness by taking into account the Reputation Evaluation.

This model is aimed not only for RERUM but also for the whole IoT domain, where the possible actions to be taken can be created dynamically. For that reason the Reacting Model additionally supports the definition of generic actions, even updating the reputation of the services and their associated devices.

Note the dynamic and distributed nature of IoT may need the ability to include logic to correlate past evaluations from different sources or systems to produce advanced reactions or to perform the mentioned update of the reputation. For instance, a single bad reputation rating for data may not be sufficient to disable a given service, but 10 bad reputation ratings in a row or 50 out of 1000 could be enough to do it. That is why the ability of the Reacting Model to alter the reputation of the objects should not be considered as an alternative to the Trust Evaluation Model but as a complement for it where a correlation between Reputation Alerts is needed.

2.1 Trust model requirements

This section presents the requirements that the Trust model must comply with. The requirements are the final result of an iterating process from the beginning of Task 3.4 till the final writing of the deliverable following the philosophy that, as a conceptual model, the model should be oriented to the whole IoT domain and not only to RERUM. For this reason, it should take into account the whole IoT needs instead of only RERUM requirements.

All the requirements have an identifier to allow tracing their compliance in each section and at the end of the document. The name of the requirement complies with the following nomenclature:

<scope><number of requirement> where <scope> is the scope of the requirement in the model, being

- TM for the whole Trust Model;
- AM for the Analysis Model;
- RM for the Reacting Model

and <number of requirement> is simply a number increasing for each requirement. Though it would be enough to increase this number per each section to produce a unique identifier, we have preferred to increase it per requirement to avoid confusions between scopes.

2.1.1 Requirement TM1: Efficient use of Hardware Resources

Many IoT environments have strong limitations in terms of hardware resources and access times, which make difficult for them to execute even a basic trust system. For this reason, the model exposed must avoid those design options that would force redundant operations or unnecessary access to the devices with less hardware resources.

2.1.2 Requirement AM2: Support Evaluation of Reputation at Service Level

The model must be able to calculate the evaluation of the reputation of their services exposed so this evaluation can be used later for the procedures of the system, including the authorization ones.

2.1.3 Requirement AM3: Support Querying Trust at Service Level

The model must be able to be queried for an evaluation of trust of their services exposed so an external client can decide whether to use that service or not.

2.1.4 Requirement AM4: Support Querying Trust at Device Level

The model must be able to be queried for an evaluation of trust of the devices involved in their services so an external client can decide whether to use that service or not.

2.1.5 Requirement AM5: Support Evaluation of Trust at Device Level

The model must be able to be queried for an evaluation of trust of the devices involved in their services exposed so an external client can decide whether to use that service or not.

2.1.6 Requirement AM6: Support Reputation for multiple Evaluation Criteria

The evaluation of the reputation for devices and services and data must be performed based on different criteria (with different purposes in mind), as required by the entities of the system, resulting in a set of reputation values for each device, service and datum produced instead of a single value for each of them.

For the sake of clarity, let us call "Reputation Manager" the process or entity creating the reputation values and "reputation consumer" the entity or process that uses those values (for instance for access control or reaction). The requirement can be expressed as follows: there is a list of "Reputation Evaluation criteria" for which the Reputation Manager will create reputation values for each service to be assessed.

This list can be dynamically extended, but most probably not automatically, with human intervention and with an agreement between the designers of the Reputation Manager and the reputation consumers regarding the semantics (interpretation or meaning and background model parameters used for the calculation) of the different reputation purposes. That is, this list is subject to change over time, but only if agreed between the designers of the Reputation Manager and the reputation consumers.

A set of standard examples of reputation purposes is given explicitly in the Section 2.4.9. The interpretation of the CLIPS rules can be as follows:

- In terms of accuracy/precision, a high reputation (say, 9 in a scale 0 to 10) corresponds to a high percentage of stream values (say 95% of the stream values) to have a small imprecision, that is a small deviation from the correct value (say, within an interval of 1% of the correct value).
- In terms of timeliness, a high reputation (say, 9 in a scale 0 to 10) corresponds to a maximum delay of a small amount of time (say, 0.5 seconds) and to a relatively small amount of loss of values (say, less than 10%).

The interpretation of the rules is dependent on the model of the process assumed and on the specific calculations of the rules.

2.1.7 Requirement AM7: Support for multiple Observers

The model must support different points of view when evaluating the trust ratings of services and data. The Reputation Evaluation must be able to take into account all this different point of views together. If a service or user notes that there is no observer that is rating the service or data stream as he would need it, there must be, at least in principle, a possibility for the user or service to create an observer with his own rules or parameters. This may be subject to some conditions: the amount of observers and the required effort and overhead, the privileges of the user or service, etc.

2.1.8 Requirement AM8: Produce Alert only for Relevant Changes

The model must be able to produce an alert in the following situations (and only in those cases):

- The internal reputation value of any device or service changes "considerably". What this specifically means is dependent on a rule on which the "Reputation Manager" and the "reputation consumers" must agree. Typically, the reputation values have a continuous scale, so small changes will have no particularly important consequences, but the "Reputation Manager" and the "reputation consumers" agree on thresholds for those numbers that when they are reached, an alert is produced.
- Any enabled device or service is producing unreliable data (see Requirement RM12 below for an explanation on enabling devices and services)
- Any disabled device or service is producing reliable data (see Requirement RM12 below for an explanation on disabling devices and services)

Note that it is very important that the model produces the alert only in those situations, because otherwise producing an alert for each trusted or untrusted data could result in an involuntary DoS attack to the system because of producing a lot of alarms that the system might not be able to process.

2.1.9 Requirement AM9: Ability of Redemption of Sevices and Services

The model must contemplate the ability of redeeming untrusted devices and services by external means, such an administrator resetting the reputation of a failing device after rebooting or repairing it.

2.1.10 Requirement RM10: Ability to react to Reputation Evaluations

The model must define mechanisms to let systems react to changes in the evaluation of data, devices and services. This reaction must include but not necessarily be limited to:

- Disabling access to untrusted devices and services by switching the state of the device/service to 'disabled' in order to the authorization components have this information available and reject any access to it;
- Enabling access to those devices and services that were previously not trusted but are not untrusted any more by switching the state of the device/service to 'enabled', in order for the authorization components have this information available and not rejecting accesses to it for trust reasons (that is, they can still reject the access for any other reason;
- Logging the change in Reputation Evaluations for devices and services; and
- Communicating the changes in values to an external agent in order to allow him to fix the problem or know that the problem is fixed.

2.1.11 Requirement RM11: Generic Reaction system

Due to the dynamic nature of the IoT, it is impossible to know in advance which will be the requesters of the system or the purposes of the request. For this reason, it is impossible to provide a full set of predefined actions that cover all possible reactions. As a result, the model must define a generic mechanism that allows defining the execution of any action at run-time assuming it has already been implemented. The mechanism must also be able to pass the information of the alert that caused the reaction to the action to be executed.

2.1.12 Requirement RM12 Support Integration with an Authorization Mechanism

The model must define a mechanism that allows that the reputation values generated in the system are available to the authorization components in order to make possible the rejection of those requests that address untrusted devices or services. It must also be able to provide warnings for those evaluation criteria whose value is lower than a threshold defined by the System Administrator.

2.2 Analysis of the possible trust approaches for IoT

There are several approaches to trust evaluation for IoT systems. Trust is crucial in the IoT, as the infrastructure and its users are distributed and quality of services depends on quality of data, which is the accuracy, availability and timeliness of data. (Further properties, like the completeness, unambiguity, meaningfulness and interpretability of data are difficult to observe with our methods and a more intensive model-based procedure would be necessary). Nevertheless, trust in data quality is not an objective notion: every entity with a different context and purpose may assess trust in a device or service differently, i.e. subjectively and in a context-dependent fashion.

Trust and reputation computation functions may get complex. A trust management system may be using both direct observations and indirect recommendations to aggregate and update trust. Trust evaluations based on direct observations may for instance weigh more than those based on certain recommendations. Older trust values may weigh less, if trust is assumed to decay over time. Weighted (indirect) recommendations need to be compared to and aggregated with (aged) past experiences. The context and the purpose of trustor and observer for evaluating trust also must to be included in the trust computation, as mentioned already. Trust assessment accuracy and trust convergence time may also be of interest.

Trust according to [Daubert 15] can be decomposed into entity trust, device trust, and data trust:

- **Entity trust** in the IoT refers to expected behaviour of participants (persons, services, devices, etc.). IoT-A in D4.2 talks about collecting user reputation scores and calculating service trust levels. Transferring trust approaches from social networks seems a promising approach in certain use cases. Approaches like behavioural attestation are still experimental;
- **Device trust**. There are several existing approaches, such as trusted computing as well as computational trust (see [Joesang 07]), that can help establish device trust; and
- **Data trust.** Even in the case of potentially untrusted device, it may be possible to trust the values that they generate. This is particularly true when redundant data originates from many independent devices. ("Independent" means here that the possible failures or corruptions of the different devices are relatively independent of each other. In other words: this is not true if there is a common reason for the failing of the devices or if all have been compromised systematically by a coordinated global attack). Trusted data might need to be derived from untrusted sources. IoT services derive new data, e.g. via data fusion and mining. For those newly generated data, a new trust assessment is required.

Trust management approaches may be centralized or distributed. Trust models may be multi-dimensional, observing for instance quality of service, security, reputation, and social relationships. In order to reach a single trust conclusion, multiple trust dimension values may need to be combined for instance using a fuzzy approach. The result of a trust evaluation process may be integrated into an access control mechanism, in order to ensure information exchange between trustworthy entities only (as for instance proposed by [Bernabe 15]).

There are many fields in which trust management is an established topic, like

- Web Contexts
  - Social networks, like Facebook
  - Electronic commerce, like Amazon or EBay
  - Semantic Web [Richardson 03]
  - Peer-to-Peer Networks [Marti 06]
- Network Management
  - (Wireless) Sensor Networks [Chang 12]
  - General Ad-Hoc-Networks [Cho 11]
  - Vehicular Ad-Hoc-Networks [Soleymani 15]

In social online contexts, like social networks and electronic commerce, trust management is essential to gain and maintain consumer trust. Here both social relationships between humans participating in the system as well as feedback and behavioural studies help to manage trust.

Also network management people are familiar with evaluating trust towards other network nodes, for instance based on the reliability and speed a node is delivering packets.

When using data sources, especially measurement data, one may need to know how reliable the contents and their sources are. Besides relying on external certifications and user feedback, one also may compare the produced data with expected data ranges derived from a conceptual model to detect implausible data.

All those three approaches can be helpful to handle trust management in IoT to a greater or lesser degree, depending on the actual deployment context. Now we proceed to briefly describe current research activities to migrate such trust management techniques to IoT.

The three main trust management techniques are based on Social Trust, Network trust or plausibility verification of data. They will be reviewed in the three next subsections. But there are several other approaches to trust, the most important ones are (see [Zan 14]):

- Trusted computing / the Trusted Platform Modules (TPM) in smart devices. The TPM is a microcontroller that combines robust cryptographic identity with remote security management features such as remote attestation. This approach is gaining quite a bit of support from vendors and academic communities, in particular due to the efforts from TCG [TCG-IoT 15], [Intel-EP], ARM [ARM-IoT], and many others. Although the approach is promising, the costs in terms of energy consumption still seem to be problematic for the applications that are in focus of RERUM;
- Fuzzy trust management (also from network management) [Chen 11]; and
- Trust seals obtained from external assessments performed by trusted third parties (see [ENISA 13]). In principle, this type of external information may be quite useful, but still here there is a lack of standards, real-time validity checks, usability, market presence and maturity.

2.2.1 Social Trust Management Techniques

A trust management system may consider social trust and QoS trust metrics, e.g. subjectively via post-transactional feedback on service quality and service provider satisfaction.

Social parameters may be used as part of the trust quantification for a specific IoT device only, if IoT devices are able to establish social relationships with each other. In this case IoT devices can be grouped in communities according to the social relationships which are set among them (and their owners/users). For example IoT devices may be in close contact because they belong to the same owner or their owners are friends or co-workers, or part of the same family. Other kinds of relationships are parental, employer-employee, and co-location object (distance below a certain threshold) relationship. Related objects may be built in the same period by the same manufacturer. A community relationship may be shared among devices that are in contact because they (or their owners/users) share a set of common interests. It is often assumed that, when the social relationships between devices get closer, the trust relationship among them get stronger as well, resulting in the need to compute weights of a kind of relationship between the devices, e.g. evaluating the degree of Interest-In-Common between the two devices as well as the Friends-In-Common (see [Nitti 12] and [Bernabe 15])

Trust properties in social encounters may include honesty, cooperativeness, and community-interest to account for social interaction. Honesty may be computed using a set of anomaly detection rules such as a high discrepancy in recommendation has been experienced, as well as interval, retransmission, repetition, and delay rules. Friends might be assumed cooperative toward each other. The number of common friends for instance may be assumed to correspond to the level or degree of cooperativeness. The level of community-interest as another example may be assumed to correspond to the number of common community/group interests of two given nodes. Thus exchanging friend-lists and community-profiles on each encounter consequently could support to compute these two values (setting aside privacy issues).

If devices or their owners/users or services are capable of giving feedback, this may be used for a trust model that takes into account recommendations from others about a particular device, service or user. The trust model needs to weight each recommender/observer/witness in order to limit their influence according to a recommender's behaviour in the past. Thus, the opinions are subject to a credibility process (see [Bao 12]) regarding smart objects in IoT as sometimes being human-carried or human-operated devices. It considers an IoT environment with no centralised trusted authority. Every IoT device has an owner and an owner could have many devices. There is no centralised recommendation database assumed. A scalable trust management protocol for IoT in this context may need to emphasise social relationships. Each node performs trust evaluation towards a limited set of devices of its interest only. The trust management protocol here is event-driven upon the occurrence of a social encounter or interaction event (due to the decentralised approach). For scalability, a node may just keep its trust evaluation towards a limited set of nodes which it is most interested in.

[Nitti 12] focuses on a social Internet of Things (SIoT) paradigm, where objects are capable of establishing social relationships in an autonomous way with respect to their owners. This study analyses how the information provided by other members of the SIoT has to be processed to build a reliable system on the basis of the behaviour of the objects using a subjective model for trust management. Each node computes the trust of its friends on the basis of its own experience and the opinion of common friends with potential service providers. A feedback system is employed and the credibility and centrality of the IoT nodes are applied to evaluate the trust level considering some objective and subjective properties of the trustee without focusing context.

[OHara 14] categorizes “trust strategies” for the Semantic Web based on how agents react to peers they have no personal experience with. [Fritsch 11] analyses a transfer of these strategies to IoT. The five basic strategies are:

1. optimistically assuming all strangers are trustworthy unless proven otherwise
2. pessimistically ignoring all strangers until proven trustworthy
3. investigating a stranger by asking trusted peers
4. transitively propagating the investigation through friends of friends
5. using a centralized reputation system.

When no reputation information can be located, an agent must decide whether to transact with a stranger based on its own stranger policy maybe using something like a “generosity” metric.

There are several works analysing trust management issues in peer-to-peer-networks (such as those used for file-sharing), like EigenTrust and PeerTrust.

EigenTrust (see [Kamvar 03]), is one of the most known and cited trust management systems in peer-to-peer networking. It is characterized by the assignment of a unique global trust value to each peer in a P2P file sharing system, based on the peer’s history of contributions.

PeerTrust (see [Xiong 04]) is a trust and reputation model that combines ideas from the management of trust and reputation in distributed systems, such as: the feedback a peer receives from other peers, the total number of transactions of a peer, the credibility of the recommendations given by a peer, the transaction context factor and the community context factor.

In general social trust management techniques in IoT have three problems: first, it is easy to mount defamation attacks or to artificially boost the reputation of malicious services. Second, it entails many privacy problems, when users have to disclose which services they are using in order to rate their trust. Even if the users anonymously exchange information, existing de-anonymization techniques would disclose personal information, starting with cliques of friends, preferences and the like. Moreover, this type of trust adds a “second-order trust problems”: the users themselves that provide trust ratings must be rated as well. This is almost impossible to do if the users act anonymously or if rapidly varying pseudonyms are used and would lead again to defamation or reputation boosting attacks.

2.2.2 Network Trust Management Techniques

WSNs are composed of sensors monitoring environmental conditions such as temperature, sound, vibration, pressure and motion. [Chang 12] classifies trust mechanisms in WSNs according to topology control (assign and allocate transmission power deterministically or not), coverage (showing the sensed area in WSNs and relocating sensors to improve coverage), localisation (need location to manage sensors data range-based or not), and target tracking (detecting target sensors by measuring the energy of signals emitted from the targets). In order to achieve transmission security, WSNs consider sensor nodes' trustworthiness. Issues about trust management in WSNs according to the paper are cluster (neighbour nodes in a cluster compute reputation to decide whether a sensor node can be trusted or not), aggregation (opinions and trust ratings are reported to the cluster head and from there to the base station) and reputation (coordination with local voting algorithm).

[Rani 14] reviews trust models for WSNs, which contain many sensor nodes with comparatively little memory and power, being vulnerable to insider and outsider attacks. As trust metric they discuss e.g. data and control packets forwarded as well as their precision, node availability, battery lifetime, and data consistency. Trust is used detecting a node which is not behaving as expected (either faulty or maliciously). They classify several trust models classified by Trust Model, Structure, Information modelling, Information gathering, propagation, and dissemination, redemption of nodes, and the attacks they can detect.

[Nitti 12] focuses on wireless sensor networks, where a trust model is required to find malicious, selfish and compromised nodes. [Probst 06] uses a social trust approach for WSNs, basing initial trust on the reputation scores given by others, and adapting that initial trust by experiences from direct interactions with and observations about that node taking into account context. Sensors may observe sensor readings or experiences of nearby nodes and compare them with their own sensor readings and experiences. Also they may observe the data propagation and aggregation behaviour of their neighbours and compare them to their own. Each node keeps a limited local experience cache, where observations may become stale and evicted.

[Sicari 13] proposes a hybrid architecture combining wireless sensor networks (WSNs) with wireless mesh networking for secure data aggregation and node reputation in WSNs using a secure verifiable so-called "multilateration" technique that allows retain the trustworthiness of aggregated data efficiently even in the presence of a malicious node.

An IoT application consisting of only wireless sensors is mainly interested in QoS trust metrics, like packet forwarding/delivery ratio and energy consumption, availability, throughput, or delay (see [Bernabe 15]). Security metrics, like communication security, authentication and authorization capabilities of devices, and their current network scope (same LAN, same prefix, Internet), and general device capabilities may also influence trust rating (see [Bernabe 15]): if data is not properly secured the values are less trusted as they may have been manipulated.

[Gomez 08] proposes a trust model for wireless sensor networks (WSN) inspired by the behaviour of ant colonies. It allows finding the most trustworthy path leading to the most reputable service provider in a network. It is adaptable to sudden changes in the topology of the network as well as in the behaviour of its participants. A set of "ants" (artificial agents) is launched through the WSN. On their search they leave some "pheromone" traces in every link connecting two nodes, qualifying their opinion about the trustworthiness and viability of that link.

Trust models in vehicular Ad-Hoc Networks (VANETs) according to [Soleymani 15] may be entity-centric (trustworthiness of vehicles), data-centric (level of trust for each received event message), or combined. [Raya 08] focuses on data trust and derives trust in data (e.g., reported event) from multiple pieces of evidence (e.g., reports from multiple vehicles). They weigh each individual piece of evidence and take into account various trust metrics, such as time freshness and location relevance, and context (here vehicular Ad-Hoc networks) using for instance voting, Bayesian inference, and the Dempster-Shafer Theory of evidence. Nodes in [Raya 08] either comply with the implemented protocols (i.e., they are correct) or they deviate from the protocol definition intentionally (due to either internal or external attackers) or unintentionally (faulty nodes).

[Saranya 15] addresses the domain of VANETs and takes into account attacks like false event generation, data modification, data dropping and data flooding. Their proposed algorithm establishes security in a VANET through assigning trust levels for nodes in the network using both reputation and plausibility checks. The algorithm focuses safety-related information broadcasted in single hop and relayed in multi-hop through intermediate nodes. If a node receives an event notification, and is itself in the detection range of that event and has no information about the event, it decreases the trust value of the sending node and broadcasts a malicious intent control; otherwise it increases the trust value of the sending node. If not in the detection range it requests confirmation of other nodes known to be in the detection range and proceeds in a similar manner.

2.2.3 Plausibility Verification

A basic problem related to plausibility verification is to find the best possible estimates of the hidden variables of a system when tracking its state, based on partial information, given by observations (noisy or ambiguous) which are sequentially arriving. The particle filter (PF) methods offer procedures to provide such estimates for location information (positioning), giving a numerical approximation to the nonlinear Bayesian filtering problem [Gustafsson 10]. [Bissmeyer 12] uses a Particle Filter (PF) to perform a data fusion of several location-related data sources in order to check mobility data plausibility of single-hop neighbour nodes locally in each vehicle, in order to detect inconsistencies introduced by faulty nodes or malicious (insider) attackers in VANETs. The PF uses an adaptive stochastic grid that automatically selects relevant grid points in the state space with linear complexity in the number of grid points. They are used to assess both the trustworthiness of the currently analysed message and the general vehicle trustworthiness.

[Momani 09] surveys security and trust concepts in wireless sensor networks and finds the issue of trust in wireless sensor networks is being explored with a focus on trust associated with routing messages between nodes (binary events). Still little attention is paid to DPT, i.e. the data content, i.e. the monitored events and reported data, both continuous and discrete, for which trust models addressing the continuous data issue are needed.

[Javed 12] addresses DPT by investigating how to verify sensor information that is gathered from multiple sensors that are managed by different entities using outlier detection. They automatically derive a model of the physical phenomenon that is measured by the sensors, then compare sensor readings and identify outliers through spatial and temporal interpolation. This works as long as the sensed phenomenon is essentially continuous, like weather.

2.2.4 The Trust Model for RERUM

It is not difficult to see that the methods described so far have a disadvantage when used in IoT scenarios: they all are relatively heavy-weighted and do not allow a stream processing, on the fly analysis of data close to the data sources (which is a basic privacy property we do not want to endanger). Further, most of them require a complex behaviour model and can’t be easily calibrated or modified for different types of data.

The proposed Trust Evaluation Model for detecting malfunctioning devices can be better understood taking a look at a simplistic – but very important – method: check if the provided values are inside a static range of values. This range may depend both on the characteristics of the sensors and on the environment where the sensors are placed (say, a value below 5°C inside a city building in Tarragona should normally raise a warning). This trivial method has also many advantages:

1. It can be easily calibrated by non-specialists to increase the sensitivity of the alarm generation or to reduce the number of false positives;
2. It does not require that large amounts of data are being kept over time for purposes of the trust analysis, but the data can be analysed immediately on what is called *steaming mode*;
3. Data is not required to be transmitted to a different location (say, central), but can be analysed close to the data collection points. This is a privacy principle that we would like to secure;
4. The general method works well for all types of sensors and physical variables (temperature, humidity, light intensity, etc.);
5. It does not require a complex or costly or work intensive behavioural model;
6. It does not rely on large amount of possibly redundant or at least related data, like some modern data mining approaches do;
7. In the case of intermittent failures, such that the data stream contains some outliers, but also valuable data, it allows the creation of “corrected” data streams, *filtering* the data streams by eliminating anomalous values; and
8. It is has a very high performance.

This simplistic method, on the other hand, is not very effective, it is not flexible and it does not adapt to changing environments. A temperature of 35°C may be by itself not strange in a street of Tarragona, but a sudden jump of 25°C to 35°C within seconds is indeed an indicator or a strange behaviour. What we would like to obtain is a Trust Evaluation Model that has, besides the aforementioned advantages also the following ones:

9. It is effective: it detects a large class of anomalous (or, at least, suspicious) behaviours, like values outside of moving intervals, jumps of a relatively large size, etc.
10. It is adaptive and depends on the recent past: the alarms raised depend on the parameters “learned”, and those parameters are adapting themselves to the changing situations.
11. It is flexible: it is possible to integrate behavioural models, if available, or may use methods of data-fusion, or may use models that are specific for particular physical values and situations.

It is easy to design a method for comparing sensor values to dynamic (varying) ranges of values. For instance, calculate an adaptive mean estimation and use an interval of fixed width around the adaptive mean. But there are no established streaming-mode methods for calculating more interesting parametric and non-parametric statistics other than the mean (like standard deviation, quantiles, characteristic “jumps”) that can be used detect on-the-fly abnormal sensor (or network) behaviour. This forms the basis of our proposed solution.

As discussed above, the most relevant trust approaches are Social Trust Management, Network Trust Management, and Plausibility Verification. As the RERUM trials are only starting, we have not yet collected social trust information from users. This is only possible after the satisfaction of the users with the services is evaluated systematically. Besides that, social trust management raises many privacy issues that are not fully understood yet, particularly in the context of IoT, like the detection of cliques of friends, the types of services that the different individuals are using, etc. Thus we concentrate on Network Trust Management (discussed in Section 3.10 -- Trust Reputation Evaluation for Devices at the network level) and Plausibility Verification (discussed in the rest of this Section 2).

IoT-A does not discuss how to design or implement a trust and reputation mechanism, but recommends a set of "critical" guidelines which should be ensured. Following all their suggestions

(IoT-A, Deliverable 4.2, Section 4.5.3, Design recommendations & security threats, [IoT-A-D4.2]) our proposal has the following characteristics:

1. We base our trust model on the **five steps (or "components"): observation, scoring, entity selection, transaction, rewarding and punishing**, as originally proposed by [Marti 06]. See details in the next Subsection;
2. Our trust model does not depend on the existence of **identities**, but it allows entities' **anonymity**. How this is done is not spelled out here in full detail, but the procedure should be clear: the observer that wants to observe and rate a service or stream must first acquire the pseudonym that the stream is using. This is relatively easy to do, since the observers are core RERUM components, but it is a delicate point, as a misuse of this interface would lead to privacy leakage. Also, the integrity of the observer (which is basic system functionality) must be maintained;
3. When weighting the collected behavioural information and computing the trust and/or reputation values, **the more recent a transaction is, the more weight it should have**. In our model this is implemented using exponentially smoothing, which, in the case of means it simply assign exponentially decreasing weights over time. The use of exponential smoothing techniques for other types of statistics (Confidence Intervals, quantiles, etc) is new and is developed here in this deliverable;
4. The **subjectivity in the assessment of a transaction is explicitly allowed**, i.e., each entity may have its own criteria when evaluating a received service. In our model, a user or service may create his own rules for rating the trust of a given service to which he has access grants. More precisely, users have the possibility of defining the rules that the Reputation Manager uses to calculate the trust that the user queries. In some cases, a privileged user has the possibility of creating *observer rules*, that is, to determine the new criteria to analyze the activities of the service to be rated from a particular point of view. A non-privileged user that is interested in defining new observation rules must ask an administrator to create and activate them;
5. **Redemption** of past malicious entities that have become benevolent is possible;
6. Benevolent **newcomers have the opportunity to participate even** if there are already trustworthy entities in the system, but, on the other hand they do not have more opportunities than non-malicious remaining nodes in the network. The exact rules for newcomers and their share of activity in the system (an in which parts of the system) can be easily modified by expert rules as desired;
7. An abuse **of a well achieved reputation** should be avoided: A **quick, accurate and effective detection** of a **repeated misbehaviour** by a malicious entity who reached a high reputation should be carried out by a system making use of a trust and/or reputation model;
8. Every entity should receive a different trust and/or reputation rating depending on the **type of service** (**purpose**) it is providing;
9. Our trust and reputation model takes care about the overhead they introduce in the system, in terms of **bandwidth** and energy consumption. In general, the observers can be placed in a distributed way, each one close to close to the data stream generators that he is observing. This architecture allows the early aggregation of data, avoiding sending all data to a central repository; and
10. The **importance** of a transaction or its associated risk influences in the subsequent punishment or reward accordingly. The more relevant a transaction is, the higher it should be punished if it is not properly carried out.

Beside those points, the first question is: what entities do we want to evaluate? Let us state upfront that our main focus is on *RERUM Devices* and *basic IoT services* closely-related to devices, simply because this is the focus of RERUM itself. We will be interested in establishing a trust model for:

- **RERUM Devices.** The main question that interests us is: it the data stream produced by a device or smart object with embedded sensors trustworthy?
- **Services.** In a certain sense, this point subsumes the previous one, since the data streams generated by smart devices are exposed as IoT services. Thus if we tackle the trustworthiness of services, we also tackle the one of devices. Nevertheless it is easier to consider them separately: first only services that generate streams (but include on actuators) and the more complex services, which includes federation of services or actuator functionalities as part of the service.

We will be not assign trust values to Single Data Values based on plausibility verification procedures. It is possible to imagine that in a data stream some values could be considered as trusted because they are "reasonable" and agree with the process model, while some outliers or otherwise "strange" values should be considered as untrusted. Further, one could propose a trust evaluation mechanism that adds this information to the meta-data of each trust value. We will not do this, as we believe that this solution would be too expensive. In Section "Observers of a Service with systematic errors" we will discuss this situation and propose a solution along a different path. We will assign trust measurements to single data values based on the integrity protection status of those values (see Section "Trust for single data values").

Of course, users (administrators, privileged users, external users, etc.) themselves may need to be trust-rated, not just devices or smart objects. Humans may behave in a suspicious manner, despite having proper credentials and this may e.g. reflect in adjustment of their future privileges. Nevertheless, we have yet no information about the behaviour of users in RERUM.

The second question is: what aspects of a service do we want to evaluate? Our rules are designed to assess the trustworthiness of services or devices with regard to:

- **(Plausibly) correct values.** Is the service or device generating data that seems plausible? If not, we will assume that the device is malfunctioning, the sensor has a problem or the service is not working properly (or that there is an intermediate node altering the data, etc.
- **Plausible aggregated values.** Note that a device may be producing incorrect values, but in a plausible way, and the error may remain undetected. This is why it will be important to have *redundancy* or *correlated data*. An observer may for instance compare the thermometer values with the heating measurements (and commands) and the outside temperature and detect errors in the thermometer that would otherwise pass unnoticed.

We will be not establishing a trust model for Privacy. Here the question is: Is the service we are planning to use protecting our privacy? RERUM nevertheless provides PETs that services may use to protect the users privacy.

2.3 Overall structure of the Trust Model

This section introduces the conceptual model that will be used to define the logic of the trust model. As such, it is considered to be a mechanism to express the logic of how to evaluate the reputation of the data and services in the system and react to that accordingly.

The trust model is made up of several components that work together by exchanging information. In order to provide an overall view of the model, this section explains what these components are, what they do, and what the information they exchange with each other is. But the internal details of how each component carries out its work are explained in Sections 2.4 and 2.5 Reacting Model.

2.3.1 Overall view of the Trust Model

The trust model is comprised of two main sub-models, the Trust Evaluation Model and the Reacting model. The Trust Evaluation Model provides and refines Reputation Evaluations for incoming data and their corresponding devices and services by evaluating the reliability of the data. The Reacting Model is responsible for making the rest of the world to react to those evaluations, from something trivial such as logging those evaluations to something as complex as executing ad-hoc programs to deal with them. The following figure shows the different components of the model and how they interact with each other.

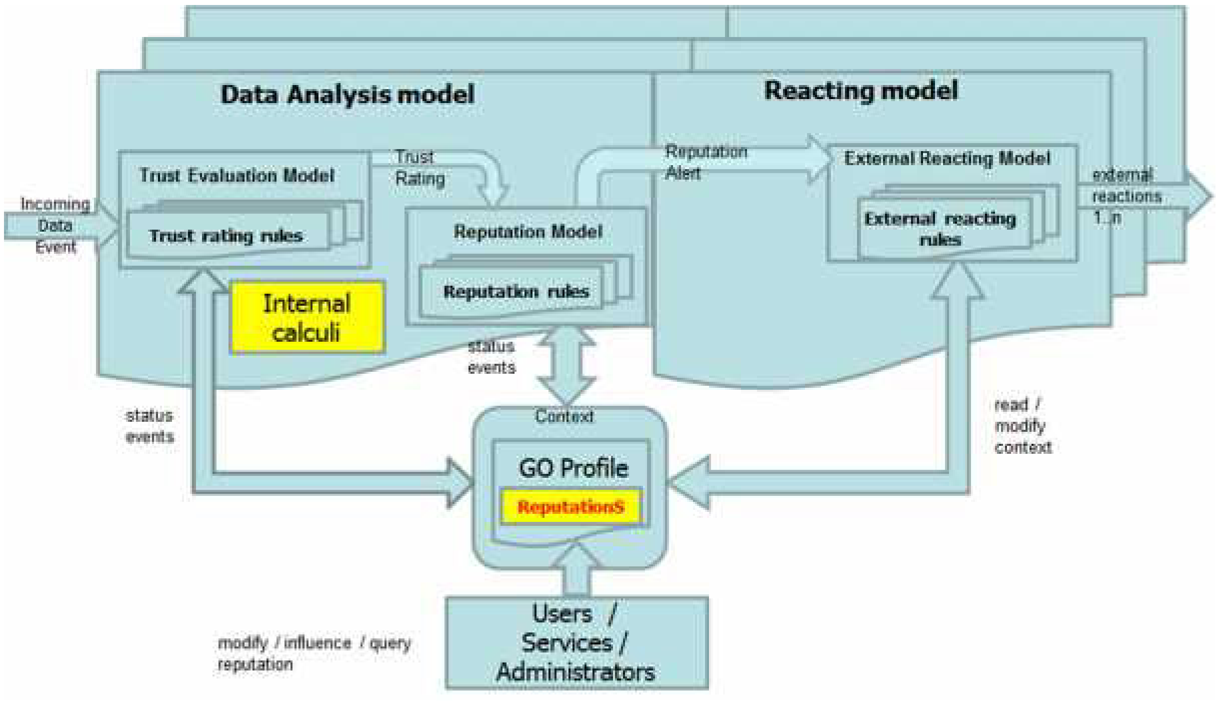


Figure 2: Overall view of the Trust Model

As the figure shows, the data from the system arrives to the model in the form of incoming data events. These incoming data events are explained in Section 2.3.2.5. We assume that incoming data events arrive in separate "streams", one stream per data source (that means in the case of RERUM, per service/device). In addition, we assume that the Trust Evaluation Model has all necessary information external to the model associated with the single streams to evaluate the reliability of the data received and update accordingly the trust on their corresponding devices and services.

The picture shows how these incoming data events feed the multi-component Trust Evaluation Model. This component is a logically (and also often in practice) distributed set of observers receiving the event streams. Each observer contains a set of rules that describes the criteria to produce a trust rating for both the data received and its associated device and service. For instance, one observer might be more interested in the accuracy of the data while another one could be more interested in the availability of the associated service. The trust ratings calculated by the different observers are aggregated to a "reputation value".

The Reputation Evaluation model is a logical module, containing reputation rules, whose goal is to define the criteria to gather the different trust ratings to produce a joint reputation value from all the trust ratings produced by the observers. Note this joint reputation value is actually a set of reputation values, so it can be used for different possible purposes. Once these reputation values are calculated by executing the rules that comprise the reputation model, a Reputation Alert containing that evaluation may be generated so the Reacting Model can execute the proper reactions for those evaluations. That is, a single Reputation Alert could potentially have several reactions associated to it.

The Reacting model is another set of rules. In this case, they define how to react to each Reputation Alerts. These reactions might be as simple as storing them and warning an administrator about them or as complex as executing an ad-hoc procedure for the alert. It can also serve to refine the evaluation process by correlating multiple alerts to influence the Reputation Evaluation itself or, in the concrete case of RERUM, to refine the access policies to take into account the result of the Reputation Evaluation.

Finally, the figure shows that the three sub-models access a common area named context. This common area is an interface to a persistent store that exists independently from the model and contains information regarding the services and devices that produce the data itself. This store is used to permanently store any information regarding the devices and services so it is accessible later, even if the hosts running the models were forced to restart. One example of such information the profile associated to the generic Virtual object (GVO) associated to that device or service, which will contain the reputation value associated to each of the purposes that it can be queried for.

2.3.2 Components of the Trust Evaluation Model

A Trust Evaluation Model has different types of components: *active* and *passive* ones. This is similar to access control systems, like XACML: which also has active modules (say, the Policy Enforcement Point, the Policy Decision Point, etc.) and passive ones, like the access control policies themselves. Moreover, there are other elements, like interfaces, measurements, etc. The processing in the active components (calculations, decisions, production of alarms, etc.) is determined in particular by the passive ones, by the rules or policies. This is true both for XACML and also for our trust evaluation system.

This Section presents not only the active elements of the model, that is, the entities that process information, but also their passive counterparts: the rules (or policies), measurements, ratings, the interfaces between active components, etc.

In the literature there are many different definitions of trust – not all equivalent – and different methods have been proposed to calculate the trust of a service. In the following, besides the main active and passive components of our system, we will also discuss in more detail how the components in the Trust Evaluation Model work.

2.3.2.1 The *active* modules of the Trust Evaluation Model

The trust evaluation model contains the following active modules:

Observers: We call *Observer* the RERUM entity that observes or monitors the data streams or services, compares the observed behaviour to some expected values using pre-defined trust rating rules, and as a consequence produces the trust-ratings. To answer a trust query for a particular service, it is necessary to analyze systematically the data streams on which the service provision depends. A service can be trusted only if the data on which it depends is also trusted. The *observer* analyzes the behaviour of the services by observing the corresponding data streams (and in some cases also the commands sent to the service) and perhaps also related data streams that offer some redundancy or side or contextual information.

Reputation Manager: The Reputation Manager combines the different trust-ratings of the observers to a coherent value, or to a set of coherent values. When several Observers are evaluating the same service or stream, they usually end up producing different trust ratings of the same service. This is natural, since the different observers use different rule sets. The entity that performs this task is the Reputation Manager (or Reputation Engine). It is necessary for the RERUM trust management system to have a central Reputation Manager to collect all relevant information to evaluate the reputation value of a service for a given user or entity. For instance, even if a given service is monitored by a single observer producing the trust-ratings, the reputation engine must correlate other relevant information on that particular service to produce a reputation value.

2.3.2.2 The *passive* modules of the Trust Evaluation Model

The trust model is based on static, passive rules that determine how the active modules process the information. Those passive modules are the following:

Trust-rating rules: If a stream of values (data and actuator commands, if present) is being analysed, then the observers are interpreting the behaviour of the service producing the streams and trying to decide if the observations are "normal" or they are indicators of some type of fault. The trust-rating rules express under which conditions the data stream is to be understood as anomalous or "not normal". If a process model is available, or a fault model, or a diagnosis system, it can be incorporated in the rules too to facilitate the task. But even without a specific process or fault model, privileged users or administrators can specify the static or dynamic (time-varying, depending on the current state of the system) ranges of values in which a data stream should remain to be plausible, and which unexpected jumps of the stream values are to be considered as anomalous. Moreover, the rules may be used to "probe" and test the system, sending commands to the actuators and observing the reaction, in comparison to the expected one.

Reputation rules: The reputation rules express the logic to be applied for combining the single trust-ratings to one summarizing entry, the reputation.

Trust rules: The trust rules express the logic used by the Reputation Manager to answer queries of a user or service, when he needs to know if it can trust a service or a data (stream) for a certain purpose in a certain context.

2.3.2.3 The *dynamic* measurements, ratings and other trust-related values:

Indicator: When comparing the behaviour of a service with the expected behaviour, the observer may detect deviations from his expectations. As an example, a thermometer may be sending consistently values around 25° and suddenly jump to 50° or drop to 0°, unexpectedly. Such jumps may be due to physical processes (a fire, an ice box on a sensor) and the sensors are reporting correct values, but more probably those jumps are due to errors in the measurement or transmission. Since the observer has no full information about the process itself and the concrete reasons that lead to those observed deviations, the observer is not able to decide unequivocally if the service is acting according to the specified rules or is behaving incorrectly or maliciously. Nevertheless those deviations should be considered as *indicators* of an error. On the other hand, the observation of behaviour consistent with expectations is an *indicator* of correct behaviour.

Trust-rating: Some indicators of unexpected or abnormal behaviour may induce an observer to modify his appreciation of confidence, reliability or trustworthiness of a service. This information is expressed as a number called the *trust-rating* of the service that this observer is producing. A trust rating contains at least one value, but also may be more complex data structure explaining how to interpret the value or values or an indicator of the *trust-rating rules* that were used to calculate the value.

Besides observers, also administrators and other privileged users are allowed to produce valid trust ratings (see below for a more detailed discussion).

Reputation: It is necessary to merge the trust-ratings appropriately to a coherent value, according to some well-defined rules. The reputation is in general not a single number, but usually a more complex data structure than just a number. The reputation will be used to answer trust queries from users or services, which depend among others on the *purpose* of the trust. This information is called the *reputation* of the data stream or service.

Trust-value (or "trust"): The trust value provided for a User A about the service or stream S may depend on several factors such as the following:

- the past behaviour of the data stream or service, compared to the expected "normal" behaviour (or also compared to the existing fault models, if any);
- who is asking for a trust value, and in particular what his trust policies are;
- the context on which he is asking;
- the purpose for which he wants to use the data or the service (for some purposes an approximate value may be fine, but for others a more exact value is needed);

We could also mention the following concept, although strictly speaking it is outside of our model:

Trustworthiness: Trustworthiness, according to the definition that we are using, cannot be calculated. If an entity is trustworthy then that entity *will behave* honestly, according to the expectations from the other honest parties. Trustworthiness is, strictly speaking, outside of the scope of our system. In our case, we only estimate the trust-value for the current state, based on the past values. Therefore, the concept of trustworthiness or the algorithm of calculating it is out of scope of this deliverable. Reputation and trust ratings are particular types of estimators of trustworthiness, but notice that the trustworthiness of an entity is something that can never be measured with certitude: an entity may have behaved absolutely correctly in the past in order to gain trust and reputation and then abuse of this trust and perform an attack.

2.3.2.4 The *Interfaces* of the Trust evaluation Module

The Trust-evaluation system, and in particular the active components, the observers (trust rating modules), and the Reputation Manager must provide the following interfaces:

- An interface used by services or users for trust queries from the reputation engine.
- An interface used by administrators or trusted and privileged entities to enter the result of their observations and experiences, regarding the trust of data streams or services. In this case, the administrators or privileged users are acting as *observers* and the values they enter to the reputation management system are their *trust-ratings*. The Reputation Manager must know what is the provenance of the trust-rating and possibly an indication of the priority or importance of the trust rating and the reason for it or the type of purposes that if affects.
- An interface for privileged users to enter the policies of new observers or to change the policies of observers for which they are authorized to do so.
- A particularly sensitive interface used by privileged administrators or users to enter the policies (rules) that control how the trust is evaluated at the Reputation Manager.

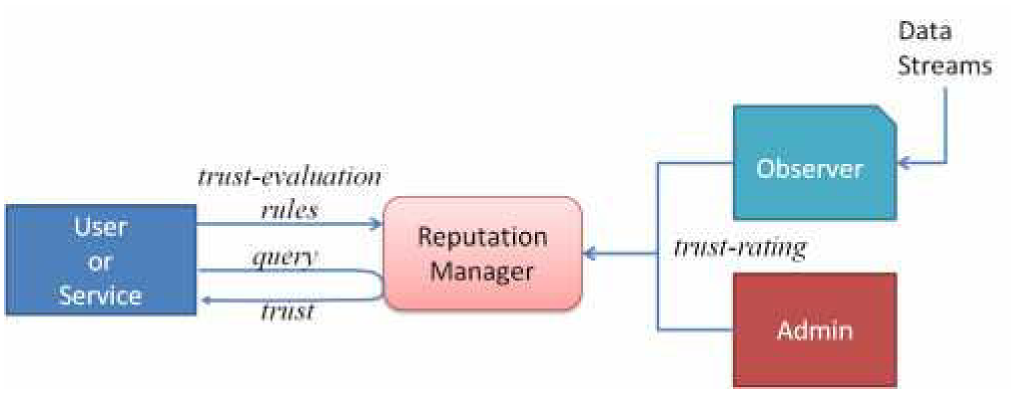


Figure 3: Interfaces of the trust management system

Figure 3 shows the interfaces of different trust management entities such as the observer and the Reputation Manager.

The trust-evaluation rules should be written in a standard declarative language. But also the observers require rules: they observe the data streams or services and produce trust-ratings for them based on some customizable rules. It is reasonable to use the same declarative language for both types of rules. A declarative language (like Datalog, CLIPS, RuleML, or Haskell) is well-suited to declare the policies. It is possible to use a conventional imperative programming language, like Java or Python, but in that case the procedure for evaluating the rues can be a complex venture. A low-level language like SQL it rather unsuited for declarative rules. We propose to use CLIPS [CLIPS-RefMan-6.30-2015] for our purposes, a modern and industrial-strength policy language. It is adequate for expressing the rules directly, without any "coding".

2.3.2.5 Incoming data events (or data streams)

The Internet of Things delivers data which is moving rapidly from sensors and devices to consumers. Consumers can be end-users or could be in turn also services, which aggregate the data, enrich it with external context information, and create in this way new data. In this model we assume that each service is producing a stream of data and that observers can "subscribe" to those streams. The observer may be "subscribed" to one or several data streams for the purpose of evaluating them (or some of them). That data is arriving in streams is not a too strong assumption: it is easy to construct a *converter* that receives, or "pulls", or "queries", or "reads" the data and produces the data streams that are to be used by the observer.

An "incoming data event" happens when a data arrives from one of the streams of data. We assume that the values of each stream arrive sequentially.

The important assumption is that when a data value of any stream arrives, the observer (or collectively, the Trust Evaluation Model) is able to identity unequivocally to which stream it belongs and has all necessary information to interpret correctly the data, that is: it knows the identificator of the stream, the value data type, the static parameters of the service (expected range, expected frequency, relation to other observed streams, description and semantics of the service), etc.

Some services do not only provide data, but they also accept *commands* that are delivered to actuators. In this case, the observer may additionally decide to send commands to probe the reaction of the system, thereby testing the possible explanations of discrepancy of the received data streams from the expected values.

2.3.2.6 Observers of a Service with actuators

Some services receive commands that are delivered to actuators, in order to open or close a door or a window, to turn on or off the heating, to change the music, etc. In this case, the service is trusted if not only the corresponding sensors are producing correct values, but also the actuators are functioning properly. Observing this service should include not only observing the data produced by the system, but also the reaction of the actuators.

Of course, this is not easy, because a faulty actuator can be giving the expected responses ("ack" or "ok", etc) but in reality the door is not closing or the heat is not turning on. This in turn may be not a problem of the actuator in the smart device; it could be some physical problem with the door or the energy resources for the heating. For the IT system it is impossible to differentiate: it is possible to add some sensors to monitor the physical world more closely, to see if "the door is trying to close" but an obstacle is not permitting it, etc, but all those precautions have a limit. The cyber-physical system as a whole has a problem, this can be clear, but to perfectly diagnose the problem may be too costly or even impossible. In our model, the "service" as such becomes untrusted, independently of the root cause for the problem, if IT-based or of physical nature irrespectively.

In order to observe the actuators and the physical process in the background it is also necessary (or at least, very favourable) to observe the commands that are sent to the service. In other words: the observer should sit as a middle-man or proxy in the interface between the services or users using the service and the service itself. There are other technical solutions for this: a simple alternative is that the services using the observed service are advised to send information to the observer. A preferred solution is to create a new "proxy service" that should be used instead of the original one, as a wrapper. This construction is closely related to several commonly used patterns in Service Engineering (for other particular purposes), called wrapper, adapter, or facade [Gamma 95], but it works in streaming mode instead of the more conventional RPC (remote procedure calls). In our case, the wrapper-service intercepts the commands to the original service and on the one hand forwards them to the original service, and on the other hand sends the necessary information to the observer.

The observer has one or several fault models that would explain the different errors in the physical devices, sensors and actuators. It may also have attacker models that describe what would be the typical cyber-attacks that an attacker would perform. In case of uncertainty, but under the suspicion that there is a problem, an observer can try different actuator commands (which are known to be safe) and try to estimate which of the fault models is correct.

2.3.2.7 Observers of a Service with systematic errors

Consider the following two situations that often occur in reality, if no measures are taken:

- A service produces a data stream which, as a whole is correct and can be trusted, but contains some single data values that are "outliers" and for most purposes (but perhaps not for all) those values should be discarded.
- A service, say on a RERUM device, produces a data stream with some type of systematic errors, transient of steady-state. The error may be for instance some superposed white noise, bursts of erroneous values, systematic deviations of correct measurements, etc. For instance, it may be recognized (by a RERUM observer, comparing with close-by values and using model-based diagnostics) that a thermometer is producing systematically values 2° below the correct ones.

In those cases, we do not propose that the RERUM platform by itself should filter the data streams, independently of the purpose and context, but rather to allow a standard methodology and tools for constructing "wrapper-services", which may be seen as "corrected versions" of the original service, and which filter the bad data elements or correct the systematic errors. Those wrapper services have been constructed exemplarily using CLIPS rules, filtering the data in a way similar to, say, Kalman filters (see [May 79]).

2.3.2.8 Observers embedded in a Service

A service itself may estimate the trust rating of its output values, depending on the trust of the input values. The trust rating of the output values is provided to the reputation engine, as in the standard case, but it may also be provided to the consumers of the output stream, so that they may use this information without asking the reputation engine.

In Figure 4, a Service is depicted which contains an embedded Observer. In this example, the Service uses two streams, say x and y and generates a service with a data stream z. The Observer generates trust-ratings for x and z while the stream y was not evaluated, it was only used to obtain coherence conditions for x.

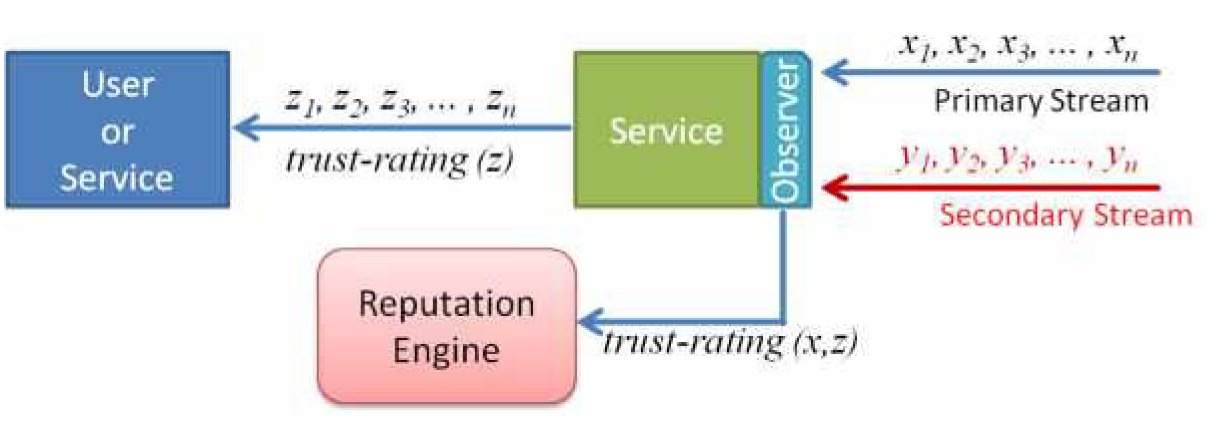


Figure 4: An observer embedded in a service

Typically an observer is "close" to the sensors or services, for instance in the Gateway of a RERUM-Intranet, or may be a part of the service that uses the streams of data to generate added-value services based on the streams.

2.3.2.9 Reputation alerts

The following types of alerts are required to be supported:

- INT_UNRELIABLE_DATA (Internal unreliable data: This alert should be generated when the reputation engine decides that incoming data from a device in a local installation (a RERUM device in RERUM installations) is not reliable and the service generating it had any of their reputation status as ‘enabled’. That is, data coming from fully disabled services cannot trigger this alert.

- DEC_INT_DEV_REPTN (Decrease Internal Device Reputation): This alert should be generated when the reputation engine decides to reduce any of the Reputation Evaluations of a given device of a local installation (a RERUM device in RERUM installations).
- DEC_ INT_SRV_REPTN (Decrease Internal Service Reputation): This alert should be generated when the reputation engine decides to reduce any of the Reputation Evaluations of a given service of a local installation (a RERUM service in RERUM installations).
- EXT_UNRELIABLE_DATA (External Unreliable data): This alert should be generated when the reputation engine decides that incoming data from an external service that is currently enabled is not reliable.
- DEC_EXT_SRV_REPTN (Decrease External Service Reputation): This alert should be generated when the reputation engine decides to reduce any of the Reputation Evaluations of an external service
- INC_EXT_SRV_REPTN (Increase External Service Reputation): This alert should be generated when the reputation engine decides to increase any of the Reputation Evaluations of an external service. Though it might seem this alert could never be generated because of disabled services being locked, enabled services could still increase their reputations, and high reputations on many purposes might potentially result on enabling a service that was previously disables for a given purpose
- INT_RELIABLE_DATA (Internal Reliable Data): This alert should be generated when the reputation engine decides that incoming data from a device in a local installation (a RERUM device in RERUM installations) is considered to be reliable and the service generating it had a reputation status of disabled. That is, data coming from fully enabled services cannot trigger this alert.
- INC _INT_DEV_REPTN (Increase Internal Device Reputation): This alert should be generated when the reputation engine decides to reduce the Reputation Evaluation of a given device of a local installation (a RERUM device in RERUM installations)
- INC_INT_SRV_REPTN (Increase Internal Service Reputation): This alert should be generated when the reputation engine decides to increase the Reputation Evaluation of a given service of a local installation (a RERUM service in RERUM installations)

The content of a Reputation Alert is:

For the alerts INT_UNRELIABLE_DATA and INT_RELIABLE_DATA:

- Type Name of the event
- Object Id of the generic object corresponding to the stream producing the data, which originally came in the incoming data event
- The total number of measurements produced for that stream of data, which will be used by the Reacting Model to calculate the number of measures taken from the last alert
- The percentage of deviation of the value from the acceptable range
- The set of purposes that this data is produced for, which originally came in the incoming data event

For the alert EXT_UNRELIABLE_DATA:

- Type Name of the event
- The url corresponding to the service producing the data, which originally came in the incoming data event
- The total number of measurements produced for that stream of data, which will be used by the Reacting Model to calculate the number of measures taken from the last alert
- The percentage of deviation of the value from the acceptable range
- The set of purposes that this data is asked for, which originally came in the incoming data event

For the alerts INC_EXT_SRV_RPTN and DEC_EXT_SRV_REPTN:

- Type Name of the event
- The url corresponding to the service producing the data, which originally came in the incoming data event
- The total number of measurements produced for that stream of data, which will be used by the Reacting Model to calculate the number of measures taken from the last alert
- The percentage of deviation of the value from the acceptable range
- The Reputation Evaluations corresponding to the service producing the data

For the alerts DEC_INT_DV_REPTN, DEC_ INT_SRV_REPTN, INC _INT_DEV_REPTN and INC_INT_SRV_REPTN:

- Type Name of the event
- Object Id of the generic object corresponding to the stream producing the data, which originally came in the incoming data event
- The total number of measurements produced for that service or device which will be used by the Reacting Model to calculate the number of measures taken from the last alert
- The percentage of deviation of the value from the acceptable range
- The Reputation Evaluations corresponding to the service producing the data

Note: It could seem a few strange to include percentages of deviation instead of values read in the Reputation Alerts, but including the real data could potentially lead to a privacy leak. Having into account that the Reacting Model should not decide based in the real data but in the order of the deviation, the percentage of the deviation should be enough.

2.3.2.10 External reactions

The reactions that can be produced by the Trust Evaluation Model and their rationale are explained in detail in Section 2.5. This section includes a little summary of their kind for the purpose of explaining the overall structure of the model.

The kinds of reactions are, in order of complexity, the following ones:

- Alert Logging: The most basic kind of reaction. It consists of storing the alert in a persistent store so It can be queried later by the administrator, who might not be necessarily be a human being (for instance a program analysing the logs for detecting attacks);
- Warning operations: Consist in warning an administrator, which, again, might not necessarily be a human being, about the alert received;
- Modify operations on the external environment: Reputation Alerts used jointly with a correlation engine may typically result in altering the environment of the system, affecting the evaluation of the reputation or even the collection of data. Hence, the following operations on the environment are foreseen:
    - Enable or disable the collection of data from a given GO
    - Alter the reputation of a given GO
- Execute external operation: This is a generic procedure for executing any kind of dynamic procedure based on the information coming from the alert. This kind of dynamic procedure is essential for an IoT environment, where the dynamic nature of the services offered makes impossible to define in advance all possible reactions because the services themselves are possibly not known in advance.

In any case, all reactions types have a common argument, which is the Alert received. With this argument, the reaction can access the information the details of the Alert and act accordingly.

2.3.3 Operations to be executed

This section explains the operations that the model can perform to interact with the rest of Software components of the system, which are: The GVOs collecting the data, the context of the System and the software components implementing the reactions. The operations to be executed are:

- Receive service and device data from the Data Events: Feed the Trust Observers with the information contained in the Data Events arriving in the Data Input Stream;
- Receive context data from the Context Input Stream and use them in the process of calculating the Trust ratings and Reputation Evaluations;
- Raise reputation alert with the results of the Reputation Evaluation and use it to generate proper reactions
- Update context data from the Context Output Stream
- Execute proper reactions, including feeding the authorization components with the result of the Reputation Evaluations

2.4 Trust Evaluation Model

As already mentioned, our goal is to create a lightweight trust evaluation system based on plausibility analysis of received data. "Lightweight" means mostly that it works on streaming mode, receiving data values from one or several related streams, keeping a small "state", which is updated regularly, and which is used for the plausibility verification. To do this, what is required is twofold:

- Define adaptive statistics for creating adaptive confidence Intervals, quantiles. The statistics should be updated on the fly, in a similar way, say, than the well-known use of exponential smoothing, a rule of thumb that is attributed to Poisson and has been extensively used in signal processing; and
- Using those adaptive statistics, we need to define rules that determine if, for instance, a given "jump" in the data values is to be considered as abnormal or if a set of values is coherent. This is the essence of the next few subsections.

2.4.1 Trust ratings

The *stream Observers* evaluate the *trust-ratings* for the streams, depending on the observed values and based on *rules* that identify anomalous behaviour, often due to abnormal changes of the values in the stream. For scalability purposes, it is more effective and more efficient (at least, in terms of communication overhead) if the observers are decentralized, and in this way all possible data streams could be observed, if required: if the data is processed close to where it is generated, less amount of communication is required. It is possible to have *a single* centralized observer, but probably it will not be able to rate all streams: there are some streams that it will not see because they are behind some aggregating Service or Gateway, or because they have a short communication range.

The trust-ratings are indicators that will be aggregated and/or further analysed. The final purpose is to identify possible faults in the sensors or other error conditions on them or in services that generate the streams. The rules will "fire" when values are outside of certain expected intervals or when relatively high jumps happen or when values are encountered that contradict some consistency conditions (for instance; in winter if the heating is active and properly working then the indoor temperature should be higher than the outside one).

In the following Figure three different observers produce trust-ratings for the streams:

$$x_1,\ x_2, x_3, \dots\ x_n$$
$$y_1,\ y_2,\ y_3, \dots\ y_n$$
$$z_1,\ z_2,\ z_3, \dots\ z_n.$$

In this example, Observer 1 produces ratings for the z stream, the second one for all three streams x, y, z and the third one for the x stream.

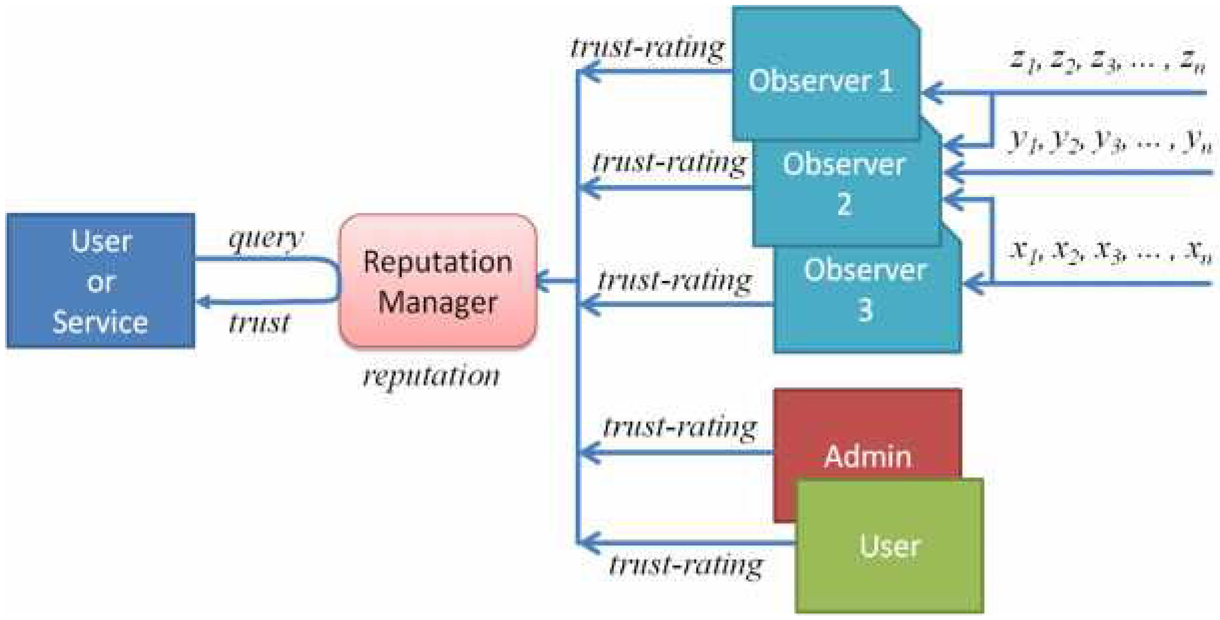


Figure 5: Trust management system

Besides observers, whose technical functionality will be discussed below in more detail, administrators or other privileged (trusted) users are allowed to also introduce trust ratings to streams as shown in Figure 5 depending on conditions they see or on external information (like the discovered vulnerabilities in the types of HW or SW used to generate the streams). In principle, also trusted (and with a high degree of assurance also trustworthy) "observers" outside of the RERUM instance and administration domain (and outside of our scope) could contribute trust-ratings, but this is not easy to manage securely. It is out of scope what information or algorithms the trusted users or administrators use to decide the values they want to enter as their trust ratings to the Reputation Engine.

Each observer can only produce trust ratings for the streams it sees, but it does not need to produce trust ratings for all streams it sees: in some cases, when an observer wants to rate a stream, say x , it requires a secondary stream y in order to calculate consistency conditions. In this case the observer rates only the stream x but not y; we assume that another observer rates y.

Figure 4 shows an example of this and it is explained later.

2.4.2 Reputation

Each stream should be analysed by at least one observer, and each observer will watch one or several streams. Each observer produces a set of trust-ratings and other trust relevant numbers (for instance, to indicate the type of possible fault in the stream of values or parameters about environmental conditions, etc). Those results or different trust-ratings are collected and sent to a more central component, the *Reputation Engine* (or *Reputation Manager*) that aggregates the values to *reputation* values. The reputation values describe different aspects of reputation from different points of view. When a consumer (user or service) wants to query the reputation engine, in his query he will add rules describing the conditions used for calculating this value or for the conditions and context on which the value will be used. More details about the reputation rules are described in the Section 2.4.9.

2.4.3 Trust for Services

As we already mentioned, one of the main specific goals of the trust-management system for Internet of Things (IoT) applications or a RERUM platform is to allow users or other entities such as services to answer the question: “Should a user rely on a given service, sensor, data, or data stream?” Thus, one of the central purposes of a trust management system is to answer trust queries for a particular service.

In RERUM, we assume that the RERUM smart objects offer services that generate data streams for which the trust or reputation values are estimated. For now, the main difference between trust and reputation that concerns us, is that trust is a value that depends not only on the trustee, but also on the trustor itself, his preferences and context, while reputation depends only on the trustee and the measurement that different observers have performed of his behaviour. We formulate this by saying that trust has a subjective component (of the trustor).

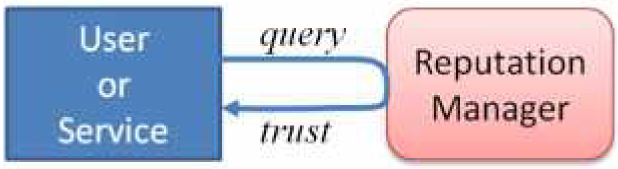


Figure 6: A trust query

In order to answer such queries the system has to support the simple functionality depicted in previous Figure the users or service can query a component of the trust management system about the trust of an IoT service or device or data. We call this component the **Reputation Manager**. The response to the trust-queries is often a particular trust value associated to a given entity for a certain user of service in a given situation. The entity (users or services) that queries the trust value of a service is a client of the Reputation Manager and it itself is a service consumer. For privacy reasons, only the entities that are authorized to call a service are able to query the trust of that service, this also corresponds to the "need-to-know" principle.

In order for the reputation manager to be able to answer the queries, he must collect so-called trust-ratings from other components of the trust management system that we call "**Observers"**. They are able to observe directly the data streams produced by devices and services and in favourable situations also test the services.

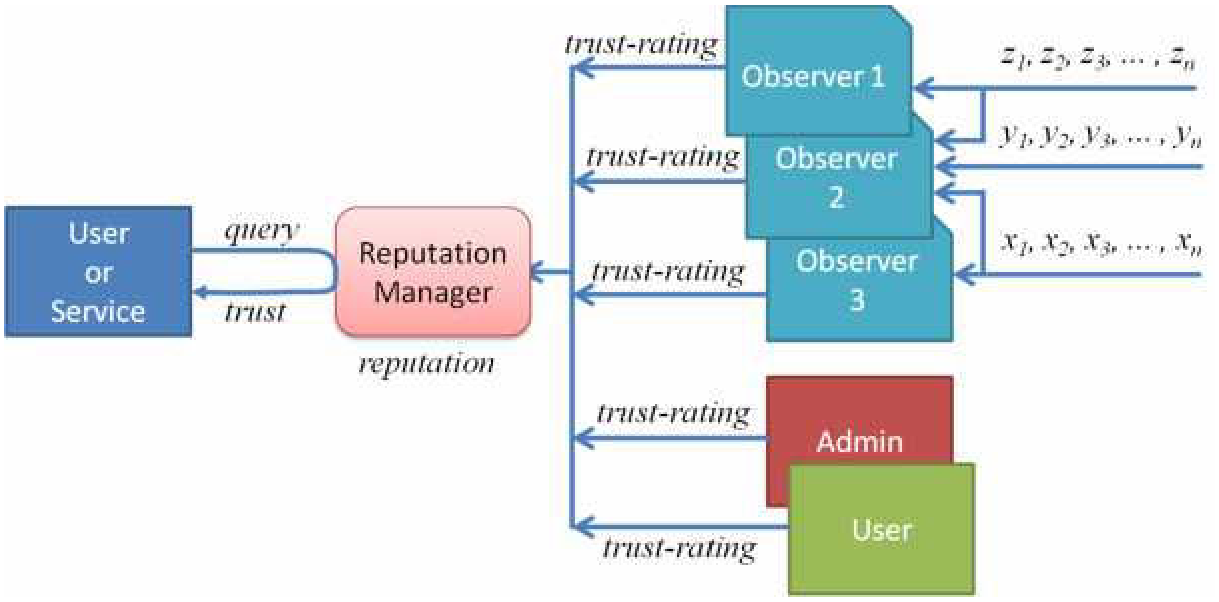


Figure 7: Trust management system: Observers

2.4.4 Trust for single Data Values

This section gives a proof-of-concept for observer rules providing a trust rating at data level and reputation engine to generate the reputation at service level (note this will be independent of the reaction mechanism). We expand the trust model with the rules corresponding to how the result of the integrity verification affects the evaluation of the trust of outgoing data.

Another specific goal of the trust management system after providing trust values for individual services (e.g. data streams) is to provide them as well for single data values. In RERUM we assume that Smart Objects can generate signed data values. Based on the verification outcome an observer can estimate a trust rating.

Signatures guarantee authenticity (created by sender), non-repudiation (sent by sender), and integrity (not changed during transit). RERUM defined these terms (non-reputation, authentication of origin and integrity) since deliverable D2.2 as follows: "The integrity protection, which RERUM requires, must allow the detection of integrity compromises, as opposed to mechanisms concerned with prevention [ISO_10181-6] of integrity breaches. The focus will be on protection against the following three violations: a) unauthorized data modification, b) unauthorized data deletion, and c) unauthorized data insertion [ISO_10181-6]".

In RERUM, as well as in many IoT paradigms, like the model used at the IETF, and in the context of signatures, data streams are encoded in JSON format. This is in contrast of seeing data as content of network packets or information exchanged between applications or users. This differentiation is similar to what was done in Deliverable D3.1.

To integrate the signature verification output into the decision of the observer we need to verify the signature. On conceptual level, the verification of a digital signature is based on asymmetric key cryptography, needs three inputs and produces one binary output. The **Verify algorithm** takes the message content as an input, the signature, and the public verification key. It outputs a decision as a bit $d \in \{true, false\}$ indicating the validity of the signature. It is important to note that the verification algorithm still misses the third input, the public signature verification key.

Therefore, we will take a brief look at the key management in RERUM: A public key must relate to a communication entity to be used as an authenticator for data origin. This relation can be direct or via some intermediate keys. Verification needs the public key of the entity that signed the data. RERUM envisions each device to have his own key. Additionally, a flat, but still hierarchical PKI (see Section 2.5 in Deliverable D3.1), will ease the management of known keys as follows:

- After the secure bootstrapping of the devices we assume that all devices have an individual key pair to start with. Furthermore, each manufacturer or bootstrapping manager has its own key and this is going to be the certificate authority in the one-level PKI.
- For each device's public key the manufacturer or device manager acts as a CA and will generate a certified public key[1].
- A certified key is produced by the CA by signing the public key together with additional data. By the mean of the signature the CA states that this is a device known to that CA.

Note, that the certified public key itself does not yet endorse any trust or trust-rating for a device or signed data. Nevertheless it is a basis to manage the verification of a number of devices instead of each device individually. It optionally eases configuration as well, but we will discuss this later. For key management it is advisable if devices capable of signing data are equipped with services (accessible via CoAP) that allow it to query it for its certificate and for its public verification key.

Summing up, a public key for each device within the network can be known per device in the following ways:

- the public key is recorded in a whitelist and is acquired out-of-band (e.g. bootstrapping process retains copies of public keys);
- the device can be queried for its certified key and the certification authorities public key is acquired out-of-band; and
- the certified public key is picked up from the flow of information and verified using a known certificate authorities' public key. If it verifies it is added to the whitelist of known device keys.

For the first option the bootstrapping authority is trusted to generate a correct list, therefore one could forgo the need to have those keys certified.

Finally, we now have the possibility to (1) obtain a public key. Then we can (2) verify the validity of key when either the key or a corresponding CA is whitelisted, and there is a process to (3) identify what public keys are known. The verification should use that list of know public keys only.

Next we look into the key selection process inside the observer. Here first the messages content and the signature-value are extracted from the stream of information. Every time a signed message is recognised, the corresponding key needs to be known to execute the signature's scheme verification algorithm. Each signature contains a short fingerprint of the public key that was used to generate it. This will be used as an index to a list of keys. We denote the list of keys that are known to the observer as list of known public keys for devices.

We envision filling the list with the public keys of devices that shall be recognised as known to the observer either static or dynamic.

[1] We do not call this a certificate to avoid applying overly complex structures/concepts like X509 public key certificates[Q] to the world of IoT devices.

Static method: With the static method the list is filled with all necessary entries during the configuration of the observer. The list is hence fixed and it requires communication with the observer to change the entries (e.g. add new known devices when added to the network or remove devices no longer known to the deployment).

One simplistic option to do this is if devices frequently broadcast their keys to be added to the public key-lists of potential recipients (see resurrecting duckling policy for public key free observers by Frank Stajano [Stajano 01]).

Dynamic method: As a static method requires manual configuration, which might be unsuitable in some environments, RERUM has designed a second mode of operation. The dynamic method works only under some additional assumptions about the information contained in the stream, and it needs an additional list. This requires more logic, but eases the configuration and operation of observers in return. It operates as follows:

Apart from signed messages, the stream of information is also monitored for certified public keys. Every time such information is detected it is extracted and fed into a process we termed certified public key verification. This process will check if the signature that is over a device's public key is valid with respect to one key that is in the static list of known public keys for CAs. Most certainly, also the dynamic method needs to have some statically assigned root of trust, e.g. this is the public key of the RERUM security centre or the device manufacturer.

If the certified public key verification is valid, then the device's key contained in the certified key message is known to that CA and thus by implication also assumed to be known to the observer. Hence, the public key of that device is added, with the fingerprint as an index, to the list of known public keys for devices. It is essential to note that the device's key is not added to the list of known public keys for CAs. Otherwise, the device could introduce new devices as known to the observer by issuing certified public keys with the device's public key. This is not wanted as only defined parties shall be able to act as CA's.

The further assumption is that for newly joining devices there is some exchange of certified keys. Therefore the relevant input data for an execution of the verification algorithm shall be present. We can now apply the Verify algorithm previously defined on the signed data using the public key from the list of known device-keys. The output obtained for standard signatures, like ECDSA, is a binary decision: *"0"* for an invalid signature or any other errors and *"1"* for a successful verification.

Next we expand the clips rules to compute a trust rating. For now we assume a static keylist locally stored at the observer with one key per device. The integrity observer is sketched in Figure 8.

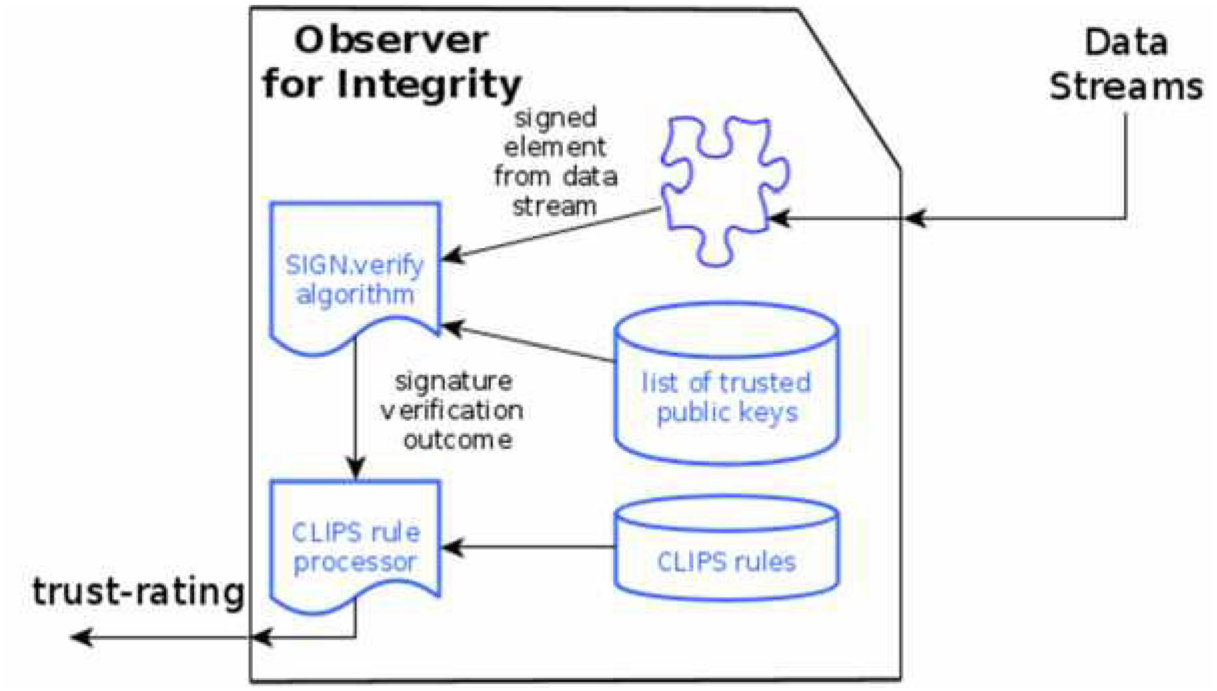


Figure 8: Integrity-observer inner working

Here we provide a simple example as a proof-of-concept for a corresponding **CLIPS rule:**

- First the signature of the value in the JSON packet is verified,
- then a binary trust rating is calculated,
- if the current signature and all previously observed signatures are correct,
- then the message can be trusted.

We assume that a correct signature can be trusted. A value with a broken signature receives "zero" trust and values without a signature are treated likewise. This is to avoid that an attacker can gain an advantage by not signing a modified value at all.

The rule can be easily extended with a recover mechanism or to address special conditions like missing values. One way to deal with missing value is to define an acceptable loss-rate, which can be calculated with (signed values lost)/(signed values send). An acceptable loss rate could be for example defined as *<0.02*. A recover could be initiated for example after receiving 50 correctly signed messages with a loss rate of *<0.01*.

The integration of the integrity observers into the model is shown in Figure 9

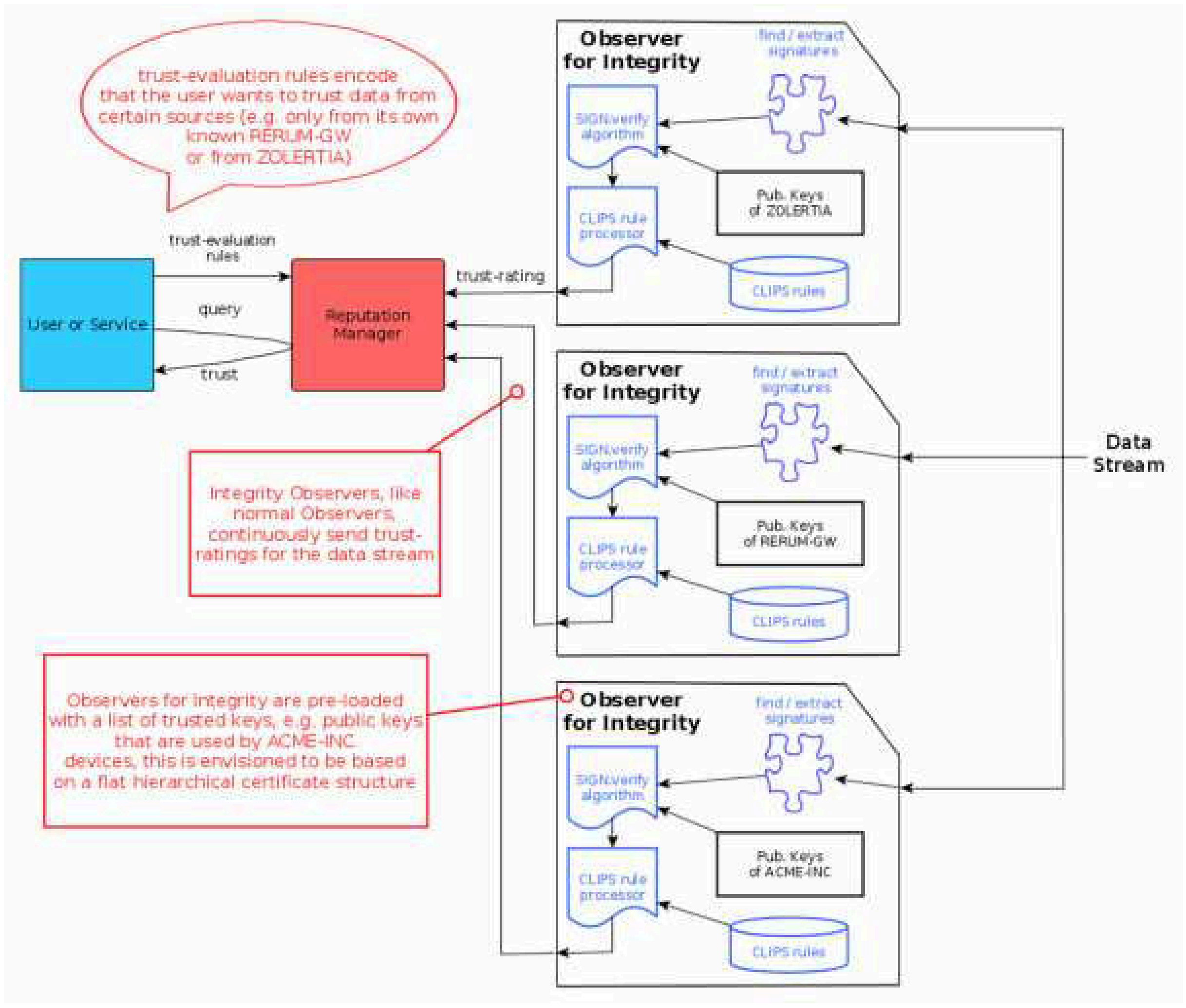

trust-evaluation rules encode that the user wants to trust data from certain sources (e.g. only from its own known RERUM-GW or from ZOLERTIA)
trust-evaluation rules
query
trust
User or Service
Reputation Manager
trust-rating
Integrity Observers, like normal Observers, continuously send trust-ratings for the data stream
Observers for integrity are pre-loaded with a list of trusted keys, e.g. public keys that are used by ACME-INC devices, this is envisioned to be based on a flat hierarchical certificate structure
Observer for Integrity
find / extract signatures
SIGN.verify algorithm
CLIPS rule processor
Pub. Keys of ZOLERTIA
Pub. Keys of RERUM-GW
Pub. Keys of ACME-INC
CLIPS rules
Data Stream

Figure 9: Integration of observers

More advanced concepts could focus on reducing the overhead to save on power consumption and/or adding security features like provided by Malleable Signatures. To reduce the overhead RERUM suggests to sign only average values send by the device as part of the data stream every x-hundred messages and verified by the observer.

Optimised signing: Signing every message is very costly in terms of required resources (see RERUM Deliverable D5.3). This goes in terms of more computing resources, more memory, communication overhead and as well energy consumption in general; all these being scarce resources on IoT devices.

In optimised signing we send messages with values normally unsigned. The trustee then calculates an average value every X messages (e.g. 200). The frequency is set by the trustee and depends on the service in general (e.g. timely response); but also on the variance expected (larger variance -> higher frequency). Only this average value gets signed by the trustee and is sent as a special message towards the observer.

The observer then calculates the average value of the preceding *X* messages from the trustee. Hereinafter the observer compares with signed value received with the locally calculated value. The trust level is then adjusted accordingly. Lost messages will be treated likewise to failed signatures.

2.4.5 Calculating Statistics in Streaming Mode

In this subsection, as well an in the few next ones, we are interested in defining a basic set of statistics and a set of rules that use the statistics to detect abnormal behaviour. There are many different statistics that can be constructed (using for instance different parameters that determine the weighting of the memory, vs the plasticity of adapting to new values) and there are also many types of rules that can be built.

For scalability and performance reasons, observers do not store the single past values of any stream that is being monitored or tested; but rather calculate statistical information about a stream, store it and update those statistics regularly. The term “*streaming-mode*” is often used to characterize this type of calculation: the data stream is observed in sequence, i.e., reading one value of the data stream at each given time (or a small amount of values), and consuming this value (or values) immediately, without storing it, without allocating dynamically more resources (processing power or memory) and without blocking the arrival of new measured values. In our proposed system, the observers therefore work on streaming-mode. Of course a small, fixed, predefined amount of "state information", including the values of related sensors will be kept, but continuously overwritten. This increases the efficiency and performance of the observers.

Each time a new value is received, the observer calculates data-stream statistics and checks certain predefined conditions. The conditions in the rules are as simple as comparing the current value or the aggregated statistical value of the stream with parameters related to the sensor or environment models or with values of related streams. In some cases, the observer may contain a model-based diagnostics procedure identifying errors in the data streams and faults in the physical sensors or actuators. In that case, the calculations are normally more complex, but still possible. If the system finds any indicator of an abnormal behaviour, it takes appropriate actions defined within the trust management system such as modifying the reputation of a service or creating an alert.

It is well known how to define streaming mode variants of the mean. But, as far the authors are aware, the method for calculating other statistics (like the quantiles, confidence intervals, etc) is new.

2.4.5.1 Calculating averages of streams

To analyze data streams it is necessary to keep some information about past behavior. The simplest example of this is to keep an average value (mean) of a given data stream. If the system changes over time, it may have a steady-state behavior or it may, more probably, continuously change, as the system expands or as new sensor methodology is introduced, etc. Calculating the averages over the whole lifecycle of the system is not convenient because as the system evolves the importance of past that is not relevant anymore increases. The average must be "moving", adapting its values as the system evolves.

If one wants to calculate precisely the sliding average of the past 100 values (or any number of values) it is necessary to keep in storage all those 100 values in order to know, each time, which is the oldest value in that window of values.

We propose to *estimate* adaptively, without consuming too much storage, the "moving" (or "sliding") average of a stream, as well as other useful statistics, using the so called exponential smoothing method: Given some raw data values (say, a stream of sensor values or service values) x_1, x_2, x_3, ... x_n, the moving average of those values is estimated as

$$Me \quad _t(x,\alpha) = \bar{x}_t = \alpha\, x_t + (1-\alpha)\, \bar{x}_{(t-1)}$$

where $0 < \alpha < 1$ is the smoothing factor, usually of the order of 0.05 or smaller. The number α is related to the "length" of the sliding window that is relevant for calculating the average. To make that more precise, let us first observe that the mean at time t is the weighted average of x_t. and the previous values $x_{t-1}, x_{t-2}, x_{t-3}, \dots$:

$$\bar{x}_t = x_t + \alpha(1-\alpha)\, x_{(t-1)} + \alpha(1-\alpha)^2\, x_{(t-2)} + \dots + \alpha(1-\alpha)^{(n-1)}\, x_{(t-n+1)} + (1-\alpha)^n \bar{x}_{(t-n)}$$

Now, if one wants that the window $[\,(t-n+1),\ t\,]$ has, say β of the weight (say $\beta = 0.9 \ = 90\%$) then the relation between n and α is given by:

$$n = \frac{\log(1-\beta)}{\log(1-\alpha)}$$

or equivalently,

$$\alpha = 1 - (1-\beta)^{1/n}$$

The equation is important to calibrate the value of α: if one wants that the last 2 hours account for 90% of the weight and the stream has one value each 10 seconds (360 per hour, 720 in 2 hours), then α should be chosen as $\alpha = 1 - (1-0.9)^{1/720} = ??$.

Notice that if x_t is a stream, then also $Me\quad(x,n) = \bar{x}_t$ is a stream and it is possible to calculate statistics based on them. The same is true for other statistics calculated on streams.

This allows estimating also a sliding variance:

$$Var(x)_t = Mean_t\,((x-\bar{x})^2, n) = \alpha\,(x_t - \bar{x}_t)^2 + (1-\alpha)\,Var(x)_{(t-1)}$$

The maximum of x_t is also easy to calculate:

$$Max(x)_t = \max(\,x_t\,, Max(x)_{(t-1)})$$

and similarly for the minimum.

2.4.5.2 Calculating the relative size of "jumps"

Given a stream $x_1,\ x_2,\ x_3, \dots\ x_n$ of values, the jump at time *t* is

$$\gamma_t = x_t - x_{(t-1)}$$

or, alternatively (with less "noise")

$$\delta_t = x_t - \bar{x}_{(t-1)}$$

But notice that γ and δ calculate two different "types of jumps".

We are interested in detecting when a jump is too high, which could be an indicator of a fault.

2.4.5.3 Calculating the Density

If p is a predicate on the data type of the values of a stream $x_1,\ x_2,\ x_3, \dots\ x_n$ (that is, it is Boolean-valued function on the values x_t), we will need to estimate the proportion of values for which $p(x_i)$ is has the value "1" (equivalent to p being "true"). This is called the density of p in the stream x. The estimator is

$$Dens_t(p.x,\alpha) = Me\quad_t(p(x),\alpha) = \alpha\, p(x_t) + (1-\alpha)Dens_{(t-1)}(p.x,\alpha)$$

If $p(x_i) = 1$ is rather often, then we need a long history on the past to estimate correctly the density. The same is true if $p(x_i) = 1$ rather often. In other words, α must be relatively small. It should be true that n should be at least of the order of

$$\frac{200}{Min\,(Dens_t(p.x,\alpha),1 - Dens_t(p.x,\alpha))}$$

2.4.5.4 Calculating Quantiles

It is not straightforward to estimate the median or quantiles of arbitrary data streams. If the stream fluctuates without well-defined time constants, it is not easy to find the "correct" value of the sliding parameter α, or equivalently the time constant parameter n. n can't be to large, because then the quantile estimation lacks plasticity and does not adapt adequately when the situation changes, but it can't be too small as to depend too much on just the very recent past (which may depend on random fluctuations or "noise"=. Also, since estimating a quantile implies estimating how many values are above and below a certain threshold, the corresponding inequality for estimating densities must be true:

$$n > \frac{200}{Min\,(\rho, 1 - \rho)}$$

We are primarily interested in calculating the quantiles for "jumps" which may fluctuate over time, but in more or less regular time periods (days and weeks, perhaps also years). It is possible to estimate, say, the 95% quantile of jumps for the past hours, or the past day or week and those values can be reliably be used to detect if a jump is "too high" or not. For streams of "jumps" it is often possible to calculate quantiles, depending on the distribution and its dynamics of the data stream, because some regularity assumptions about the variability of the dynamics of the data stream are met. If it is possible to find a number n larger than the number above, such that for the past $4n$ time steps, the stream is more or less in steady-state, then we can estimate the quantile.

If we are able to estimate a moving quantile $\Theta(x,\rho,\alpha)_t$ for a stream x, where ρ is the quantile level (say, a number between 0 and 1) and α is related to the window size n (which, in general may be much larger than the size of the windows to calculate averages), then we are able to determine if a value we see is comparatively too high or too low, compared to the statistics estimated for the past (of window size *n*)

2.4.6 Rules: Thresholds

If a value of a sensor or service data stream goes out of a certain range, a rule fires. There may be different ranges with different meanings (the larger a range, the stronger the alarm that is generated if an outlier is found). The rule, if it fires, will re-calculate the trust rating of the data stream (or equivalently, of the service) or the stream may be "marked" for further observation, etc.

In general, we will write τ_l^x and τ_h^x to represent the *lower* and *higher* values for thresholds for the stream x.

A possible way to calculate thresholds could be $[\overline{x_t} - Z * \sqrt{Var(x)_t},\ \overline{x_t} + Z * \sqrt{Var(x)_t}]$

2.4.7 Rules: Jumps

Even if the value does not move outside of the interval where it should usually remain, the magnitude of the jumps of values could be also used to detect possible malfunctioning. If a value "jumps" too much, this may be a symptom for a fault. For the purpose of detecting this type of events, we need to determine thresholds for the jump sizes. The thresholds are simply two values τ_l and $\tau_h$such that if $\delta_t > \tau_l$ or $\delta_t < \tau_h$, the jump is considered to be "too much" and a rule fires. The rule may modify the trust rating of the data stream (or equivalently, of the service) or may lead to further analysis. As a first approximation, we propose to use the quantiles of the jumps as threshold values: $\tau_t = \Theta(\delta, \rho, n)_t$

There are different sources of thresholds: static ones can be provided by the vendors of the sensors, or may relate to minimum and maximum reasonably expected values. The expected values may depend on the conditions of the environment (say, where the sensor is located). Dynamic ones may be learned from the past, including values for quantiles or may depend on current conditions of the environment (say, the temperature may depend on whether the window is (or was recently) open or on the status of the heating system.

2.4.8 Trust Rules

To understand why the trust may depend on the policies of the user or in the context or purpose of his query, let us briefly review some typical situations:

1. A certain restaurant may have a very good reputation among the people who regularly attend it, say single young people, but this information be irrelevant for conservative seniors.
2. A transportation service could be evaluated differently based on different criteria such as: punctuality, safety, overall stress, etc. The trust that a user may deposit on the safety of the service may also depend on the context of the query: Is he travelling to a suburb of a large city, or moving only in the centre? Is he travelling during daytime or on the evening? Is he alone or with his small children, or with his friends?
3. The reputation of a well-known specialized doctor may be high within the general public, but a certain patient may decide, when choosing a specialist, to trust the opinion of his own family doctor than the trust-ratings of the general public.

In RERUM terms the three situations may be abstracted as follows, respectively:

1. Given a set of trust-ratings for a service B, produced by observers O_1, O_2, ... O_n, the trust that A bay deposit on B depends on the characteristics (specifically: on the rules used to create the respective trust-ratings) of the individual observers.
2. The trust that A may deposit on B depends on the context of the query.
3. The trust of A on B may depend on the personal preferences of A.

Similar to the aforementioned real world examples and its factors, the IoT world encounters the same type of situations: the trust that A deposits on B may depend on the characteristics of the different available observers, on the context of his query, or on his personal preferences.

1. A service may have evaluated as offering timely data values (but not so precise) or very accurate ones (but a bit older).
2. Assume a person is leaving for work and sets an alarm when the bus is approaching, he may prefer that the alarm is timely as that the position of the bus is more accurate than say 200 meters.
3. A user may have a long-lasting relationship to a service provider, which he prefers to use for personal information, instead of using different providers.

2.4.9 CLIPS Rules

The basis of our proposed solution is the calculation of parametric and non-parametric statistics beyond the well-known adaptive mean estimation, including for instance adaptive estimations for standard deviation, quantiles, characteristic “jumps”, etc. We use those estimators to detect on-the-fly abnormal sensor (or network) behavior.

As a proof-of-concept, this section contains an excerpt of the formalization of the rules for generating alarms. We use the expert system CLIPS (see [CLIPS-RefMan-6.30-2015]) to demonstrate an implementation of many of the rules described in Sections 2.4.5 - 2.4.8. Our proof-of concept also uses python rues for numeric calculations, not shown here, used for instance to update the statistical estimators, to read and write the data streams (in our prototype implemented as UDP streams), and to generate synthetic test data.

This proof-of concept demonstrates the streaming mode processing and the simplicity of the calculations. Further, an interested reader, as soon as a user gets accustomed to the Lisp-like syntax, based on prefix notation, where for instance "x+y" is represented as "plus (x y)", may convince himself how straight-forward it is to create more interesting rules, using for instance confidence intervals, constructed using adaptive standard deviations instead of fixed values. Also it should be clear to him how expressive is the language, so that rules for many different types of purposes may be easily crafted.

2.4.9.1 A-priori conditions

The following rule generates an alarm if the observed values are outside of an interval [minA, maxA]:

```
;=======================
; Rule valC-mima
;=======================
(defrule   valC-mima   "checks   valC   (str   val)a-priori   boundary
conditions
 of each observer [ 0 < valC < 40 ]"
      (a-valC-mima   (obsN   ?obsN)(strN   ?strN)   (ruID   ?ruID)(minA
?minA)(maxA ?maxA))
      (a-str (strN ?strN)(valC ?valC) (timC ?timC))
      (test (or (< ?valC ?minA)  (> ?valC ?maxA)))
      =>
      (assert (bad ?strN ?obsN "valC-mima" ?ruID (getTime ?timC)))
      ;(printout  t  "Alert!  "?obsN"'s  "?strN"  values  are  abnormal"
crlf)
)
```

2.4.9.2 Standard-deviation (aveE-stdE)

The following rule compares the observed values with the observed estimated adaptive average. If the difference is larger than an adaptive threshold (zed x std). Here, std is an estimated adaptive standard average and zed is a parameter that determines the width of the confidence interval (corresponding to the z-value in the classic confidential interval $\overline{X} \pm Z\,\sigma_X$.

```
;=================
; Rule aveE-stdE
;=================
(defrule   aveE-stdE   "checks   if   aveE   [avgA-(zed)std   <   aveE   <
avgA+(zed)std]"
      (a-aveE-stdE (obsN ?obsN)(strN ?strN) (ruID ?ruID) (zed ?zed)
(aveA ?aavg))
      (a-str (strN ?strN) (aveE ?aveE)(stdE ?estd) (timC ?timC))
      (test (or (< ?aveE (- ?aavg (* ?zed ?estd)))
                  (> ?aveE (+ ?aavg (* ?zed ?estd)))
              ))
      =>
      (assert (bad ?strN ?obsN "aveE-stdE" ?ruID (getTime ?timC)))
      ;(printout t "Alert! "?obsN" "?strN"'s aveE is abnormal
      ;compared  to  a-priori  average  (+-)  "?zed"  times  the  ewm-
Standard Deviaton" crlf)
)
```

2.4.9.3 Historical-value (valC-valH/avgH)

Consider the following situation: the temperature for Munich is being measured outdoors, say, in a park. There are services that provide historic values corresponding to statistics of temperatures in this city over the last 50 years, during the different weeks of the year and during the different hours of the day. The fist of the following two rules compares the observed values to the statistical maximum and minimum values that have been observed in the past history. The second rule compares the current average to the average maximum and minimum of the historic data.

```
;======================
;Historical value
;=======================

(defrule valC-valH "checks str val (valC) [ min-xvalH < valC < max-
x-valH] in a particular geographical location geoL"
      (a-obs-str-rul (obsN ?obsN)(strN ?strN) (rulN "valC-valH"))
      (temp-hist (rec-high ?maxA) (rec-low ?minA) (timH ?timH) (geoL
?geoL))
      (a-str (strN ?strN) (valC ?valC) (timC ?timC)(geoL ?geoL))
      ;using python call getMonth function
      (test (and (or (< ?valC ?minA) (> ?valC ?maxA)) (= (getMonth
?timC) ?timH)))
      =>
      (assert (bad ?strN ?obsN "valC-valH" (getTime ?timC)))
      ;(printout t "Alert! "?obsN" "?strN"'s values are abnormal
compared to the historical
      ;temp values of timH: "(getTime ?timC)" geoL "?geoL crlf)
)

(defrule valC-aveH "checks str avg(avgS)
[ min-avgH < avgS < max-avgH]"
      (a-obs-str-rul (obsN ?obsN)(strN ?strN) (rulN "valC-aveH"))
      (temp-hist (avg-low ?minA) (avg-high ?maxA) (timH ?timH) (geoL
?geoL))
      (a-str (strN ?strN) (aveS ?avgS) (timC ?timC) (geoL ?geoL))
      ;using python call getMonth function
      (test (and (or(< ?avgS ?minA) (> ?avgS ?maxA))(= (getMonth
?timC) ?timH)))
      =>
      (assert (bad ?strN ?obsN "valC-aveH" (getTime ?timC)))
      ;(printout t "Alert! "?obsN" "?strN"'s average is abnormal
compared to the historical
      ;average of the timH: "(getTime ?timC)" geoL "?geoL crlf)
)
```

2.4.9.4 Reputation value (reputation engine)

The following rule calculates a reputation value as a weighted average of observed values.

```
(defrule reputation-upd " reputation rating"
      (declare (salience 10))
      ?obs <- (obs-truR (obsN ?obsN) (strN ?strN) (truR ?strTR)(timC
?timC))
      ?adm <- (adm-truR (obsN ?obsN) (truR ?obsTR) (timC ?timC))
      ?strT <- (str-rep     (timC  ?timC)(strN  ?strN)  (repV  ?repV)
(truR ?truR))
      =>
            (modify ?strT (repV (+ ?repV (* ?obsTR ?strTR)))(timC
?timC) (truR (+ ?truR ?obsTR)))
            (retract ?obs)
            (retract ?adm)
)

(defrule divide-truR
      (declare (salience 1))
      ?strT <- (str-rep (timC ?timC)(strN ?strN) (repV ?repV) (truR
?truR))
      (test (neq ?truR 0.0))
      =>
      (modify ?strT (repV (div ?repV ?truR)) (truR 0.0))
)

(defrule reputation-new "create a new reputation for the stream at a
particular time"
      (declare (salience 20))
      (obs-truR (obsN ?obsN) (strN ?strN) (truR ?strTR)(timC ?timC))
      (adm-truR (obsN ?obsN) (truR ?obsTR) (timC ?timC))
      =>
      (assert (str-rep (strN ?strN) (repV 0.0) (timC ?timC)))
)
```

2.5 Reacting Model

In a generic sense, a Reacting Model is a set of criteria that defines how a System is meant to react to a series of events. More specifically, in this document, our Reacting Model defines how an IoT oriented system reacts to a series of Reputation Alerts resulting from the Trust Evaluation Model. [KeWi06] already presented the concept of a Reacting Model that was able to interact with a Trust engine for Internet Applications.

However, the very nature of the IoT implies that the System can include any type of device connected and for any purpose. This makes it impossible to define in advance for a generic System which devices will be connected and its purpose [ITU-T Y.2060]. As the reactions to the behaviour of these devices will also depend on their nature and this cannot be known in advance, it is impossible to define in advance all these reactions. For instance, the reaction to a fused light bulb will not be the same as the reaction for a failing moderator of a nuclear plant.

For this reason, what this section explains is:

- A small set of reactions that can be applied to the result of any Trust Evaluation Model because they do not depend on the nature of the devices connected to the System but only on the evaluation results; and
- Instead of defining how any IoT System must react to any Reputation Alert from the Trust Evaluation Model, this section explains how to define and process these reactions in a way that can be supported by any IoT System.

Additionally, this section explains how a Reacting Model can interact with the rest of the System to improve the robustness of the whole System through the incorporation of the trust evaluation to the authorization of the requests.

In Summary, the structure of this section is the following:

- General IoT Reactions: Introduce a small set of reactions that can be applied to any IoT System, along with the concrete rules to define them
- Generic Reaction System: Study the different alternatives to specify how a System should react to Reputation Alerts and a generic mechanism to execute these reactions according to that
- Trustworthy Information Sharing: Explains how the Reacting Model can interact with the rest of the System to improve the robustness of the whole System
- Distributed Heterogeneous access control and profiles: Explains the different alternatives to incorporate the trust evaluation to the authorization of the requests using the Generic Execution Mechanisms

2.5.1 General IoT Reactions

As explained in the introduction, the very nature of the Internet makes impossible to define a generic mechanism that covers all possible reactions because the range of possible devices connected and the services they offer cannot be known in advance. But it is still possible to define some rules based on the result of the Trust Evaluation Model, because its interface is already defined by the specification of the Reputation Alert, which is the way that the Reaction Engine is triggered to be executed from the Reputation Engine.

In fact, in this section we identify various types of reaction that can be of interest for any IoT environment and introduce them in order of complexity. But actually there are so many possible reactions in the IoT as systems connected to the internet. For this reason, all the reactions included in this section must be considered to be possible options available for the System administrator to take them, instead of universal rules suitable for everywhere. Note the reactions provided are not novel per se. What is novel is the analysis of how they are relevant for a Reacting Model for the IoT

2.5.1.1 Logging alerts

This type of reaction is the most basic one, and consists simply in storing the information received in the Reputation Alert in some kind of log storage. This log is meant for the System Administrators being able to read the alerts and decide what measure to be taken on its own. Note that this log storage needs to be very easy to be queried or filtered or otherwise be accompanied with a query tool. The reason for this is if the System Administrator is meant to look for relevant alerts on its own, then he will need this query tool to locate the relevant ones.

Though this log storage could theoretically be a plain text file, in practice, it is worth considering taking a more versatile approach, such as a database, because they provide much more powerful tools and, what is more important, the information in the alert may refer to entities that have complex n to m relationships with each other. For instance, if an alert refers to a RERUM service which has several RERUM devices associated, storing that relationship in a text file could be very cumbersome.

Even if the logs are stored in a database, it is advisable that the system provides a specific tool for querying the logs and marks the alerts as seen, so the administrator can focus only in those alerts that he has not seen before.

Regarding the information to be included in the log, it should contain:

- All the information included in the alert
- An identifier for the alert
- A field for marking whether this alert has been dealt with or not

Logging alerts is useful especially for those systems that contain a number of devices so big that the number of alerts is expected to be very high and potentially overwhelm the System Administrator. In these cases the System Administrators often prefer to take a look to the alerts on its own.

2.5.1.2 Warning administrators

As mentioned, in very big systems the number of devices can be so big that could potentially overwhelm the system administrator. But what happens if a given service is considered to be very important or if the number of devices is not very big? In these cases it makes sense to warn the System Administrator or even a given final user directly. The mechanisms for these warnings are debatable. One possible option is to send an email to an email address that is specific for this purpose and meant to be read only by the System Administrator. Another option could be to have a different email address assigned to each kind of alert that could potentially address a final user. Or it is even feasible to send a message to some kind of corporate social network for the group of System administrators to take them. The concrete warning medium is not the aim of this section. What is important to make clear is that the warning should either be accompanied by the information of the alert or preferably include a reference to the alert in the log storage for alerts. The ideal case would be the latter because it would offer the best of logging alerts and warning administrators and it is also safer because it avoids providing the details of the alarm in the message, which is subject to be intercepted.

2.5.1.3 Disabling services

As it is possible to define some rules based on the result of the Trust Evaluation Model, it is also possible to define reactions to the Reputation Evaluation. More specifically, it is possible to enable or disable the delivery of data or a service based on its overall reputation value and the configuration of the system. For instance, access to untrusted services could be rejected after it has been evaluated to have a low Global Reputation Evaluation. But the concept of disabling the access to a resource is quite debatable itself and should therefore be optional. For this reason, System Administrators should be allowed to take the decision on whether to activate this or not. This decision would be implemented by storing that decision in a configuration store, which can be whatever persistent store that the System uses to store its configuration, such as a database table or a configuration file. A single numeric value in that configuration store with a value of 0 can mean that no disabling is used or any other positive value for indicating a threshold for the Reputation Evaluation could allow the System Administrator to indicate whether to use this feature and what reputation levels are low or high enough to warn about their values accordingly.

Section 2.5.4 shows the different alternatives to use this information in the authorization process. In very short, there should be disabling policies with a meaning such as:

```
IF (alert.global_reputation_evaluation < configuration.global_threshold)
THEN REJECT
```

That is, for disabling services, there would be a policy that will check that the Global Reputation for that service is less than the global threshold for that reputation and service. For enabling services, this policy will simply be removed, causing the global reputation being omitted and hence the service being granted except for the rest of the access checks.

It is important to note that this mechanism is not talking about disabling the production of the data itself, but only the access to the data consumed in the System. This has two consequences:

1. The data will keep on being produced by the devices. That is, actually devices would never become disabled directly, unless they were exposed directly as services, in which case what would be disabled would be the service exposing the device and
2. As a result of 1 the data will still be available for the Trust Evaluation Model to keep on analszing them and consequently create Data Alarms from them. This feature will allow that a disabled service can become enabled later depending on its behaviour.

2.5.1.4 Warning Requesters on Reputation Evaluation

We initially said that the interface with the Trust Evaluation Model is already defined, but there is still one issue regarding that: According to the requirements, the Trust Evaluation Model does not produce a single reputation value, but a set of reputation values, one per each external evaluation criterion defined in the system. Then, how is it possible to define the reactions for a set of evaluation criteria potentially unknown? The answer to this question is though the set of purposes cannot be known in advance, the Reputation Alert will contain the names of the evaluation criteria provided by the Reputation Evaluator. Hence, it will be possible to update the configuration store with a threshold value for each purpose together with their purpose name to let the administrator update it later.

Section 2.5.4 shows the different alternatives to use this information in the authorization process. In very short, there should be policies that produce concrete advice based on the following logic criteria:

For each type of evaluation criterion:

```
IF (alert.reputation(type) < configuration. Threshold(type))
THEN AddAdvice("reputation(type) < Threshold(type)")
```

Nevertheless, disabling the service for each evaluation criterion is even more arguable than disabling the whole service. That is, if the service has an overall Reputation Evaluation extremely low, it seems reasonable to disable it, but in the case that the Reputation Evaluation is very low for a given purpose but not so low for a different purpose, disabling the service only in some situations seems more arguable. As the reason for providing different evaluation criteria is to provide an evaluation that fits the needs of the invoker, the invoker should have the chance to decide on its own whether to take into account the Reputation Evaluation and for what criterion. Hence, another common reaction is letting this evaluation to be available to the invoker. Http already provides a mean to do this since http 1.1 [https://tools.ietf.org/html/rfc2616#section-14.46], which includes warning headers that allow this mechanism. Then the idea would be to produce a warning for each evaluating criteria whose value is lower than the threshold specified in the configuration store.

Regarding the structure of the warning itself, it should follow the structure specified in http1, that is:

```
Warning    = "Warning" ":" 1#warning-value
warning-value = warn-code SP warn-agent SP warn-text
                                          [SP warn-date]
warn-code  = 3DIGIT
warn-agent = ( host [ ":" port ] ) | pseudonym
             ; the name or pseudonym of the server adding
             ; the Warning header, for use in debugging
warn-text  = quoted-string
warn-date  = <"> HTTP-date <">
```

where warn-code is specified in [https://tools.ietf.org/html/rfc2616#section-14.46]. As there is currently no specific warning code for reputation purposes, it will be necessary to use the miscellaneous one 299, which allows the client to obtain the warning without necessarily showing it to the user.

For instance, if we have the following evaluation criteria and configuration:

Reputation Evaluation	Threshold criterion
Accuracy evaluation = 2	Accuracy criterion = 4
Availability evaluation = 9	Availability criterion = 4
Response time evaluation = 6	Response time criterion = 7

In this example, the accuracy evaluation and the response time evaluation are lower than their respective threshold criteria, but the availability evaluation is not. For this reason, one warning should be generated for each of two failing evaluations. Hence, for this example, the warnings generated should look like:

Warning: 299 [2]autz_host Accuracy Evaluation = 2

Warning: 299 autz_host Response time evaluation = 6

Both of these warnings must be included in http headers named “Warning”

[2] 299 Is the code defined in the http standard to mean a generic warning

For the details of how this is specifically achieved for RERUM, please read Section 3.8.

Once the mechanisms to access the information needed for taking the decision are defined, it is still necessary to define the rules that state how to activate or deactivate the services (recall that devices are exposed through services). In a first approach, it could possibly be enough to state something similar to:

```
ON    dataAlert
IF    dataAlert.Reputation_for_Evaluation_Criterion_i <=
     Threshold_for_Evaluation_criterion_i AND
     dataAlert.state_of_object = enabled
THEN execute_action(disable, dataAlert)
```

And, similarly, for enabling a device:

```
ON    dataAlert
IF    dataAlert  .Reputation_for_purpose_i  >  Threshold_for_purpose_i
AND
      dataAlert.state_of_object = disabled
THEN execute_action(enable, dataAlert)
```

The internal details about how execute_action works will be explained in Section 2.6.2. But that approach has the problem of being too rigid: It is normally not a good idea to disable a service or device for a single bad datum. Common enabling or disabling criteria often takes into account several failures / successes in a row or a percentage of success / failure in a given data window. For that, it is necessary to include a new concept that has not been introduced in this document yet, e.g. the correlation of Reputation Alerts.

The correlation of Reputation Alerts consists in creating new alerts from the information of previous alerts, possibly using internal memory to store intermediate data related to them, and to use those new derived alerts to create the rules. In our concrete case, we want to correlate failure and success events in the following way:

```
ON    dataAlert
IF    dataAlert.Reputation_for_purpose_i  <=  Threshold_for_purpose_i
AND
      dataAlert.state_of_object = enabled
THEN  raise_disable_warning(dataAlert, 1)


ON    disableWarning
IF    disable_warning.counter >= DisablingThreshold_for_purpose_i
THEN  executeAction(disable, disable_warning)
```

That is, we combine two different events to take into account not only the threshold to consider a given object as a candidate for disabling but also a counter for the number of failures considered enough to disable a service.

The opposite rule for enabling a service would be nearly but not completely identical:

```
ON    dataAlert
IF    dataAlert.Reputation_for_purpose_i   >   Threshold_for_purpose_i
AND
      dataAlert.state_of_object = disabled
THEN  raise_enable_warning(dataAlert, 1)


ON    enableWarning
IF    disable_warning.counter >=
      (Sample_Length - DisablingThreshold_for_purpose_i)
THEN  executeAction(enable, enable_warning)
```

That is, we are far more exigent for enabling back a service than for disabling it, because instead of being enough with some exceptions for disabling the service, we are demanding the whole sample to be successful with only some exceptions.

Till now, we have used informal language to describe the idea of what the rules of the Reacting Model means. But in practice, it is necessary to describe such rules in a formal language, preferably a standard one that is widely used and supported by tools able to execute it. In our case, we have used EPL because it is a standard language for processing events, but any other language for processing events could probably fit as well. For a matter of completion, here is the complete set of rules that we have identified for IoT in EPL language:

Rules for disabling a service due to a row of failures:

```
INSERT INTO LowReputationStreak SELECT * FROM PATTERN
[EVERY ([5] a=GlobalReputationEvent (a.userdata < threshold)) WHERE
timer:within(3600 sec)]
```

This rule creates a LowReputationSteak alert when a GlobalReputation event is received 5 times consecutively with a value under the threshold. For executing the reaction (stop the associated service) a new action can be created and associated to that alert in the SIEM web interface.

Rules for enabling reputation warning due to low Reputation Evaluations:

```
INSERT INTO NormalReputationStreak SELECT * FROM PATTERN
[EVERY   (a=LowReputationStreak   ->   ([3]   b=GlobalReputationEvent
(b.userdata > threshold))
WHERE TIMER:WITHIN(24 hour)]
```

This rule creates a NormalReputationStreak alert by receiving 3 events with reputation value above the threshold after a LowReputationStreak have been raised. The same way as before, an action that reactivates the service can be associated to that alert.

2.5.2 Generic Reacting System

So far, Section 2.5.1 has covered the ability to define a very limited set of reactions based in a limited set of data, which were contained in the Reputation Alert or in the context of the application. These limitations allowed us to provide an initial Reacting Model, but there are still two topics to cover, which are: what happens with non-trivial rules and how complex reactions are executed.

This section introduces some of the most relevant options to deal with both problems and the paradigm most commonly used to formally express the rules for reacting systems, which is the Event-Condition-Action (ECA) paradigm. This section explore the most relevant options for dealing with the mentioned problems and the most common rules, analysing the advantages and drawbacks of each of them for the environment where they are meant to apply. But vast nature of the IoT makes impossible to state that any of the proposed solutions is better that the other ones, simply because that depends on the Systems that are being exposed. For this reason, this section limits to presenting the options, analyse what kind of environments they are good for and how they can fit in the IoT and be used to construct the Reacting Model

Regarding how to integrate ECA languages with existing frameworks, such as the RERUM one, which whose SIEM is based in Java, there are already several studies about how to integrate ECA languages with Java, such as [Maatjes07], which introduces an ECA language named ECA-DL (ECA – Definition Language). ECA-DL is built to be integrated with JESS, which is a rule engine for Java.

2.5.2.1 Manually Defined Reaction

There is a problem with designing systems that automatically reacts to alerts, which is that in some critical infrastructures this is forbidden by law. Project MASSIF, which deals with the automatic detection of threats in critical infrastructures, warns already about this in its D2.1.1 deliverable on scenario requirement [Llanes 11]: In some countries, the legislation enforces strict protocols for treatment of some specific threats that must be managed by human beings. In such cases, the system does not even care about defining the reactions or only provides very simple tools for defining them, which excludes the use of a formal language for storing the rules. In these cases, instead of a formal language, what the System does is presenting a graphical interface to the user (GUI) that allows her to associate a predefined group of actions to the events in the system.

This is the worst case scenario, but must be considered because it is probably the most common one and hence must be taken into account. For instance, the project WISER [WISER 15] uses a SIEM similar to the one used in RERUM that follows this approach: Though it uses EPL to define the interaction of events, EPL does not directly supports associating actions to events. Thus, WISER provides a GUI to let the users to associate events with a predefined set of actions, which include the ability to launch a script.

This approach lets defining the rules in the System, but it does not let documenting it outside the application. For this moment, when it comes to define the model for a concrete system that works in this way, it is necessary to use any language that supports the concept of associating events to actions. And here ECA languages help again: though these systems do not interpret any kind of formal language, if the reaction is to be documented anywhere, it is necessary to have some language to specify the event, the condition and the reaction, and an ECA language, even an informal one, comes handy here. For this reason we recommend documenting such rules in any ECA language when it comes to define those reactions.

2.5.2.2 XML Based ECA language

As explained in [Jelia 06], it is important that any ECA language is complemented with a knowledge base, which should be richer than a simple set of facts and allow for the specification of both relational and classical rules. That is, it is necessary that the ECA language is able to interact with the system for both exchanging information and obtaining the rules to be evaluated.

There are several ways to deal with this knowledge base. As explained in [Boglaev 04] one of them is using XML for defining facts, rules and reactions as shown in the figure below taken from [Boglaev 04].

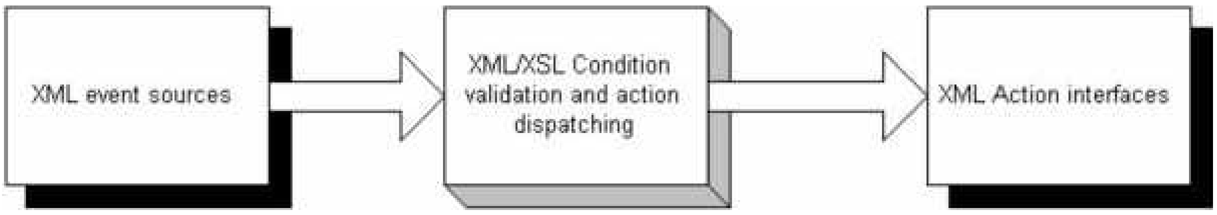


Figure 10: Combining ECA rules with XML to specify reactions

As the figure shows, event sources (in our case the Reputation Alerts) can be defined in XML. These sources can be crossed against proper rules defined in XML complemented with XSL to refer the fields of the sources and the actions to be dispatched.

Applied to our Reacting model, the XML sources and rules might look something like:

```
<!-- variables used in xpath expressions -->
<variables>

<!— Reputation Alert specification -->
<var name="data_alert">/dataAlert/
      <var name="reputation_global">/dataAlert/reputation_evaluation/global</var>
      <var name="reputation_accuracy">/dataAlert/reputation_evaluation/accuracy</var>
      <var name="reputation_availability">/ dataAlert/reputation_evaluation/availability</var>
      <var name="reputation_response_time"/ dataAlert/reputation_evaluation/response_time>/
dataAlert/reputation_evaluation/response_time</var>
      <var name="idObject">/ dataAlert/object/idObject</var>
      <var name="type">/ dataAlert/object/type</var>
      <var name="state">/ dataAlert/object/state</var>
      <var name="url">/ dataAlert/object/url</var>
      <var name="parent">/ dataAlert/object/parent</var>
      <var name="siblings">/ dataAlert/object/siblings</var>
</var>

<!—Configuration specification -->
<var name="Configuration">/config/
      <var name="threshold_global">/ config /threshold/global</var>
      <var name="threshold_accuracy">/ config /threshold/accuracy</var>
      <var name="threshold_availability">/ config /threshold/availability</var>
      <var name="threshold_response_time">/ config /threshold/response_time</var>
</var>
</variables>



<!—example of rule -->
<rule name="fire disabling action">
<if>
      <condition>$ data_alert/reputation_global <  $Configuration/threshold_global</condition>
      <action>interface:disable_object($data_alert)</action>
</if>
<else>
      <action>interface: enable_object($data_alert)</action>
</else>
</rule>
```

Some authors, such as [Bailey 06] go even further and suggest the chance for removing the dependency on a given ECA language combining XML and XLST for the rules. But they discard this option because XSLT would require processing the entire document before making any update to it. In the concrete case of IoT, where all data will come in streaming mode, this feature seems to be too constraining.

XML Based ECA languages offer the power of ECA languages and take advantage of the de-facto standards in document processing, which are XML and its extensions, making it a very interesting option for systems aiming to automate the definition and processing of reactions. But XML processing has the drawback of adding additional processing load. However, as the Reacting Model is meant to be invoked only when alerts are produced and when they are not redundant, the Reacting Model is expected to be invoked with a frequency low enough to let the system usually afford this additional workload.

2.5.2.3 Database Based ECA Language

Another alternative is to use database triggers as the source of events and have these events associated to ECA rules processed in the Database Managing System (DBMS) itself. This option is especially interesting for systems that store their information in databases, such as almost all business applications. In fact, some large database corporations such as Oracle support combining ECA rules with their database engines [Oracle 1] [Jelia 06] discusses how to use SQL3 triggers to fire the events involved in ECA rules. Other authors, such as [Kantere 06] go even further to support distributed ECA rules in multi database environments.

The point is many systems, especially those oriented to business, use database engines to store its information, and normally finish their operations executing some insert, update or delete operations on the database. Thus, if the tables of the database that will be affected by these operations can be stated on advance, then it is possible to set triggers for the insert, update and delete operations on those tables to be the sources of the events. The following figure shows the concept:

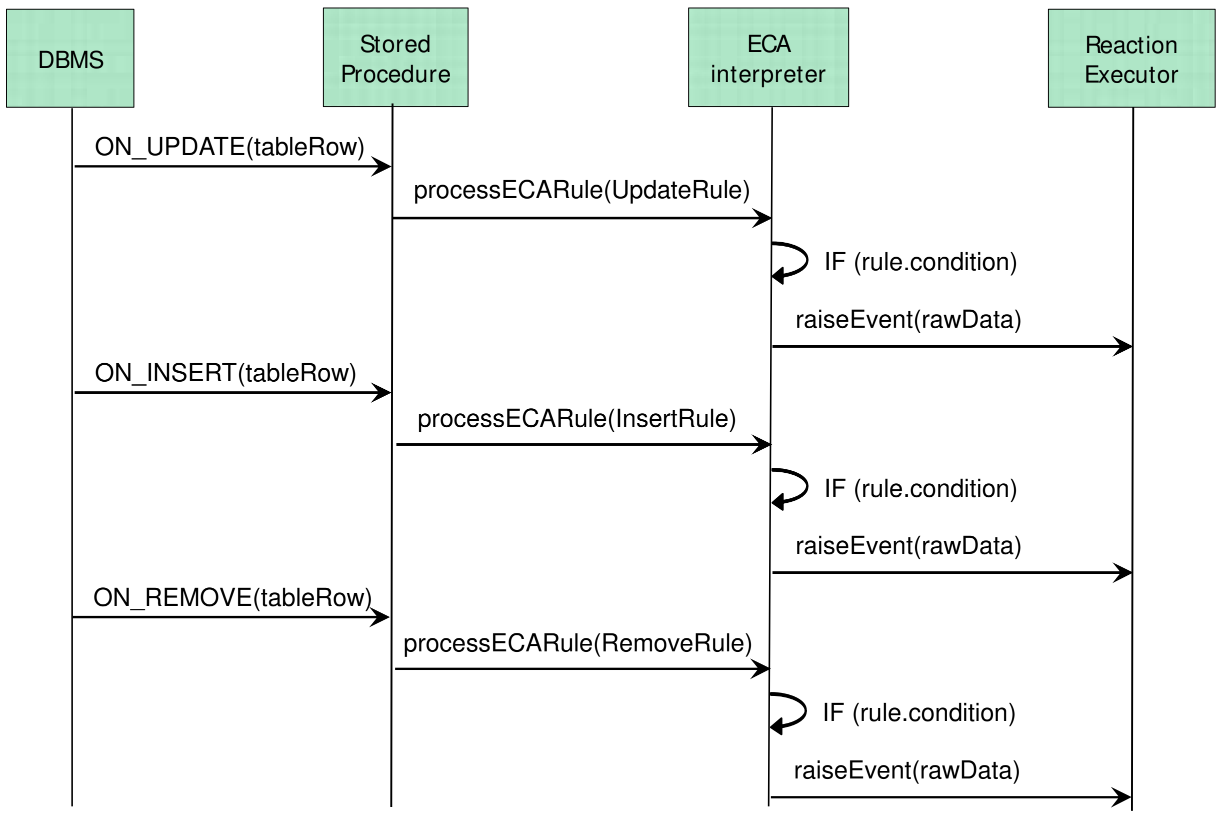


Figure 11: Executing Database based ECA Rules

As the figure shows, the DBMS triggers an ON_UPDATE, ON_INSERT or ON_REMOVE event for any of these operations passing the affected row to its appropriate Stored Procedure (beware that there will normally be a different Stored Procedure for each of them) varying the ECA rule to be executed. This rule is passed to an ECA interpreter that finally executes the Reaction Executor if the condition of the rule is triggered. It seems complex at first because of the need to trigger the operation on the database, define the stored procedure, provide the ECA interpreter and invoke the Reactor Executor, but having into account that in this approach the DBMS already provides all of that, it becomes very simple.

As mentioned, many business applications are based on database systems, which often support ECA languages, even natively, such as Oracle [Oracle 1]. For those applications, database based ECA languages can be an optimal solution, because they combine the versatility of a formal definition of reaction rules with the native support of the database.

2.5.2.4 Generic Execution Mechanism

The previous subsections introduced several reactions that make sense in almost any system, but after the system decides to execute the reaction associated to it, it is necessary to run the corresponding software implementing that reaction. But that association between the reactions and the corresponding implementing libraries can only be known at execution time, because the dynamic nature of IoT makes impossible to define in advance all the possible reactions and consequently their associated software libraries. Thus, it is necessary to provide a generic mechanism that is able to define and provide the reactions by the administrator when setting up the rules and being able to execute any software associated to the reaction. It is impossible to provide a library that fits the need for any IoT system but it is possible to define a generic execution mechanism that is able to execute a SW library whose access route is provided by the System Administrator. The key for that is defining an interface with a method that receives the Reputation Alert and is responsible for dealing with the corresponding reaction. For instance, in a java based system, it could be possible that the Administrator provides at run-time the name of a class that implements that interface.

The main drawback of this approach is that executing a sw library can be cumbersome if it is not integrated with the System that is invoking it. There are several ways to deal with this:

- Write the library to be invoked in the same language as the SIEM tool. This option would be ideal because the integration should be trivial but it is not always feasible, especially in the case of third party libraries;
- Invoke the library as a remote object via any shared communication channel, such as sockets. This option is more versatile, because it does not require the invoked library to be written in the same language as the SIEM, but it forces both the SIEM and the library to support that communication socket, with an additional effort for this and the need to protect such communication channel; and
  The SIEM can open a session and invoke the sw library via the command line. This is the most versatile option because it only imposes the ability of the sw library to be invoked from a command line as a program. Even if the library did not support to be invoked directly form the command line, adding it should be normally trivial. But this option can be the most expensive one in terms of resources consumed and execution time, because even if the execution of all reactions ran on the same Operating System Session, the execution of any program needs that the program is read and loaded in memory each time it is executed. And if the sw component was based in an interpreted language such us java, that execution could imply loading the JVM in memory each time the component had to be executed. For this reason, this option is not used very often except if the processing time is not an issue and the latency for the alerts is considerably less than the time needed to execute the program with the reaction.

2.5.3 Trustworthy Information Sharing

Section 2.5.1 presented several possible reactions to the Reputation Evaluations and Section 2.5.2 explained how to make a system execute those reactions according to the alerts produced by the Reputation Model. This section introduces how this will help in achieving Trustworthy Information Sharing.

First of all, it is important to clarify that Trustworthy does not necessarily mean true, accurate or exact. In fact trust in a given service is a subjective topic that depends on the evaluation criteria for that service, which may vary depending on the observer. For that reason, it is not possible to provide a universally trusted service. What is possible is providing a trust evaluation according to objective criteria and let this evaluation be available to the consumer of the information provided to the service, or alternatively, to disable the service to guarantee it will not keep on producing fake data.

And this is precisely the aim of the Trust Evaluation Model: disabling those services with extremely low reputation and providing trust evaluations for service consumers so they can decide on their own whether the information provider is worthy to be consumed or not. In other words, the Trustworthy Information Sharing will be based in using 2.5.1.3 and 2.5.1.4 to address both objectives.

In both cases, the system will rely on the mechanisms specified in Section 2.5.2.4 to invoke proper ad-hoc components that are able to pass the information of the Reputation Alert to the authorization components so this information is taken into account when authorizing requests. The reason for invoking the generic execution mechanism for this action, which is known in advance, is the implementation of this action depends on the nature of each system.

2.5.4 Heterogeneous Access Control and Profiles

Once it is established that the result of the Reputation Evaluation will be used in the authorization process, it is necessary to specify what will be that use.

This model proposes a way to make the result of the Reputation Evaluation available to the authorization components, which will be access to define fine grained authorization policies via referencing explicitly the result of the evaluation for each evaluation criterion. Additionally, the ability to combine several sets of access policies, including two specific sets for Reputation Evaluation (disabling and warning set of policies) makes the authorization process to be heterogeneous.

Again, it is necessary to take into account the diversity of IoT systems and, in this case, of the multiple authorization components. For this reason, this chapter introduces 3 kinds of access profiles instead of a single one, each of them with its own advantages and drawbacks that suit different kind of systems.

2.5.4.1 Reputation Evaluations available for Authorization Components

This option consists in storing the distinct Reputation Evaluations in a store of the system that can be accessed later by the authorization components. The idea is whenever a request accesses the system, the authorization components must access this store to obtain this information and use it in the access decision. Then the access policies of the authorization components may specify whether to block the access to the service or provide additional warnings based on low Reputation Evaluations. The following figure shows the process:

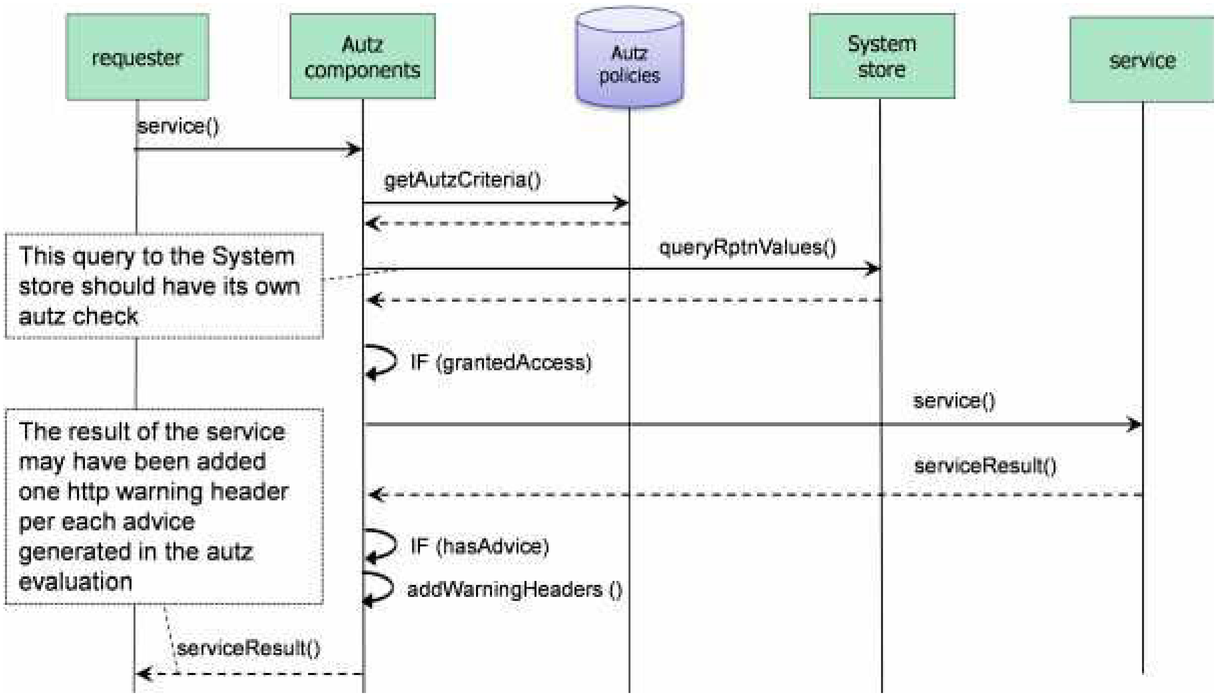


Figure 12: Supporting authorization for reputation with direct access to the System store

Note that actually the access to the System store should have its own authorization check and the authorization might even reject the request if the overall Reputation Evaluation were low enough. Apart from that, this figure is deliberately obviating the rest of authorization checks that are not directly related with the reputation for a matter of simplicity.

This process has the following implications:

1. If the authorization components are designed to be decoupled from the system they are protecting, then they would not have direct access to that store. Hence it would be necessary to add a service to provide access to this data to the authorization components;
2. That service would have to be protected as well;
3. It could be necessary for each authorization evaluation to execute at least one query to that store. If the authorization components were designed to minimize the network traffic, the addition of that query could have a heavy impact on the performance of the authorization components;
4. The values for the distinct evaluations of the reputation of the services are always up to date

2.5.4.2 Reputation Evaluations used in Authorization updated by Reactors

The procedure of making the Reputation Evaluations available for the authorization components can be refined to minimize its additional network workload by caching the result of the query per each service, but that would have the cost of implementing both the cache and a refreshing procedure triggered by the action of updating the store. The following figure shows the refined process for obtaining and refreshing this information:

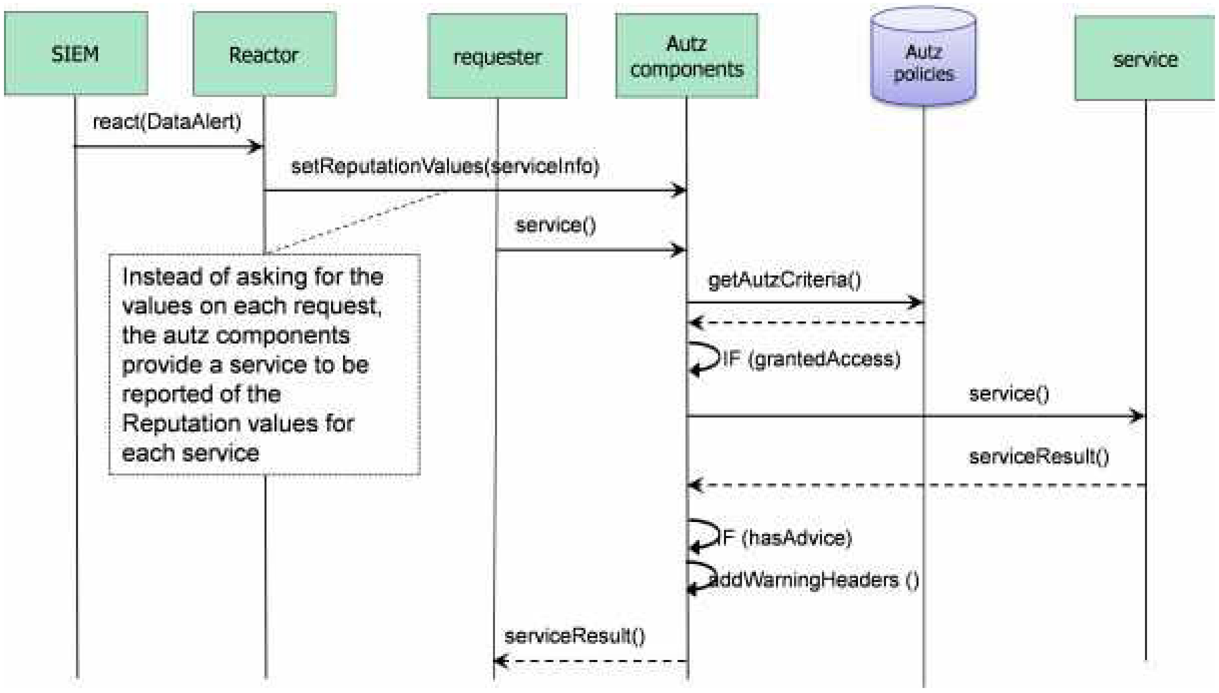


Figure 13: Reporting Reputation Evaluations to Authorization components

The figure is obviating the process of the method setReputationValues, because it is a responsibility of the Authorization components, whose implementation can vary. In any case, it is meant to store and cache internally this value so the Authorization components do not need to retrieve it again from the System store. Note that the information to be sent to the authorization components is not exactly the same as the Reputation alert, but obtained from it and containing, among other things, the identifier of the service.

This method has the additional advantage of letting the authorization components work even if they lose the connection with the System Store, and removes its main drawback, but it introduces two new drawbacks:

- It requires that its invocation is authorized as well. But this has been obviated from the figure for a matter of simplicity, too and
- It requires the authorization components to add the new method. This is not trivial, especially for third party components that might not publish its source code.

2.5.4.3 Specific Policies for Disabled Services

The first method explained in 2.5.4.1 consists in the Authorization Components invoking a remote System Store for each request. The second method explained in 2.5.4.2 removes the drawback of those remote invocations for each request via the reaction system providing those values for each Reputation Alert. This third method takes that approach a step further: instead of the reaction system providing the values for the authorization components evaluating them for each request, what the reaction system does is preparing the policies on advance embedding the values in the policy itself.

For instance, if a policy would normally state ‘IF global_reputation < global_threshold’, then the policy would be prepared to state ‘IF <global_reputation_value> < <global_threshold>’, being

<global_reputatio_value> the value for the global Reputation Evaluation and global_threshold the value obtained from the configuration of the service.

The following figure shows the messages involved in this procedure:

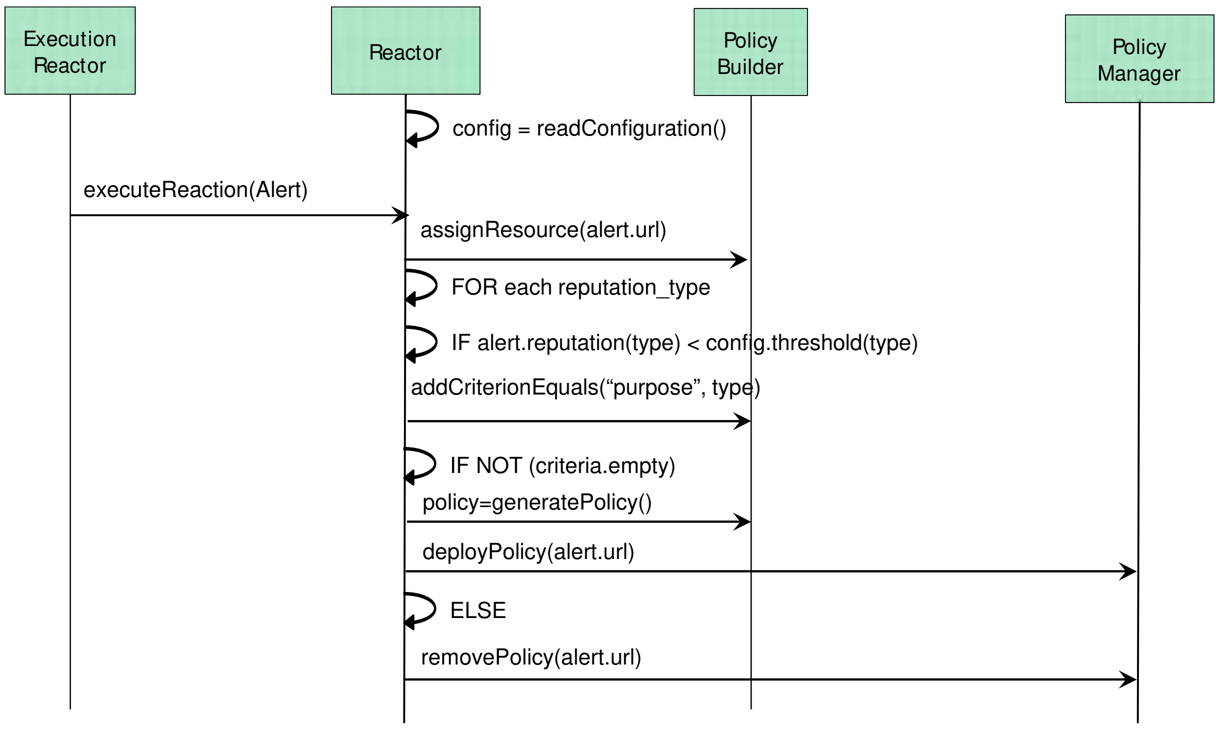


Figure 14: Embedding Reputation Evaluations in Access Polices

The benefits of this method are negligible in terms of performance compared to 'Reputation Evaluations used in Authorization updated by Reactor'. In both cases there are no remote invocations for obtaining those values. But this method has two main advantages over it:

- First and foremost, this procedure will fit authorization components that work only with the contents included in the http request, because the reputation information is embedded in the policy
- Secondarily, the policy will reflect the exact decision and the values that the policy is working with. This can help a system administrator to locate what the system is doing exactly with the reputation

It might seem that this method has the drawback of being less flexible because the generated policies are static, but this is not true, because they are generated as dynamically as the information they depend on changes. It might be argued that the improved speed for evaluating the policies comes at the cost of making the reactions being slower because they have to generate the new policies. But, taking into account that the Reputation Alerts are meant to be fired only when the Reputation Evaluation changes, and that the frequency for those changes is expected to be much lower than the frequency of the requests, the performance gain due to direct execution of the policies is expected to be bigger than the performance loss for building again the policies for each change in the reputation. Additionally, even if the update process took a very long time, it would probably not affect the evaluation of the authorization because it would normally be executed in a different thread because normally the authorization process will run on its own thread dedicated to listen in the port to be intercepted.

3 Integration of the trust model in RERUM

The Trust Model introduced in Chapter 2 is not architecture but a conceptual model for the IoT. In fact, that generic model contained some sections meant to be defined by each specific project. This chapter explains how that IoT generic model is applied to the specific architecture of RERUM and specifies those sections.

The chapter first introduces an overall picture of the integration of the IoT Trust Model with RERUM and the role of each component. The rest of the sections explain the internal design of the components explained in Section 3.1 in a way that goes from the collection of these data to the calculus of the reputation of the services, their reaction and the incorporation of that evaluation to the authorization process. In short, the chapter is organized this way:

- Overall integration of Trust Evaluation Model in RERUM Architecture: Overall view and explanation of how the Trust Model is applied to RERUM
- Interception of RERUM data: Explains how the RERUM data is intercepted so they can feed the processing of reputation
- Interception of external data: Explains how external data requested by RERUM components is intercepted so they can feed the processing of reputation
- Managing context data: Explains how the components implementing the Trust Model interface with the Context Manager component to make the Reputation Evaluations available to the processing of reputation
- Processing of reputation: Explains how the Trust Evaluation Model evaluates its rules to calculate the reputation and trust values, which are then passed to correlating reactions to Reputation Alerts
- Correlating reactions to Reputation Alerts: explains how the Reputation Alerts that reach the Reacting engine can be correlated with each other to produce more refined alerts
- Processing of reactions: Explains how the Reacting Model is able to associate rules to the Reputation Alerts and how they trigger the generic execution system
- Incorporating the Reputation Access Profiles to the Authorization Process: Explains how the reactions can set up the authorization components by updating their Reputation Access Profiles in the form of Security Policies
- Trust Reputation Evaluation for Devices at the network level explains a RERUM specific Observer to produce trust ratings from the RERUM nodes at network level
- Processing Reputation for Users at Service Level: Explain how the reputation of the users can be incorporated to the authorization process
- Trust Reputation Evaluation for Devices at the network level: Explains a specific RERUM of figure that allows to perform a Trust Evaluation of the nodes at network level and generate trust ratings for the nodes that feed the Reputation Engine

3.1 Overall integration of Trust Evaluation Model in RERUM Architecture

The following figure shows an overview of the trust, reputation and reactor components and how they fit in the RERUM architecture:

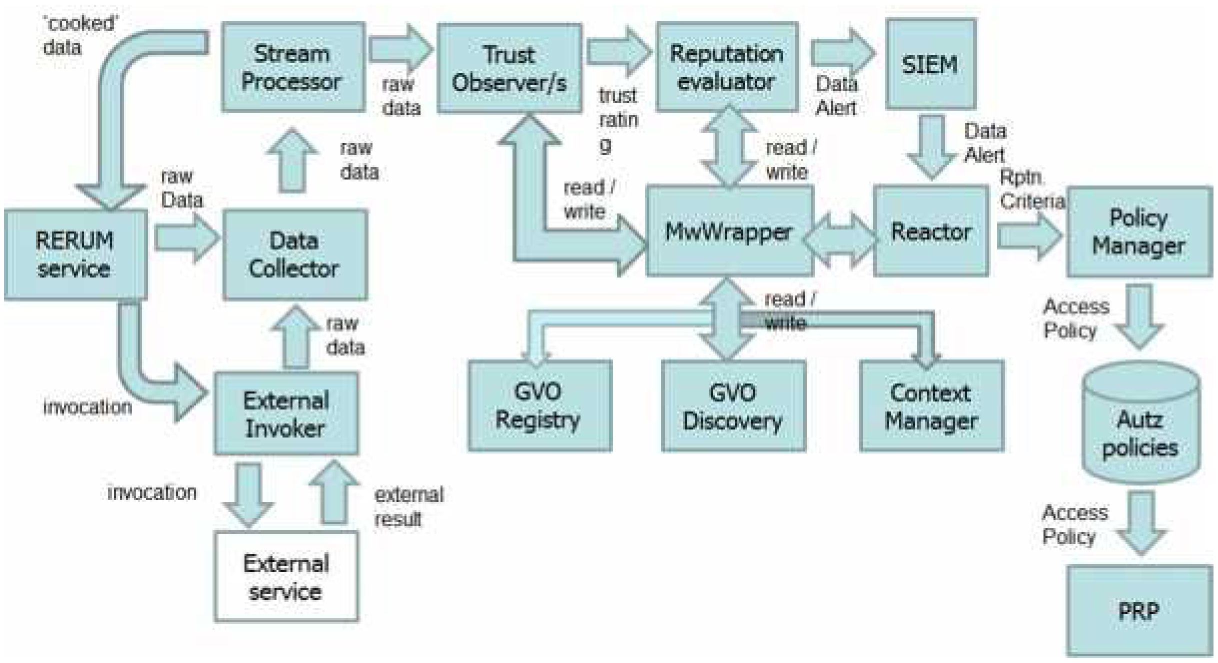


Figure 15: Overall view of Trust & Reputation Components Design

In RERUM, all data travels through the data collector and Stream Processor. Data from Data Collector are said to be in a 'raw state', that is, they are in exactly the same state that they were generated. Data from external services reach the Data Collector from the External Invoker. The Stream Processor is the component that can perform additional process on the data, such as performing statistical operations or updating the reputation of the services according to the data they are providing, and this is precisely the place where the Trust & Reputation components are invoked. Note that the Stream Processor is in charge of performing any necessary calculi, which include, but it not necessarily limited to the Reputation Evaluation. For this reason, what the Stream Processor does is delegating each kind of calculus to a proper agent, such each of the Trust Observers. But different implementation of RERUM might attach additional calculi with their corresponding implementing components. That is why the neither the trust observers or the Reputation Evaluator are part of the Stream Processor. Instead, they are separate components that are invoked by the Stream Processor to achieve the calculus of the Reputation Evaluation and its associated reactions.

The first Trust & Reputation component to be invoked is the Trust Observer, which can potentially be more than one because there is one instance per service invoked; each of them with a configuration (set of rules) according to the service they are evaluating. The trust observer is in charge of producing the trust ratings explained in Section 2.4.3. These trust ratings are delivered to the Reputation Evaluator to produce a joint reputation evaluator that is delivered to the SIEM, which is the program controlling the flow of the Reacting Model and in charge of invoking the reactors. Any trust observer, Reputation evaluator and reactor are meant to be able to read or write information about the GOs being evaluated through a MwWrapper object, which is in charge of wrapping the complexity of accessing the information in the MW and work in the streaming mode needed by the Trust observers and Reputation evaluators. In the POC prototype provided with the deliverable, the Observers and the Reputation evaluator are integrated in a single Reputation engine, which is described in Section 3.5. More details on the MWWrapper can be found at Section 3.4.

After the Observers produce their respective reputation ratings and pass it to the Reputation Evaluation, it may produce a Reputation Alert depending on its Reputation rules. These Reputation Alerts are received by the SIEM that decides what is the proper reactor for them according to the SIEM EPL rules.

Finally, the reactors are in charge of executing the proper from the SIEM. More specifically they update the authorization policies of the system through the Policy Manager. These policies will be later retrieved at run-time by the PRP, which is the part of the authorization components responsible for retrieving the policies applicable to each request.

3.2 Interception of RERUM Data

This section explains how the raw data produced by the sensors is intercepted to be delivered to the trust components analysed so that they analyse the data to revaluate the reputation of the service.

According to D2.5, the data travels in RERUM according to the following figure, which is extracted from D2.5 but is slightly modified to highlight the most relevant functional components for the purpose of Reputation Evaluation:

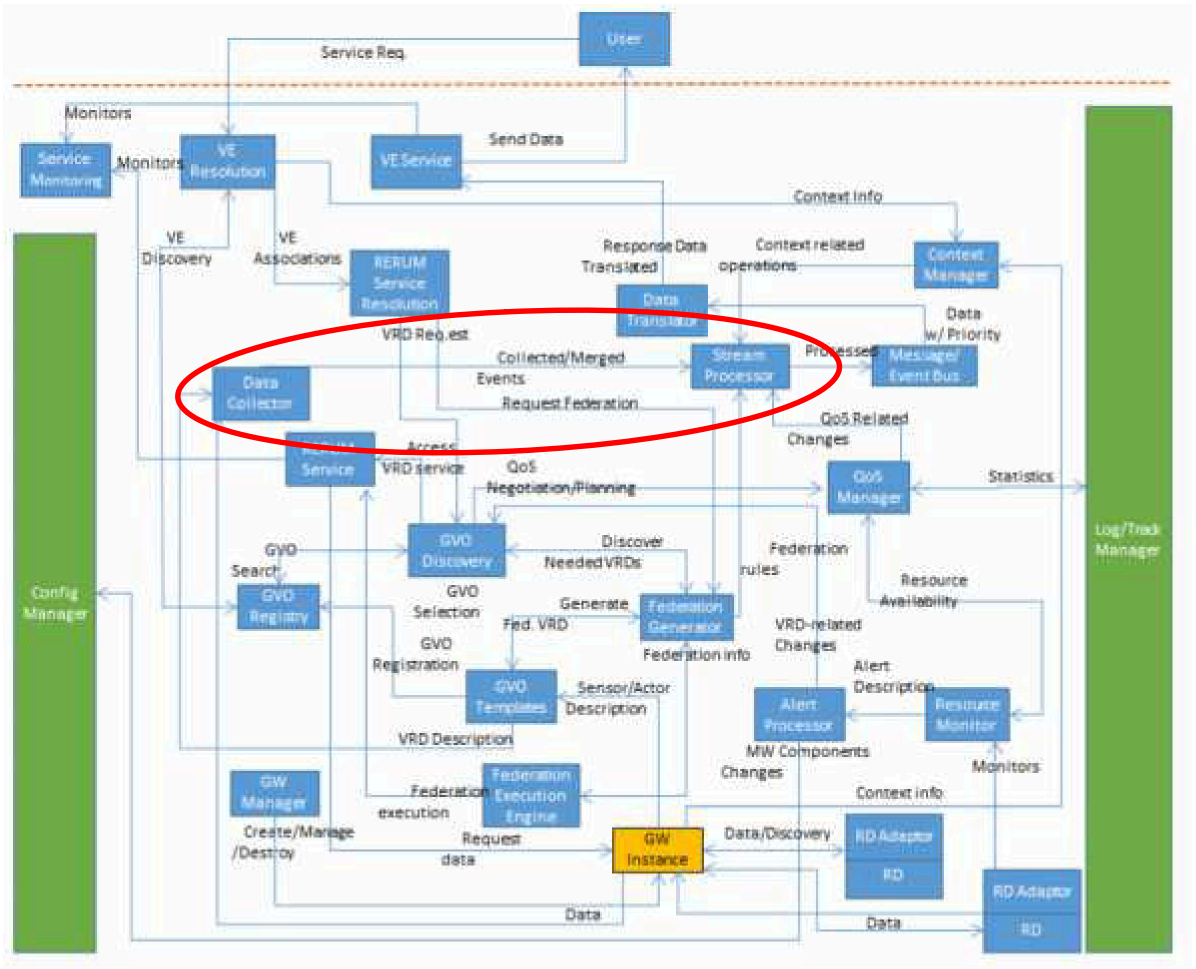


Figure 16: RERUM Architecture with Data Collection highlighted

As the figure shows, all data travelling in RERUM passes first through the Data Collector in a raw state, that is, exactly as they were produced and immediately after this is passed to the Stream Processor, which is in charge of performing any additional calculi on the raw data before releasing this. Typical additional calculi may be performing statistics, but do not necessarily limit to that; in fact, it will be precisely the Stream Processor that will invoke the reputation components, because the Reputation Evaluation will be a function from the current reputation of the devices/services and the incoming data.

Strictly speaking, the Trust Evaluation Model proposed in Section 2.4 works with a stream of raw data, but this would not take advantage of the Stream Processor, which is specific of RERUM and can provide additional data to the observers that would be quite difficult to retrieve. The project opted for optimizing the use of the generic Trust Evaluation Model by incorporating this feature specific from RERUM. Additionally, this feature would ease the integration of RERUM with an adaptation of the POC prototype included in Section 3.5.2.

This section explains how to upgrade the Stream Processor to generate the data events needed by the trust observers to generate the trust ratings. An upgrade is needed because the Stream Processor of RERUM does not support the collection of context data needed to retrieve some of the fields included in the event. However, it is out of the scope to provide a full detailed description of the design of the Stream Processor, because that is already included in [RD3.2]. Instead, this section will focus only in those steps needed to produce the data event to be delivered to the trust evaluator.

The Stream Processor already works with a Declarative Language, named DOLCE, which was created for the Stream Processor and defines events according to incoming data and operations to be executed on them. In the case of evaluating the reputation, that evaluation will consist on:

- Defining the proper DOLCE rules for incoming raw data;
- An Event Collector will collect incoming events with the raw data from the Data Collector. In practice, this is achieved by obtaining messages containing those data from the Message/Event Bus where the Data Collector sends them;
- A Complex Event Detector will prepare a proper Data event with the information needed by the trust observers, which involves the following sub-steps:
  - Evaluate the Dolce rules against the raw data collected;
  - Invoke a Context Collector object to collect the needed information from the Context Manager. This feature is the upgrade mentioned and
  - Create a proper Data event to be consumed by the trust observers.
- Send the created Data Event to the Complex Event Publisher which will deliver internally to an Emitter object and
- Emit the Data Event to the observers through an UDP connection.

The figure below shows the main classes involved in the process.

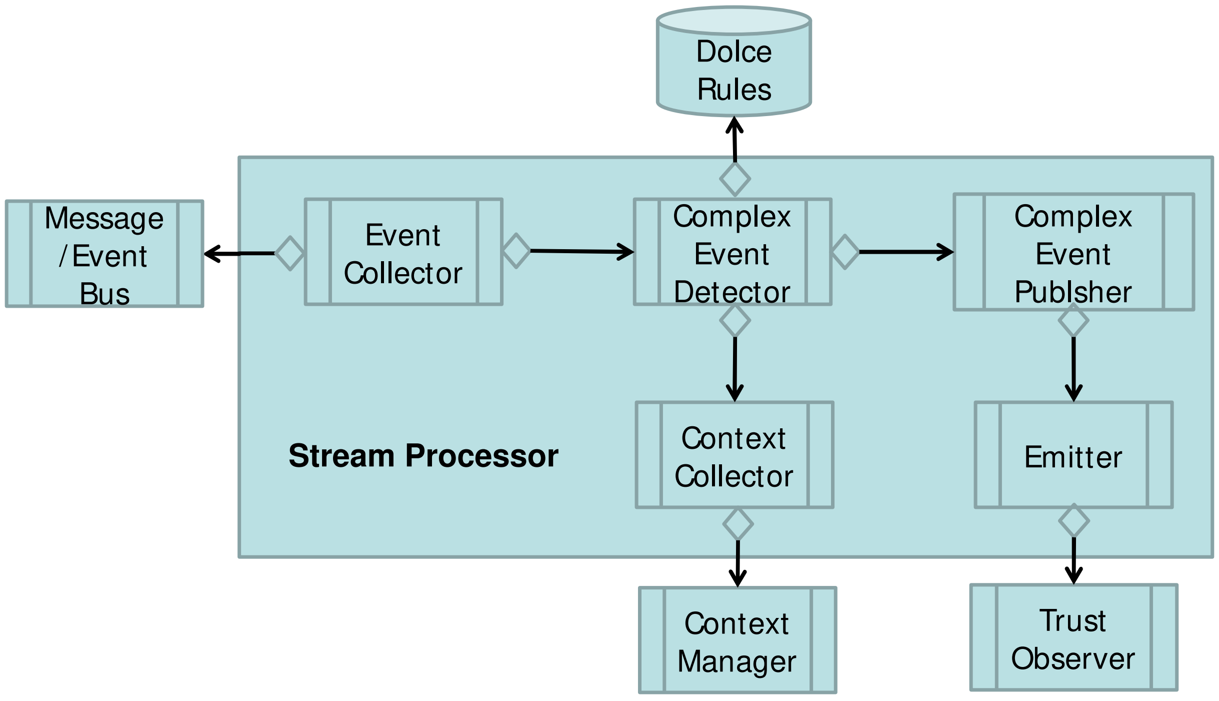


Figure 17: Main classes involved in Data Interception

The flow of messages through the different classes is shown in the following sequence diagrams. Due to the big number of classes involved, we have split them in three for the collecting events, creating events and sending events operations.

The figure below shows the message involved in the interception of raw data:

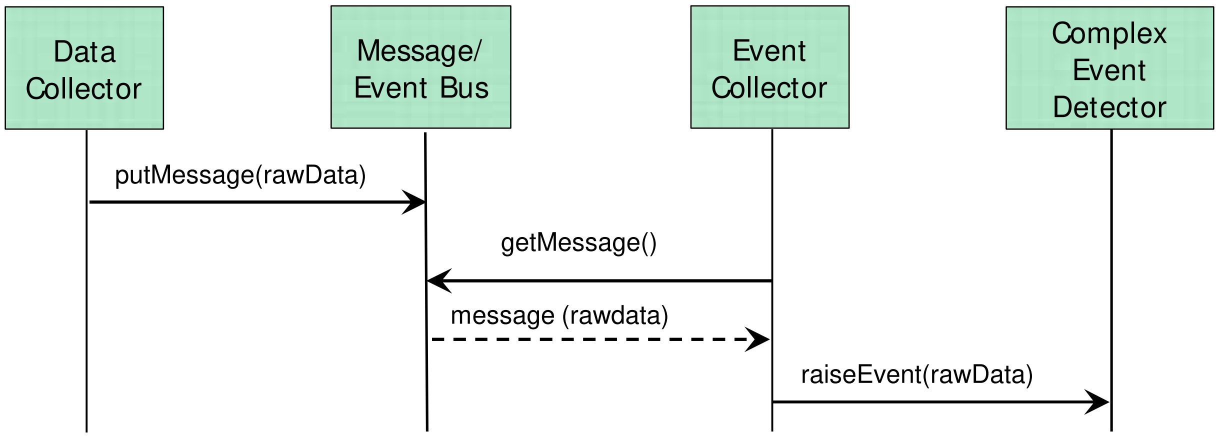


Figure 18: Intercepting raw data

The data collector puts incoming data as messages in the Message/Event Bus that are later retrieved by the Event Collector, which delivers them to the Complex event detector for their processing.

As the next figure shows, the DOLCE rules are read on advance, independently of the arrival of the events containing the raw data. When the raw data arrives, the Event Collector creates an event with them and delivers it to the Complex Event Detector, which crosses it against the DOLCE rules. There must be some rules that create the proper Data Events according to the raw data contained in the event and the information contained in the context.

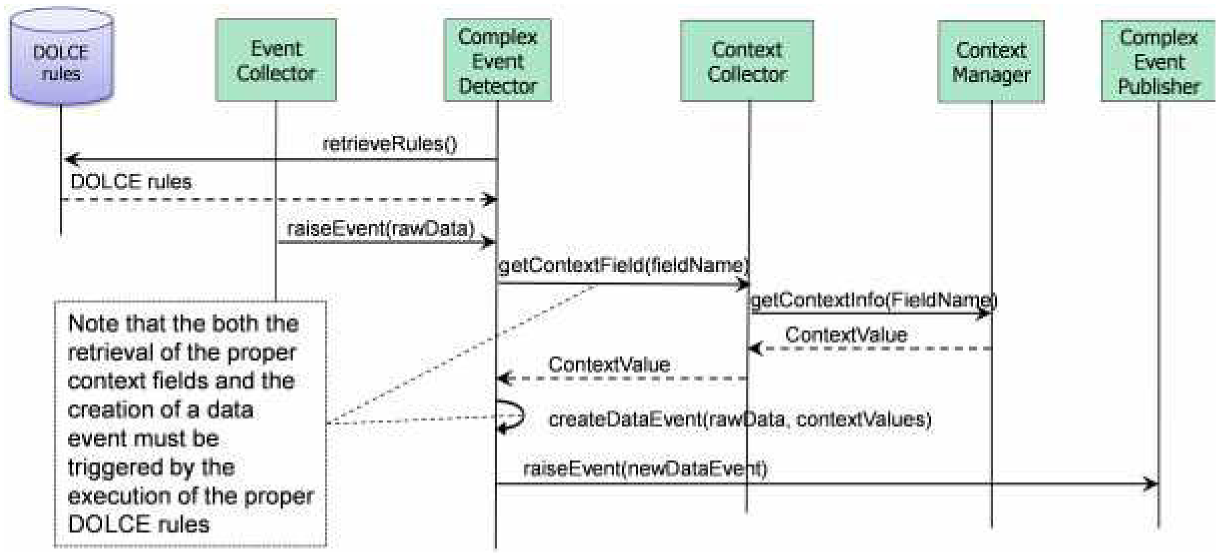

DOLCE rules
Event Collector
Complex Event Detector
Context Collector
Context Manager
Complex Event Publisher
retrieveRules()
DOLCE rules
raiseEvent(rawData)
getContextField(fieldName)
getContextInfo(FieldName)
ContextValue
ContextValue
createDataEvent(rawData, contextValues)
raiseEvent(newDataEvent)
Note that the both the retrieval of the proper context fields and the creation of a data event must be triggered by the execution of the proper DOLCE rules

Figure 19: Creating the data event for the trust observers

Those fields are retrieved from the Context Collector object that communicates with the Context Manager of the MW for this purpose. The reason for not collecting these values directly from the Complex Event detector but delegating it in a different object is this allows the Complex Event Detector to abstract itself from the different possible methods for retrieving external information. The Context Collector is a new sub-component created for wrapping the complexity of invoking a distributed MW object as if it was a local oen.

The following figure shows the process of delivering the newly created data events to the trust observers:

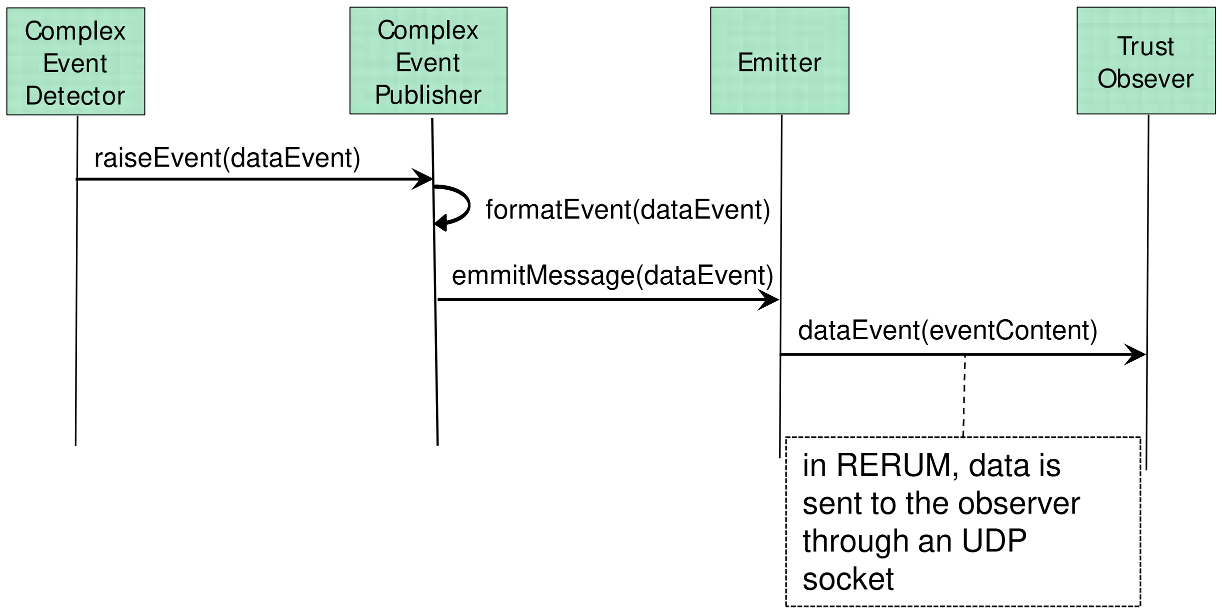


Figure 20: Emitting data events to trust observers

Note that the Complex Event Publisher is responsible not only for sending the event, but also for choosing the recipient and formatting the message according to it. In the concrete case of the evaluation of the reputation, the process is the one shown in the figure, but for the delivery of data in RERUM, there is another distinct event that corresponds to the delivery of the data to the services. This event generates its corresponding message sent to the message/event Bus. The description about how these messages are processed does not change from the procedures already described in [RD2.4].

Though this section has already presented the procedure for generating the data events, it depends on some DOLCE rules to be configured in the system in order to create the events. Here is the specification of such rule:

```
event incoming_data {
 use {
      string idSource,
      byte[] content,
      int type,
      };
}


complex data_event {
 payload {
      string IdSource=idSource,
      string IdParent=getContextField('parent', idSource),
      string[] Siblings= getContextField ('siblings', idSource),
      int Type=RERUM,
      byte[] Content=content
      };
      detect incoming_data
}
```

These rules work this way:

- the first block of code for incoming data defines an event for intercepting the data coming from the Data Collector;
- The complex data event structure has two main parts:
    - A series of field declarations that indicates the content of the structure and how to fill it, including references to the new internal DOLCE function getContextField, which, as shown in Figure 19: Creating the data event for the trust observers, invokes the context collector to retrieve the proper value and
    - The 'detect incoming_data' clause, which states the condition for this event to be raised, which in this case, is the event incoming_data.

So, in short, what this code does is defining an event for incoming data and another event which is triggered by the arrival of incoming data and fills its content with help of the new internal DOLCE function, which invokes the Context Collector to retrieve proper values

Note that though it is possible to change the DOLCE rules, the ones provided here are enough to do our goal. That is, they intercept the incoming data and invoke function getContextField to obtain the needed external data.

3.3 Collection of Data from External Services

The previous section explained how data generated in RERUM were intercepted so it could be possible to deliver this data to the trust observers. But RERUM aims to be able to evaluate the reputation not only of the RERUM services, but also of external services, and for that, it is necessary to feed the observers with the external data produced by these services, too.

Actually, the design already provided for intercepting the RERUM data will work for external data, too, but with two exceptions:

1. In that design it is assumed that all data used in RERUM will travel through the Data Collector to the Stream Processor, but this is not necessarily true. In fact, RERUM does not include any scenarios where the services need to actively ask to external services for data, and for that reason, the RERUM MW does not include any component for asking for external data yet; and
2. All RERUM data must be labelled when generating their corresponding data event as 'RERUM', while external data must be labelled as 'external'. The reason for this is the processing of RERUM and external data is slightly different in the Trust Model, and hence it is necessary to provide this datum to properly process them.

Because of these two reasons, if RERUM needed to invoke external services and evaluate their reputations, it would have to be upgraded with a brand new component to be invoked by any RERUM component when trying to access the external services. This component would have to invoke the external component and put its corresponding value in the Message/Event Bus for the Data Collector. The following figures show the process and the involved MW components.

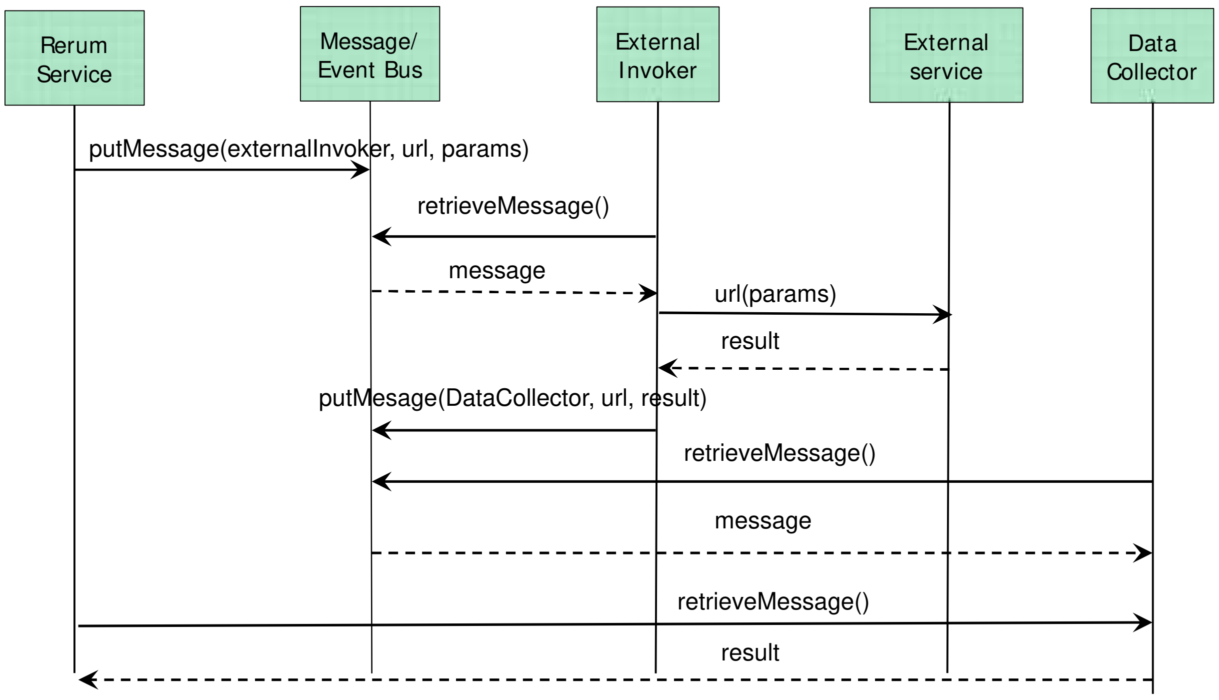


Figure 21: Invoking external services from RERUM

As the previous figure shows, all RERUM services that need to access external services should do it through the External Invoker class. As the invocation of distinct MW components is carried out through the Message/Event Bus, this means in practice that this invocation consists in putting a message in it with an identifier of the recipient and the message which, in this case, contains the url to invoke and their corresponding parameters. Once the external invoker has retrieved the message containing the url to invoke and their parameters, it invokes the url, gets the result and put it in another message to the Message/Event Bus addressed to the DataCollector.

However, there are two minor differences between the generated data event and the one generated for RERUM data:

- All Data Events have a source identifier that identifies the origin of the entity generating the information, let it be a RERUM device, a RERUM service, or the external service. In the case of RERUM devices and services, this identifier is the one held by the MW, but in the case of external services it corresponds with the url to access that service; and
- All Data Events have a field named type that indicates whether they correspond to a RERUM generic object or an external service. In this case, its value will be 'external'.

Once the result of the invocation arrives to the data Collector, its process will follow the be the same as the RERUM data, that is, it will be delivered to the service that requested it and to the trust rating observers so they can evaluate a trust rating for this service.

The DOLCE code to be configured for external data is almost identical to the one for internal data:

```
event incoming_external_data {
 use {
      string idSource,
      byte[] content,
      int type,
      };
}

complex data_external_event {
 payload {
      string IdSource=idSource,
      string IdParent=getContextField('parent', idSource),
      string[] Siblings= getContextField ('siblings', idSource),
      int Type=EXTERNAL,
      byte[] Content=content
      };
      detect incoming_data
}
```

As it can be seen, it is almost identical to the one from RERUM data, being the only difference the source of the event and the type that is delivered to the Trust observer, which in this case is EXTERNAL. That is, the event incoming_external_data defines the structure of the incoming data, the event data_external_event defines the structure of the outgoing event, including the type set to external and the ‘detect incoming_data’ clause, which indicates that the data_external_event is raised when the incoming_data event arrives.

3.4 Managing Context Data

Both Trust evaluation components and Reacting components need to access and manipulate data stored in the RERUM MW. These data include the reputation values associated for each publicised evaluation criterion associated to each service and device. That is, for *m* possible services, *n* possible devices and *o* possible evaluation criteria, there could possibly exist a maximum total of (*m*+n) X of possible values to be accessed and stored in the MW context. As it is explained in Section 3.2, it is out of the scope of these components to deal directly with the MW, and therefore they need a component where to delegate this task. We have named this component MwWrapper.

The MwWrapper deals with the access of that information in the RERUM MW, which is a complex task because it involves communicating with the following MW distributed components:

- GVO Registry to access the attributes associated to the GVO;
- GVO Discovery to discover any other GVO associated to the GVO being queried, such as any possible service associated to it, any children nodes depending on it (for RERUM services). This information is needed to know if the GVO is a service (which have children nodes associated) or a sensor (which will have no children but a father service instead) and to locate the other GVO that correspond to the same service, which is needed to compare the values provided by a node to the ones provided by the rest of nodes of the same service; and
- Context Manager for temporary data associated to the object.

And there is another reason. Both the Trust evaluation Engine and the Reacting Engine work in an asynchronous streaming mode. For this reason, it is necessary that this intermediate object is also able to properly convert the synchronous read and write operations in the asynchronous mode supported by the mentioned engines.

As the MwWrapper is built to work with the Trust & Reputation components, which receive their information and provide their results on UDP Streams, the way to invoke these methods follow the same procedure. In concrete, to invoke a method with name 'method_name' the invoker must include in its output UDP stream the content method_name (<parameter_list>):ipAddress:port, where <parameter list> is the list of the values for the arguments of the method separated by commas and ipAddress:port is the ipAddress and port where the invoker will be waiting for the result, which will always have the form method_name: <result> where <result> is the result of the operation.

Given the asynchronous modes of the engines, the read operations cannot be invoked directly. Instead, it is necessary that the engines work with the MwWrapper using a subscribe/listening mode. That is, the engines subscribes to the MwWrapper indicating what context attributes they want to receive and the MwWrapper will send a message to the engines in two events: immediately after subscribing and whenever these values change.

Note that it is impossible to subscribe to the fields individually because the fields are related with each other. That is, there is no sense in getting a value for the list of siblings without specifying the object that those siblings refer to. And similarly, there is no way that the MwWrapper can obtain that information without knowing the objectId.

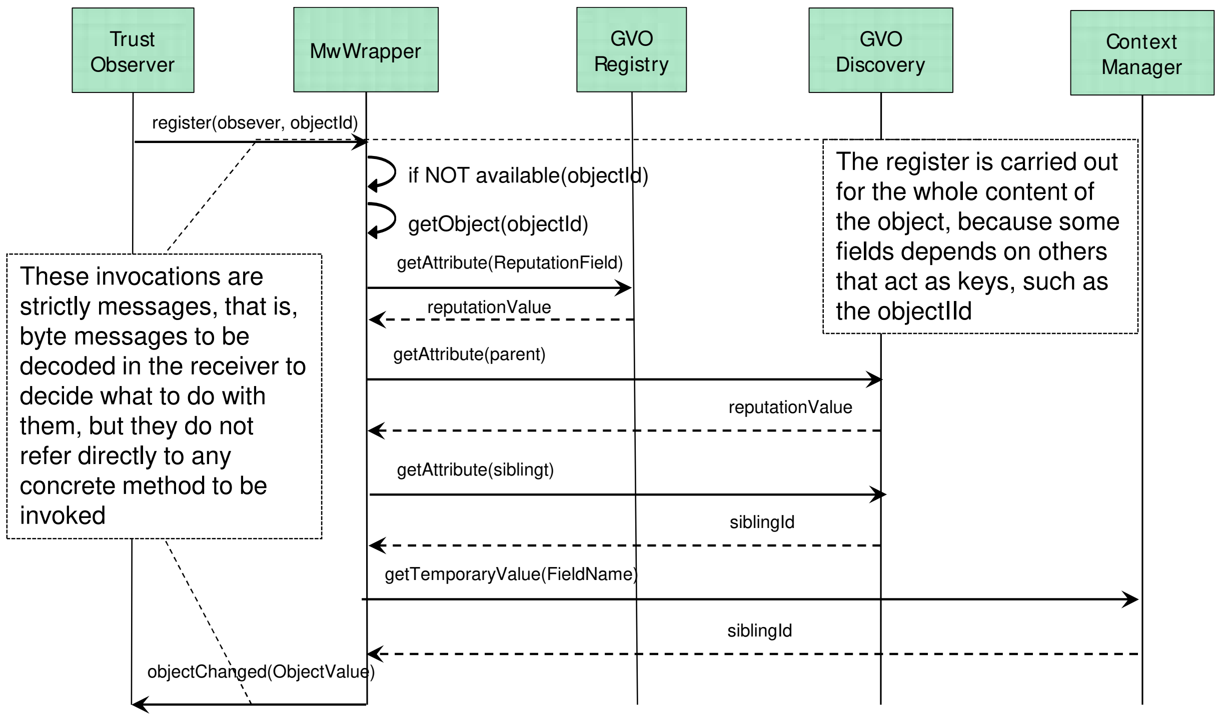

Trust Observer
MwWrapper
GVO Registry
GVO Discovery
Context Manager
register(obsever, objectId)
if NOT available(objectId)
getObject(objectId)
getAttribute(ReputationField)
reputationValue
getAttribute(parent)
reputationValue
getAttribute(siblingt)
siblingId
getTemporaryValue(FieldName)
siblingId
objectChanged(ObjectValue)
These invocations are strictly messages, that is, byte messages to be decoded in the receiver to decide what to do with them, but they do not refer directly to any concrete method to be invoked
The register is carried out for the whole content of the object, because some fields depends on others that act as keys, such as the objectIId

Figure 22: Obtaining context values

As the previous figure shows, the register operation needs that the registering object identifies itself. Due to the streaming mode of the trust & reputation components, this, in practice, providing an ipAddress:port combination so the MwWrapper can respond with the values later. For this reason, the registered component must be waiting in that ip address and port for the response.

The register operation also needs that the registering component indicates the object it wants to register to, which is provided in the objectId argument. In case that the registering object does not know the identifier of the GVO that it is trying to inspect, the MwWrapper provides additional query methods for this purpose that are discussed later.

After the observer has registered for a given object to the MwWrapper, it immediately checks if it already has the values for the listened object. In case it does not have the information, it proceeds to retrieve it from the GVO Registry, the GVO Discovery and the Context Manager. Once it has retrieved all of it, it generates a new message with the information of the object and sends it to the ip address and port that the listener observer when registering.

This way of registering the classes willing to access information of the context is the same for the Trust Observers, the Reputation Evaluators and the Reacting Components.

Regarding the way to obtain the objectId of the components, MwWrapper offer two methods for this purpose:

Int getServiceIdByName(String serviceName), which given a service name looks for the corresponding service in the GVO recovery and returns its identifier. If there were no such service, it would return -1. The next figure shows the process.

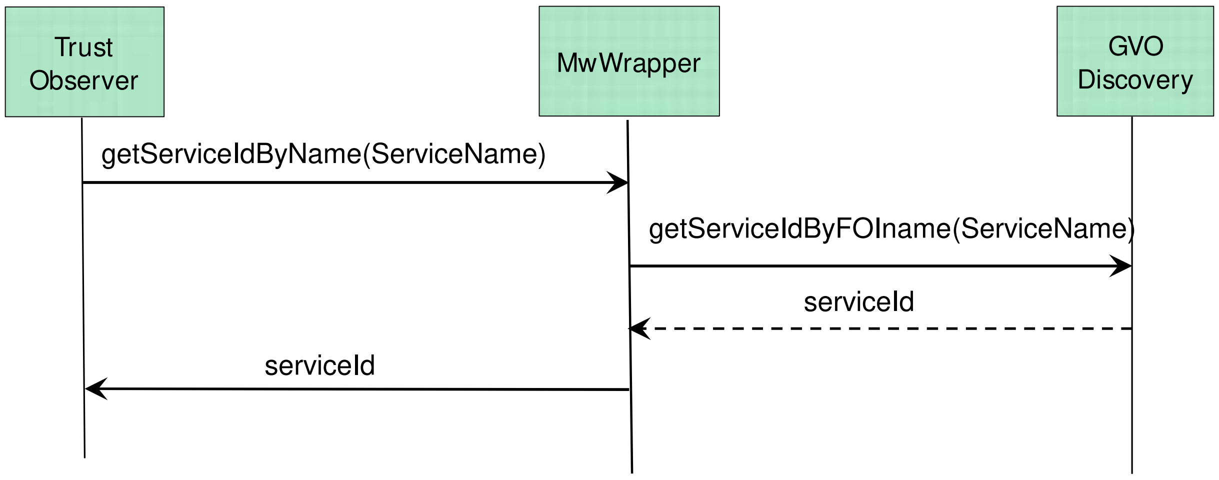

Trust Observer
MwWrapper
GVO Discovery
getServiceIdByName(ServiceName)
getServiceIdByFOIname(ServiceName)
serviceId
serviceId

Figure 23: Retrieving the ObjectId for a service

Note that the service could be an external service instead of a RERUM service. For this reason, in this case the MwWrapper would return the same value of serviceName.

Int[] getDeviceIdsByService(String service), which given a service name looks for the corresponding GOs associated to it and returns their identifiers separated by commas. If there were no such service, it would return a single -1 value. If the service existed but had no GOs associated to it, it would return an empty list.

Again, it is feasible to ask for the GOs associated to an external service. In this case, the MwWrapper would return an empty list, because there is no way the system can know about the GOs associated to an external service or interact directly with them.

Normally an observer would invoke the method getServiceIdByName to obtain the id corresponding to the service and later invoke getDeviceIdsByService to obtain the list of devices associated to the service. The following diagrams show the process.

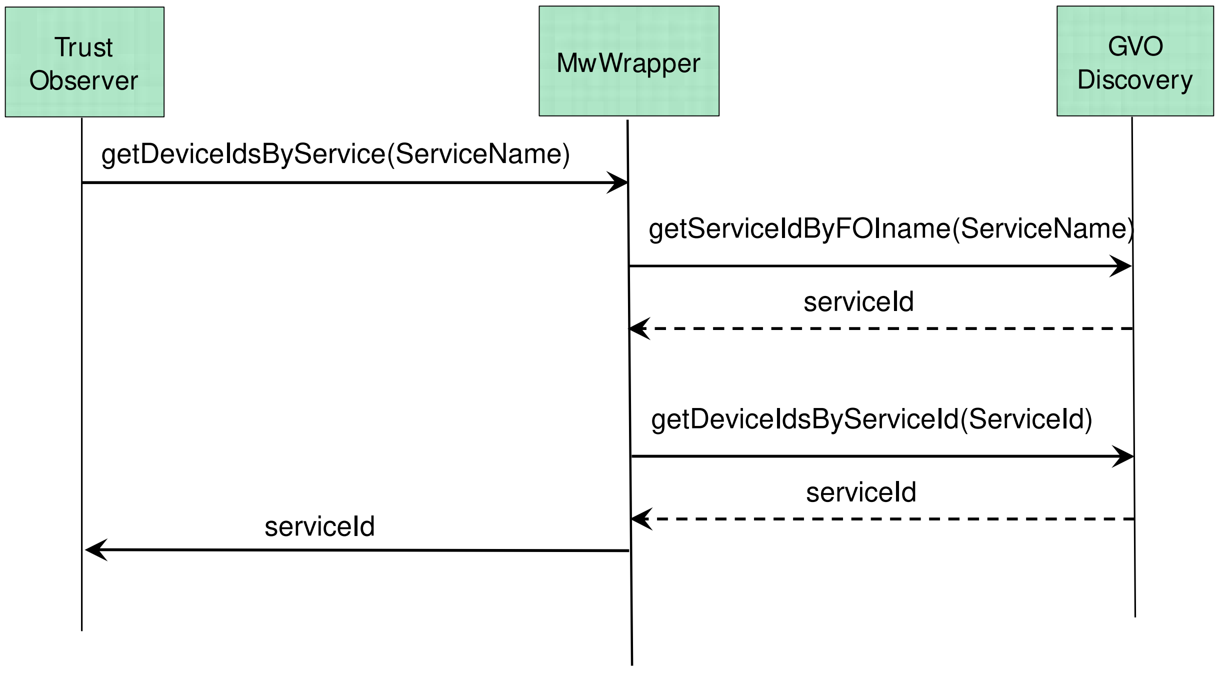

Trust Observer
MwWrapper
GVO Discovery
getDeviceIdsByService(ServiceName)
getServiceIdByFOIname(ServiceName)
serviceId
getDeviceIdsByServiceId(ServiceId)
serviceId
serviceId

Figure 24: Retrieving identifiers for Rerum Devices

For the write process there is no need to subscribe, but the write operation will trigger the event of sending the new value for the context attribute to each object currently listening for them. The following figure shows the process.

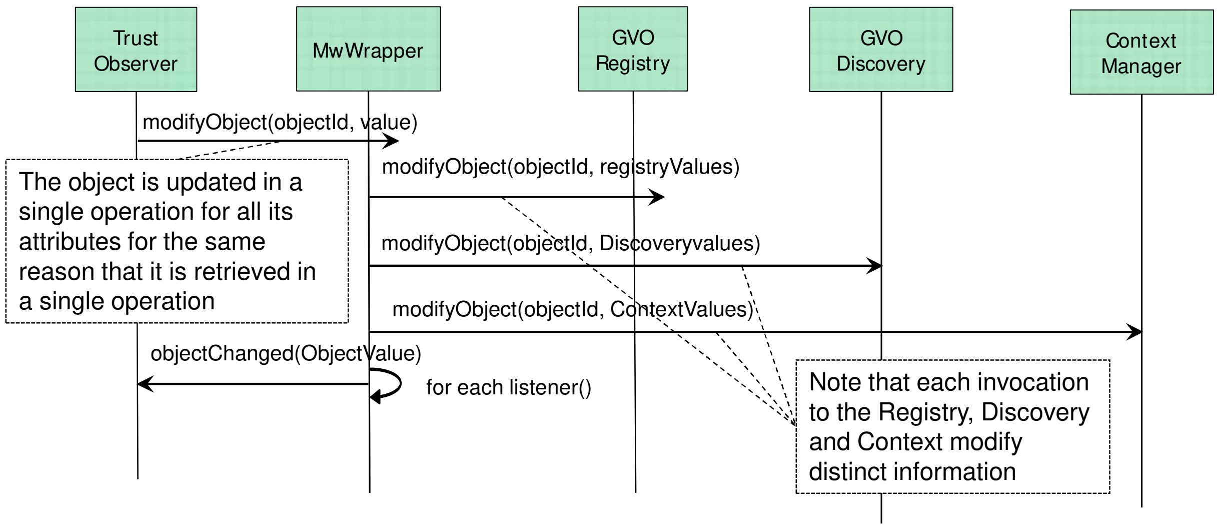

Trust Observer
MwWrapper
GVO Registry
GVO Discovery
Context Manager
modifyObject(objectId, value)
The object is updated in a single operation for all its attributes for the same reason that it is retrieved in a single operation
modifyObject(objectId, registryValues)
modifyObject(objectId, Discoveryvalues)
modifyObject(objectId, ContextValues)
objectChanged(ObjectValue)
for each listener()
Note that each invocation to the Registry, Discovery and Context modify distinct information

Figure 25: Modifying context values

Again, we have used the Trust Observer as an example class of what the process involves, but it is completely analogous for any other Component trying to modify the content of the context associated to a GVO.

For consistency with the way to read the object the values of an object requires providing the content for the whole object. Besides the consistency reason, updating some values of the context may require the update of several attributes at once. When invoked, the MwWrapper properly updates the GVO Registry, GVO Discovery and Context Manager with the parts of the object that affects them. Finally, after finishing the update operation, the MwWrapper notifies all the objects that registered for listening the values of that object (which included the Trust observer).

3.5 Processing of Reputation

The previous sections have explained how both RERUM data and external data are gathered to be delivered to the Trust Observers and how the components that implement the Trust Model are able to interact with the MW to deliver their results, know the context and store internal temporary data associated to the objects being analysed. This section explains how the trust and reputation components perform the evaluation.

But the Processing of reputation has a special feature: The project has selected it to provide a POC because it is considered to be a central feature of the trust model. For this reason, this section contains not only the design of these components but also the technical details of the POC program.

3.5.1 Overall Design of the RERUM Reputation Engine

In the case of RERUM, the reputation engine is comprised by:

- The Trust Rating rules, written in CLIPS
- The Reputation Evaluation rules, written in CLIPS
- A CLIPS interpreter
- An incoming UDP stream for receiving input data from the Stream Processor
- An incoming UDP stream for receiving context information
- An Outgoing UDP stream for raising Reputation Alerts

The following figure shows how this is achieved and related with the other RERUM components:

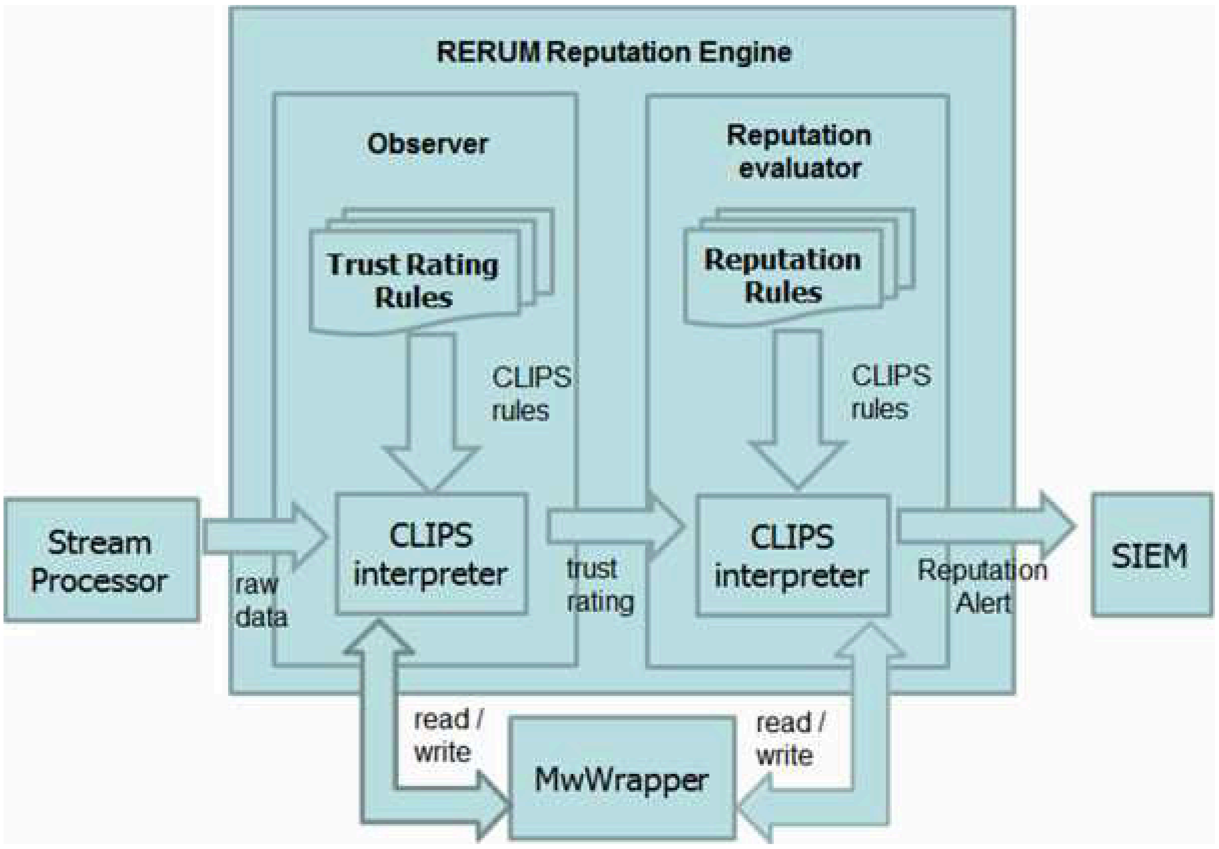


Figure 26: Processing of reputation

As the figure shows, the Observer and the Reputation Evaluator follow an identical structure. That is, both of them are composed by a CLIPS interpreter that executes its corresponding CLIPS program. In both cases, the CLIPS programs receive its input from two UDP data streams and produce its output to the outside as Data Streams as well.

From an architectural point of view, the main difference is there may be as many instances of observers as sets of trust rating rules, each of one representing a different observer, while the Reputation Evaluator will normally be a single one collecting all trust ratings. From a functional point of view, the difference is the program they run. In the case of the Observer, the program to be run is the trust rating rules, which are written in a declarative language and produce trust ratings from the raw data. In the case of the Reputation Evaluator, the program to be run is the reputation rules, which produce the Reputation alerts after consolidating the trust ratings from the different observers.

3.5.2 Prototype of Trust Evaluation Model (CLIPS Rule Engine)

This section presents the POC prototype for evaluating the Trust Evaluation Model. As a POC, it is not integrated in RERUM and hence it includes one slight adaptation over the Trust Model that does not affect the concept but decreased considerably the effort needed to provide the prototype. This adaptation consists in the input contained in the input stream, which includes some additional data that could be difficult to be provided by a generic system.

In this section we demonstrate the usability and expressiveness of the CLIPS rules for the purposes of implementing our trust model. Our experience with CLIPS can be summarized as follows:

- The CLIPS language is expressive enough for all types of rules that we have envisioned.
- This is due also in particular to the integration of Python, which is ideal to make numerical calculations, process input and output (streams/memory/files) and synchronize processes.
- The CLIPS rules are easy to write, read and modify. An engineer with no previous experience in expert systems at all and in CLIPS in particular should be able to write and modify rules after one day of practice.
- The CLIPS rules that we have used all follow "standard patterns". This is not necessarily always the case (in particular when using complex hybrid models to represent the physical processes) but we believe that in most of the cases this will happen. The standard patterns are easy to learn and reuse.

3.5.2.1 Trust Rating - Observers

As represented in Section 2.4.3, an observer can monitor one or more streams and is estimates the trust-rating those streams. To estimate the trust rating of a stream we need a structured input data that the stream provides to an observer. For evaluation purposes in RERUM, a stream shall provide the following data (in any required format that the observer can decode it):

```
Stream: {
    stream-identifier (StrN)           =    "stream 1"

    timestamp (timC)             =     "UTC timestamp"

    value (valC)                 =     "20.0"

    service (SrvN)               =     "temperature"

    geographical-location (geoL) =     "munich"

}
```

The stream data is further processed to calculate statistical information such as stream-average (aveS), stream-standard-deviation (stdS), etc., based on the previous or historical data of that stream and it is briefly explained in Section 2.4.5.

Each observer is defined to do one or more specific tasks or more specifically to match one or more rules for the streams it monitors. Those rules may differ depending on the streams the observer monitors. To model the observer to estimate the trust-ratings within the CLIPS environment, we define an environment for an observer as the following:

- a-observer-stream-rule pair
  If a stream-x is observed by an observer-y for a specific rule-n, then they form an observer-stream-rule pair. This pair is used more often in matching the rules within the CLIPS engine.
- Rule-identifier
  Each rule may have a rule-identifier that can have different threshold value (conditions) for different observer-stream-rule pair
- Rule specific data
  If a stream provides the temperature service, it is appropriate to validate the temperature data of the stream against the historical temperature data for that location on a given time period of the year. Therefore, the historical-temperature data is an example of rule-specific data which needs to be inserted into the CLIPS rule matching system.

Once, the observer has all the data/information of a stream it checks if the rules match against the stream's current statistics, based on the matched rules the observer calculates the trust rating of the stream. In the following, we briefly describe the trust-rating rules used by the observer:

3.5.2.2 A-priori conditions (valC-mima)

A-priori conditions are defined for a stream with a rule-identifier. An a-priori condition may have two threshold values for a given stream i.e., a lower-threshold value and an upper-threshold value. A stream can be checked against two different a-priori conditions with different rule-identifier. An a-priori rule (valC-mima[3]) is matched when a stream's value (valC) is outside the threshold limit (minA or maxA) defined in the following rule.

$$!\,(minA < valC < maxA)$$

The corresponding CLIPS rule is presented in Section 2.4.9.

3.5.2.3 Standard-deviation (aveE-stdE)

A stream's statistical data such as exponential-moving average (aveE) and standard-deviation (stdE) are important to find any minor change in the RERUM system and it could be a faulty sensor in the system. An observer monitors the exponential moving average of the stream through the rule (aveE-stdE). The aveE-stdE rule has two additional constants: the multiplying-factor (zed); and, the a-apriori average (aveA); both are rule specific data for a given stream-observer-rule pair.

The aveE-stdE rule is matched if the following condition is not matched and its CLIPS representation is available in the appendix Section 2.4.9.2.:

$$! \ [avgA - (zed * stdE) < avgE < avgA + \ (zed * stdE)]$$

3.5.2.4 Historical-value (valC-valH/avgH)

The valC-valH/avgH[4] rule checks the historical values against the stream's value/statistics based on location and at a given time. For instance, a temperature service stream provides location and time information for the given data, and the historical temperature data for a location (for example, Munich) can be retrieved from the world weather information service [WWIS_Mun-2011]. The historical data needs to be stored in the CLIPS system as facts before the system is initiated. An example of such fact (historical data) for the month January is shown below:

$$(temp - hist\ (daily - mean\ 0.3)\ (timH\ 1)\ (rec - high\ 17.2)\ (avg - high\ 2.7)(rec - low\ - 30.5)\ (avg - low\ \ - 3.7))$$

This rule is matched if all the following conditions are true and its CLIPS representation is available in the appendix Section 2.4.9.3:

- An-observer-stream-rule pair: An observer is monitoring the stream (stream-identifier) with rule identifier "valC-valH" or "valC-avgH"
- The stream's location and time matches the historical data's location and month.
- The stream's value (either current value or stream's static average) is not within the threshold of historical data as expressed below

$$\begin{gathered} !\,(\,minValH\ <\ valC\ <\ maxValH) \\ or \\ !\,(\,minAvgH\ <\ avgS\ <\ maxAvgH\,) \end{gathered}$$

3.5.2.5 Reputation value - Reputation engine

The reputation engine as described in Section 2.4.9.4, receives trust-ratings from the observer, user or an admin and estimates the reputation-value of that observer.

The observer estimates stream-trust-ratings of streams that represent the real numbers between 0-10 along with supplementary information such as stream-identifier, timestamp and observer-identifier. The reputation engine also receives trust-ratings from administrators about the observers also known as observer-trust-ratings. The stream and observer trust ratings received by the reputation engine are shown in the following code below:

observer-trust-ratings:{
- Admin –identifer(admN) = “admin1”
- Observer-identifier (obsN) =”observer1”
- Trust-rating (obsTR) = “5.7”
- Timestamp (timC) = “UTC timestamp”
}
	stream-trust-ratings:{
	- stream –identifer(admN) = “stream1”
	- Observer-identifier (obsN) =”observer1”
	- Trust-rating (strTR) = “5.7”
	- Timestamp (timC) = “UTC timestamp”
	}

3.5.2.6 Reputation engine rules

The reputation engine calculates the reputation value of a stream based on the trust-ratings provided by observers and the administrator. For instance, a stream can be monitored by one or more observers that can result in different stream-trust-ratings of a stream. Let’s assume that the stream x is monitored by observers $O_1, O_2 \dots O_n$ each observer produce a stream-trust-rating ($\dot{\mathrm{T}}$) for stream x represented as $x\dot{\mathrm{T}}_1, x\dot{\mathrm{T}}_2, x\dot{\mathrm{T}}_3 \dots x\dot{\mathrm{T}}_n$. In the similar way, an administrator (“admin”) shall provide observer-trust-ratings ($\dot{\mathrm{A}}$) for each observer as $O_1\dot{\mathrm{A}}, O_2\dot{\mathrm{A}} \dots O_n\dot{\mathrm{A}}$

The reputation value of the stream is a weighted average of the trust-ratings produced by both the observers and the administrators and can be expressed as follows:

$$reputation\ of\ stream\ (x) = \frac{x\dot{\mathrm{T}}_1 * O_1\dot{\mathrm{A}} + x\dot{\mathrm{T}}_2 * O_2\dot{\mathrm{A}} + \cdots + x\dot{\mathrm{T}}_n * O_1\dot{\mathrm{A}}}{sum(O_1\dot{\mathrm{A}}, O_2\dot{\mathrm{A}} \dots O_n\dot{\mathrm{A}})}$$

The clips rules for the reputation engine are shown in the appendix Section 2.4.9. It includes three sub-rules: first, to create an instance of the reputation value of a stream; second, to update the reputation value based on different observer-admin pair; third to finally calculate the weighted average of the observer-trust ratings.

3.6 Correlating Reactions to Reputation Alerts

This section explains how the Reputation Alerts that reach the Reacting engine can be correlated with each other to produce more refined alerts. Though strictly speaking, this is an optional step and the processing of reactions could be performed without this step, these correlation is explained before the processing of the reactions because it can generate additional Reputation Alerts that are subject to be reacted as well and hence it affects the input of that processing.

As it will be explained in Section 3.8, in RERUM, the processing of the reactions to the data is carried out in the SIEM component of RERUM (which is part of RERUM architecture but not of the MW).

The SIEM works with a declarative language for processing events named EPL, which stands for Event Programming Language. EPL allows defining new events from previous ones combining them with formal logic. This allows for correlating different Reputation Alerts from different Reputation Evaluators as explained in Section 2.5. In practice, this means that for processing of the example provided in that section it would be necessary to define in the SIEM those EPL rules that implement it, such as:

EPL Statements filtering events by source and type:

```
insert into UserReputation select * from rerumSchema_default where
(plugin_id=15001 AND userdata1 = "user")
insert into ExpertReputation select * from rerumSchema_default where
(plugin_id=15001 AND userdata1 = "expert")
```

1. EPL Directive that generates a LowGlobalReputation alarm event when conditions are met:

```
insert into LowGlobalReputation select * from pattern [every
(a=ExpertReputation (a.userdata2 < 2)) AND (b=UserReputation
((b.userdata2 < 2) AND (b.src_ip = a.src_ip))) where
timer:within(300 sec)]
```

That EPL rule would create an additional Reputation alert that can be reacted later. The following figure shows the process:

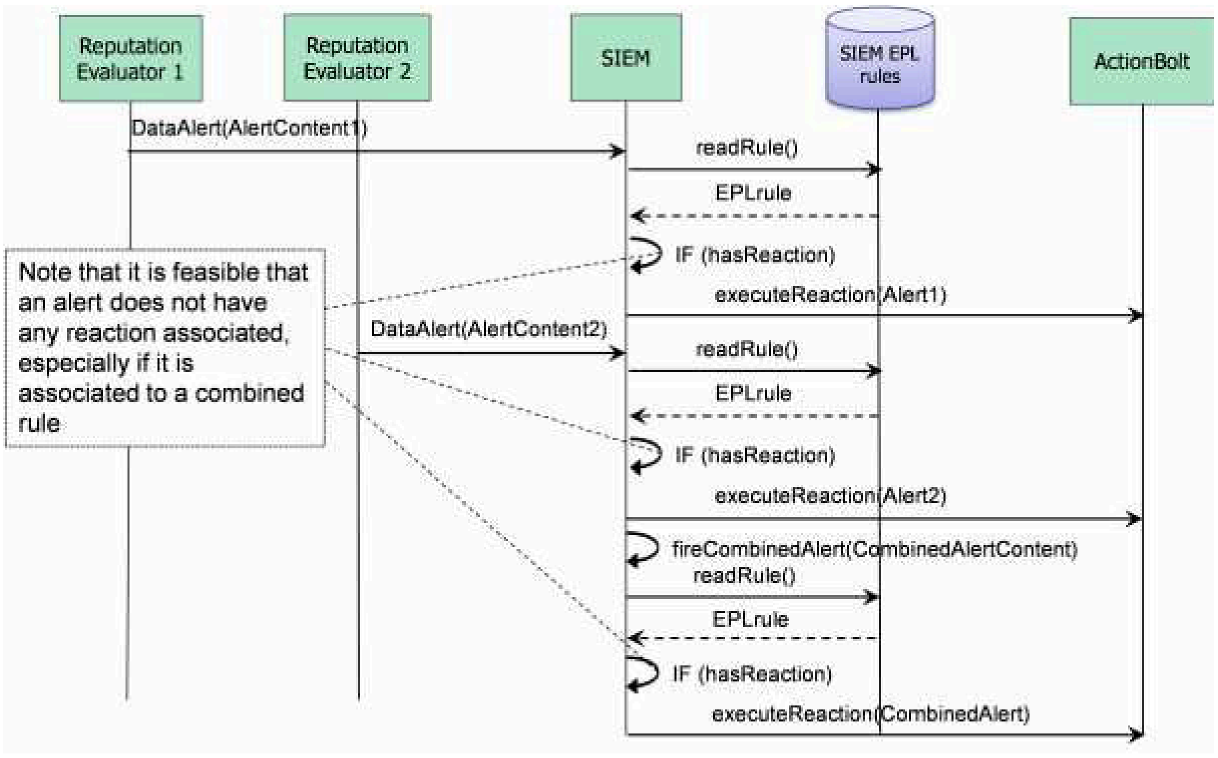


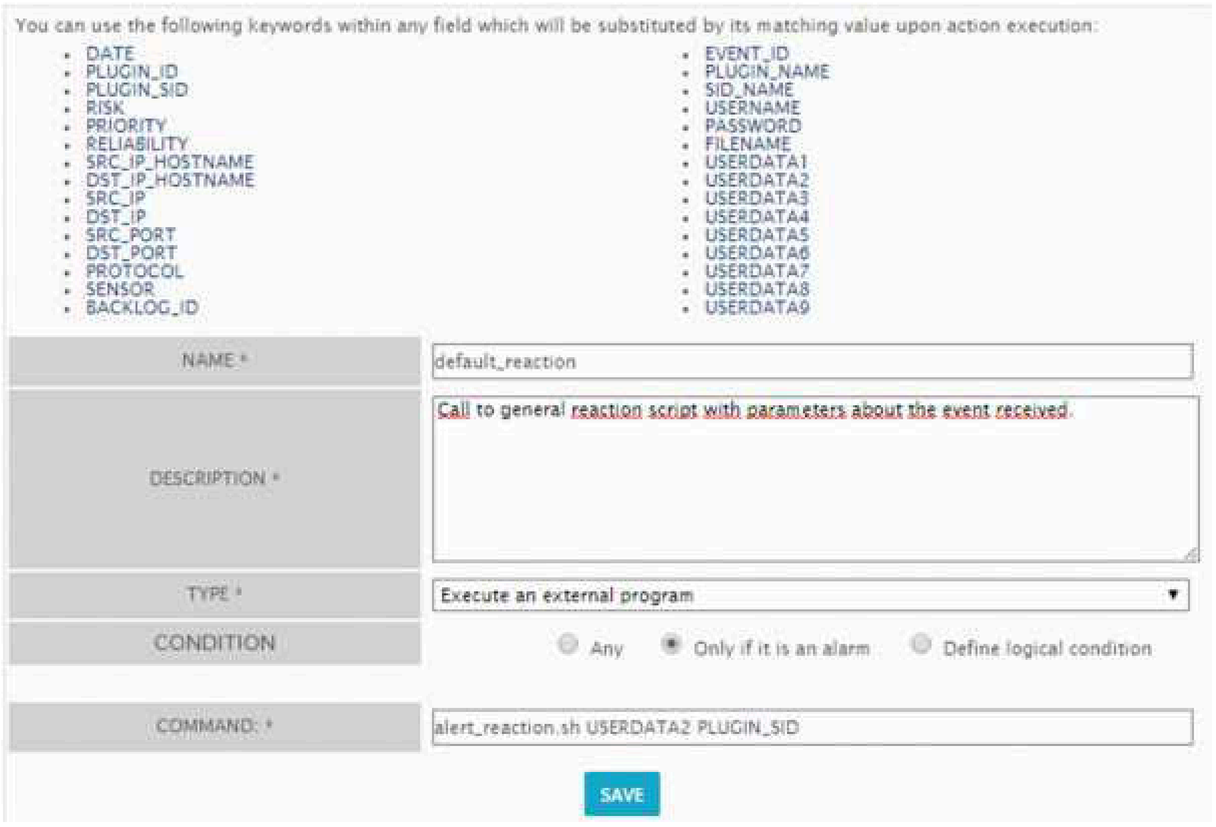
Figure 27: Firing correlated alerts

As the figure shows, each Reputation Alert has its own processing associated, with its own reaction triggering, but it is feasible that a Reputation Alert has no reaction associated, especially if it is associated to a combined alert, because it could happen that a part or all the Data Events that triggers a Combined Alert are reacted jointly in the reaction for the Combined Alert. In any case, the combined Alert is considered to be a Reputation Alert in all aspects and hence it follows exactly the same processing of the rest of the Reputation Alerts. That is, it retrieves their corresponding EPL rules and executes its related actions if it has any.

Finally, it would be feasible that an incoming Reputation Alert would not have any EPL rule associated. Such alerts are ignored because it is in the EPL rule where they are defined for the SIEM component.

3.7 Processing of Reactions

As explained in Section 2.5.2.1, many systems do not allow providing the reactions in a formal language that allows to execute them later, but provides a GUI for it instead. This is the case of the SIEM provided in RERUM, which is based on Apache STORM [STORM 15]. More specifically, a web page allows the user to associate one of a given set of predefined reactions to a concrete alert already defined in the system. The following figure shows the page:



You can use the following keywords within any field which will be substituted by its matching value upon action execution:

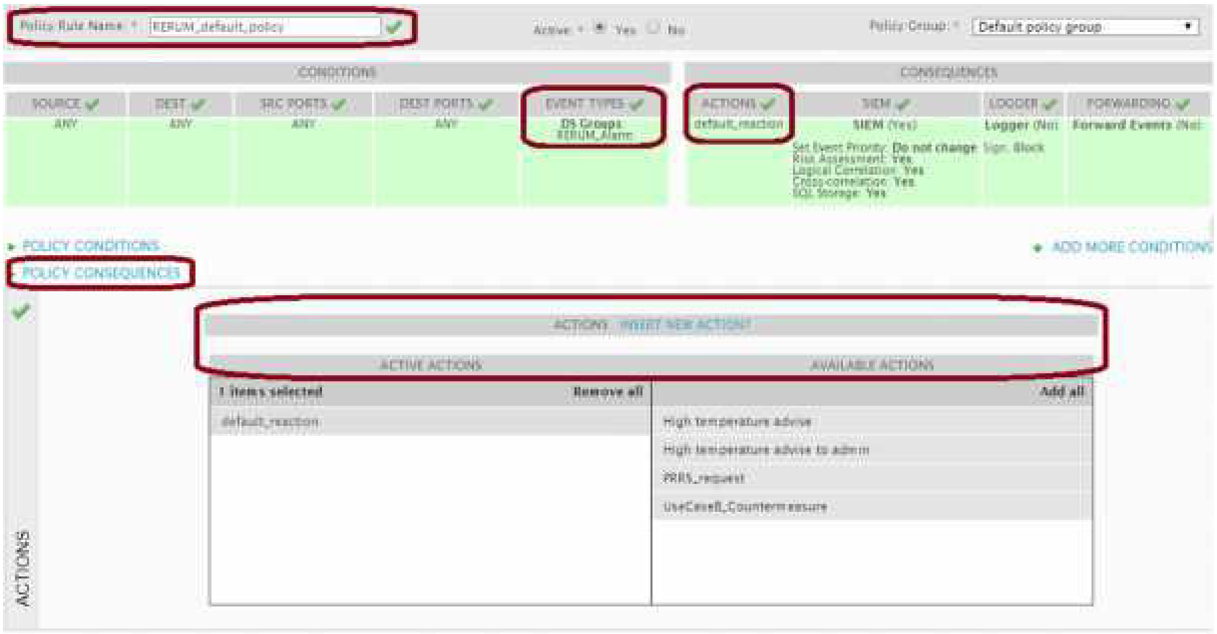
- DATE
- PLUGIN_ID
- PLUGIN_SID
- RISK
- PRIORITY
- RELIABILITY
- SRC_IP_HOSTNAME
- DST_IP_HOSTNAME
- SRC_IP
- DST_IP
- SRC_PORT
- DST_PORT
- PROTOCOL
- SENSOR
- BACKLOG_ID

- EVENT_ID
- PLUGIN_NAME
- SID_NAME
- USERNAME
- PASSWORD
- FILENAME
- USERDATA1
- USERDATA2
- USERDATA3
- USERDATA4
- USERDATA5
- USERDATA6
- USERDATA7
- USERDATA8
- USERDATA9

NAME *	default_reaction
DESCRIPTION *	Call to general reaction script with parameters about the event received.
TYPE *	Execute an external program
CONDITION	Any [✓ Only if it is an alarm] Define logical condition
COMMAND: *	alert_reaction.sh USERDATA2 PLUGIN_SID

SAVE

Figure 28: Setting up new events in the SIEM


Policy Rule Name: * RERUM_default_policy

Active: * [✓ Yes] No

Policy Group: * Default policy group

CONDITIONS					CONSEQUENCES			
SOURCE	DEST	SRC PORTS	DEST PORTS	EVENT TYPES	ACTIONS	SIEM	LOGGER	FORWARDING
ANY	ANY	ANY	ANY	DS Groups RERUM_Alarm	default_reaction	SIEM (Yes) Set Event Priority: Do not change Risk Assessment: Yes Logical Correlation: Yes Cross-correlation: Yes SQL Storage: Yes	Logger (No) Sign: Block	Forward Events (No)

POLICY CONDITIONS

POLICY CONSEQUENCES

ADD MORE CONDITIONS

ACTIONS INSERT NEW ACTION

ACTIVE ACTIONS	AVAILABLE ACTIONS
1 items selected — Remove all	Add all
default_reaction	High temperature advise
	High temperature advise to admin
	PRRS_request
	UseCaseB_Countermeasure

Figure 29: Assigning actions to events in the SIEM

The current supported mechanisms for specifying the reaction are the following ones:

- Sending an email to a given email address
- Sending a message to a given ip address and port. The meaning of the message will depend on the program that is listening on that ip and port
- Executing a script

Though it might seem that the third option could allow the execution of the generic mechanism stated in 2.5.2.4, actually it has a serious drawback: The execution of a command needs to create a session in the Operating System, issuing and interpreting the command, load the program in memory, execute it and release the program and the session. Though it could be possible to reuse the same session for all the reactions, the loading of a program in memory to execute it is usually very costly in terms of CPU and execution time. For instance, executing a java program this way would require to create a JVM and load all needed classes, including the Java classes each time the reaction is triggered. This is completely undesirable, especially in very large systems that could need to deal with many alarms.

Thus, it is necessary that the component/s that are able to execute the reactions remain in memory in some way so they do not need to be started from scratch each time that a reaction is triggered. The solution of sending a message to a given ipAddress and port could suit this need, but it has the drawback of forcing the classes that implement the reaction to provide a server socket to listen to the petitions. For instance, if we have a service for updating authorization policies of a service, that service will have to be listening on a given ipAddres and port, such as localhost:8080, and the message sent will need having the structure imposed by the listening service, such as the content of the Reputation Alert as a sequence of characters.

For this reason, RERUM has opted for the approach of defining a common interface to be implemented by all reacting classes and providing the name of the implementing class when associating the reaction to an event. This interface would contain a method void executeReaction(DataAlert alert)[5]. That is, a method that receives the alert that causes it, which must contain the information needed for the reaction to know what to do. What the SIEM will need to do then is simply loading that class dynamically in memory and invoke the corresponding method.

Note, however, that the current implementation of the SIEM does not allow performing this way; however, the changes needed to implement this behaviour would be quite straightforward. The following figures show the classes involved for registering the reaction and executing it.

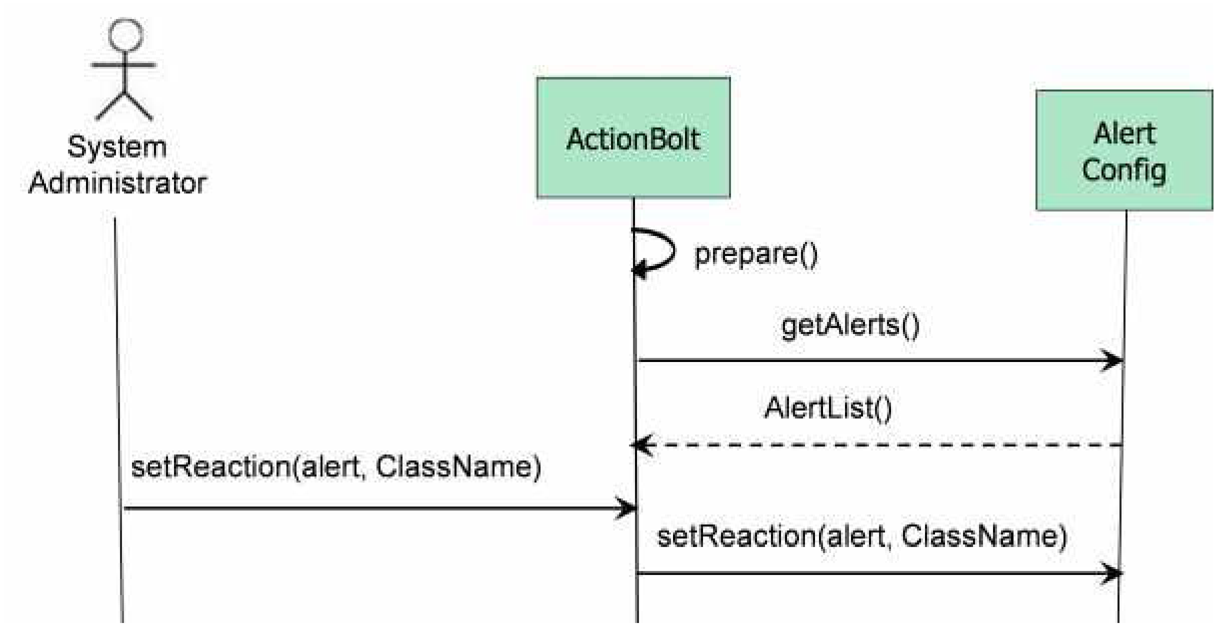


Figure 30: Setting up a reaction for an Alarm

In STORM, the execution of the actions associated to the alerts is managed from the Class ActionBolt, which is always initialized at startup through the method prepare, which reads the configuration of the alerts on advance through the class AlertConfig.

As the figure shows, the System administrator is meant to select an alert from a list of alerts presented by the ActionBolt, which obtained them previously by the AlertConfig. With this list of alerts, the System Administrator can indicate the fully qualified name of the implementing class to the Reaction Manager, which will assign it to the Alert Manager.

And the following diagram shows how the reaction is executed:

[5] Strictly speaking, STORM Works internally with objects named tuples that contain the information of the event, but we have preferred to refer to them as Alerts for a matter of readiness

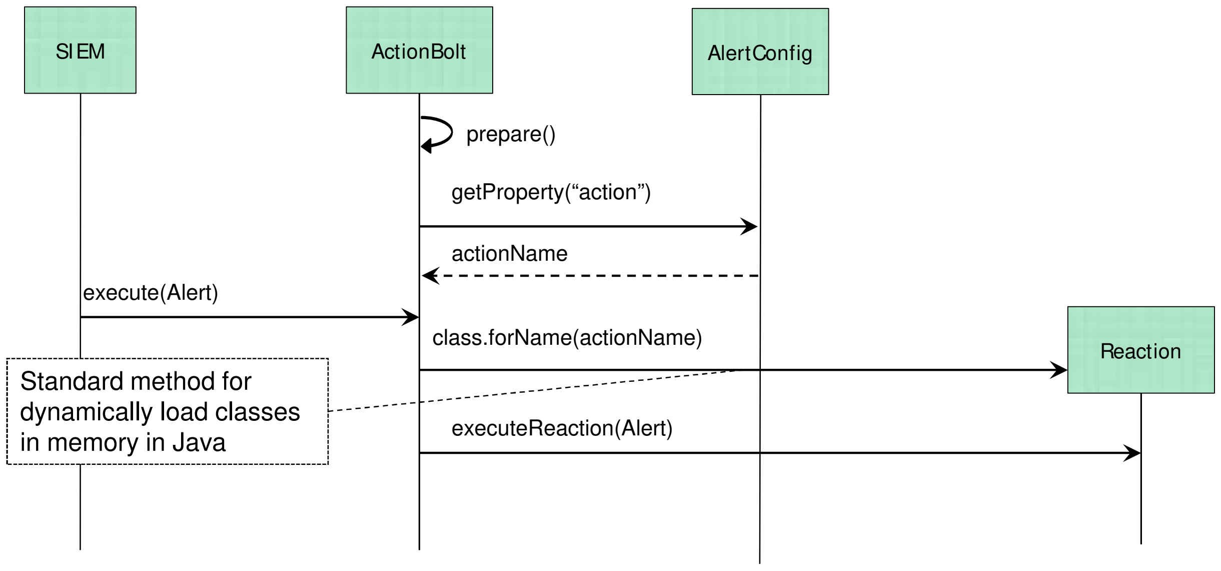


Figure 31: Executing reactions in RERUM

As the figure shows, when the SIEM wants to execute the action associated to an alert, it passes it to the ActionBolt, which had previously retrieved from the AlertConfig the name of the class implementing the Reaction interface that has to execute the reaction. With that name, the Action Bolt executes Class.forName(reactionName), which is the way to load classes explicitly by his name in Java. After this, it is ready for invoking the Reaction object passing it the information regarding the alert with the message executeReaction(Alert).

3.8 Incorporating the Reputation Access Profiles to the Authorization Process

As explained in Section 2.5.4 there are several possible methods for incorporating the Reputation Access profiles to the Authorization process, including:

a. Making the evaluation results to be accessible by the Authorization components through a PIP that consults them and that is executed in each request;
b. Sending the evaluation results to the authorization components each time they change and
c. Producing a set of additional policies which already have that information embedded to reject the access to the services that are not trusted

All these options have their advantages and drawbacks. Option a) can seem more elegant than option b) because it would let to have a single global policy for the whole system that would reject the access to the blocked services if its reputation level is lower than a threshold. This looks simpler than having one rejecting policy per each non-trusted service.

However, option a) has a very important drawback: Due to the federated nature of the MW, retrieving the Reputation Evaluation would require at least two additional remote invocations to the MW, one to the GVO discovery and another one to the GVO Registry. And this would have to be carried out per each request. In contrast, working with policies that have that information already embedded removes that need because the policy already contains all the information to take the decision.

In the concrete case of RERUM, the authorization policies are designed to work without a PIP precisely for avoiding this performance problem. Even the retrieval of user attributes is performed once at the start of the session (see deliverable D3.1 for more details) and included in a security token in the request to avoid having to make additional requests to an Identity provider. That is, the philosophy of the RERUM authorization components is to boost performance by minimizing the number of network accesses during the processing of each request.

For this reason and especially because RERUM authorization components were originally designed to work without a PIP, RERUM opts for method c) that is, generating policies with the information embedded (see Section 2.5.4.3 for details on why this is important). More specifically, what RERUM does is assigning a reaction to the Reputation Alerts that decides whether to create a rejecting policy for a given service or removing any existing one for that service. This decision is made comparing the values of the Reputation Evaluations included in the alert with a threshold value for each purpose read from a configuration store. If there is any Reputation Evaluation for a service whose value is less than the threshold defined in the configuration for that given purpose, then a condition is added to a rejecting policy that checks whether the request wants to check for that criterion. On the contrary, if no Reputation Evaluation of the alert is less than their respective thresholds, then any existing rejecting policy for that service is removed.

The next figure shows the classes and messages involved in this treatment.

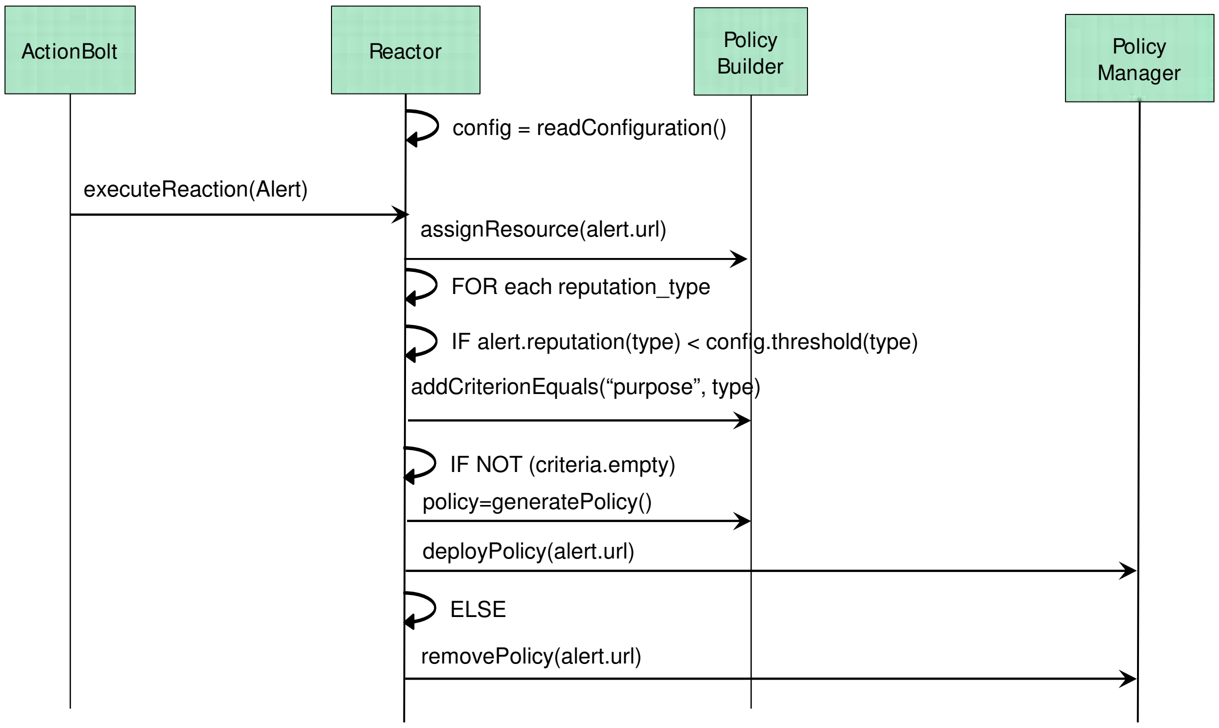


Figure 32: Producing a static policy for rejecting access to non-trusted services

As the figure shows, the configuration is read in advance to avoid the need to read it over and over again. The Reactor uses the class Policy Builder for creating a new policy and assigning it criteria. It starts by assigning the resource of the policy to the url contained in the alert. After that, it goes through all possible evaluation criteria comparing them with their respective thresholds obtained from the configuration. If any of them is lower than its assigned threshold then it adds a condition to check the purpose of the request for that value. Finally, if there is any criterion assigned it deploys the generated file. In any other case, it removes the file.

It is not a coincidence that the figure looks the same as Figure 14: Embedding Reputation Evaluations in Access Polices from the conceptual IoT model. On the contrary, the Authorization Components of RERUM were already prepared for creating and deploying dynamically policies, but do not include a PIP. This is why RERUM has opted for this option.

The reason why the generated policies are checking the purpose of the request is because the rejection has been decided per each purpose and therefore it is necessary to check that the purpose of the request is included in any of the ones that cause the rejection.

3.9 Processing Reputation for Users at Service Level

Though strictly speaking, the reputation of users is not in the scope of this document, its integration with the authorization process is interesting from a research point of view, because it would empower the administrators by letting the system to take into account the reputation of the user when performing the authorization decision. This empowerment could help very much in an IoT world because the dynamic nature of the IoT causes the appearance of new (and therefore unknown) purposes making necessary the use of mechanisms able to take the decision for such situations automatically according to the criteria defined by the users.

Hence, as the key point is not evaluating the reputation of the users, but being able to incorporate that evaluation to the authorization decision, what RERUM has made is defining and implementing a system able to take that evaluation into account and providing a very simple reputation evaluator as a proof of concept. In concrete, this simple reputation evaluator is simply a reputation configuration store that can be set by the administrator per each user defined in the system. The evaluation consists mainly in crossing the user against its reputation that was read from the configuration and serve that value. Secondarily, it also involves checking the security token that contains the identifier of the user to check that it has not been tampered. The following diagram shows the process:

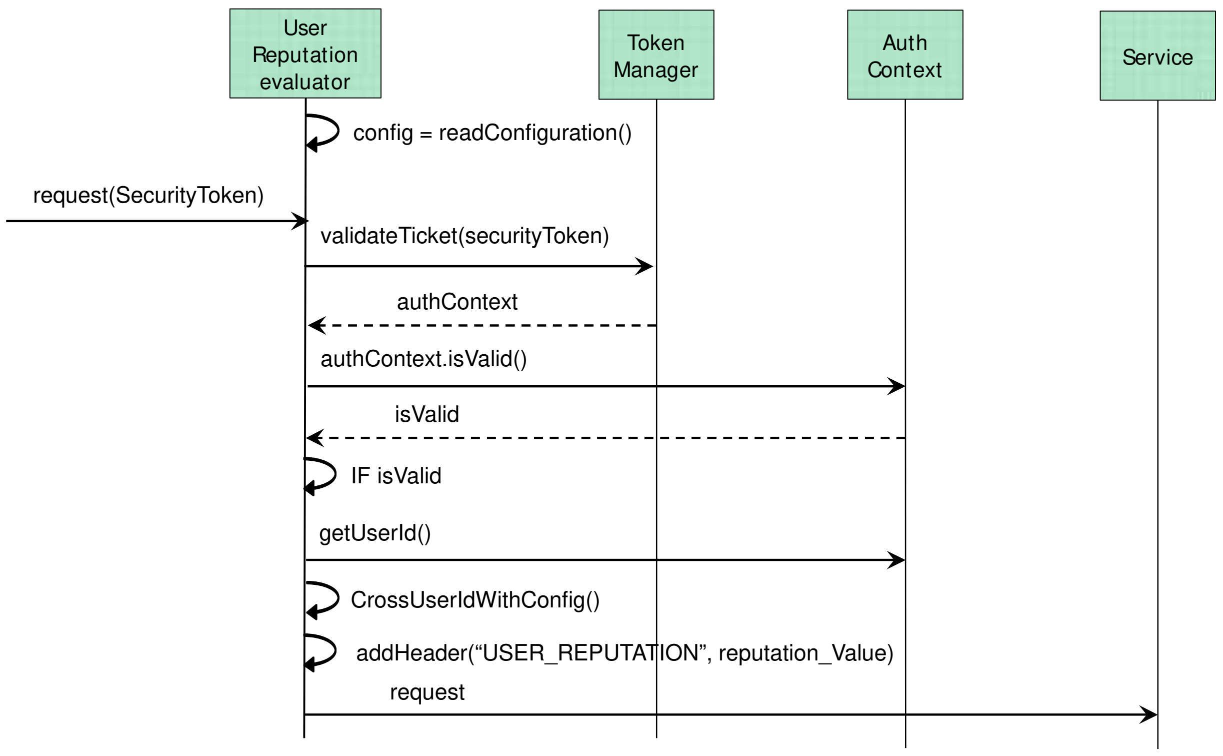


Figure 33: Evaluating the reputation of the users

As the figure shows, the configuration with the reputation of the users is read in advance to avoid having to read it for each request. When a request with its corresponding security token arrives, the first step is validating the integrity of the ticket with validateTicket and authContext.isValid. More details on the security token and how it is included in the request can be found on D3.1 Chapter 4 [RD3.1]. If the ticket is valid (not tampered and signed by a trusted party), then the User Reputation Evaluator crosses the userId against the configuration to get its reputation and includes it in a header named "USER_REPUTATION". Finally the request is forwarded to the service. But actually the request will not hit directly the service because it will be intercepted by the authorization components as it will be explained soon.

Hence the real focus of this task section is defining the mechanism to incorporate this evaluation to the authorization process. The way that RERUM achieves this is inspired in the concept of chain of filters, which is already in use in widely used frameworks such as Java Enterprise Edition (JEE) [Oracle 2].

In short, the concept of filters means that before a request reaches a service, it is subject to suffer a series of transformations that are transparent to the service but can alter its input. Moreover, this concept is even applicable to the output. That is, the output of a service can be subject to an output filter to alter it before it reaches the requester. An essential feature of filters is they are meant to be cumulative. That is, filters are supposed to be able to be executed one after another and each filter is meant to be transparent to the other ones. The only constraint to this is though the existence of input and output filters is optional, if they exist, the order of execution of input and output filters should be symmetrical. That is, if we had input filters named 1, 2 and 3 in this order, then the output filters should be 3, 2, and 1 in this order. A possible example of a filter could be the execution of decryption and encryption, being the decryption the incoming filter and the encryption an output filter.

RERUM is already using this concept in the authorization components, which, in fact, act like a filter that inspects the request and decides whether to grant access to RERUM or not. The following figure shows how the filters are designed to act in RERUM and how the user reputation component is incorporated to the chain as an additional filter.

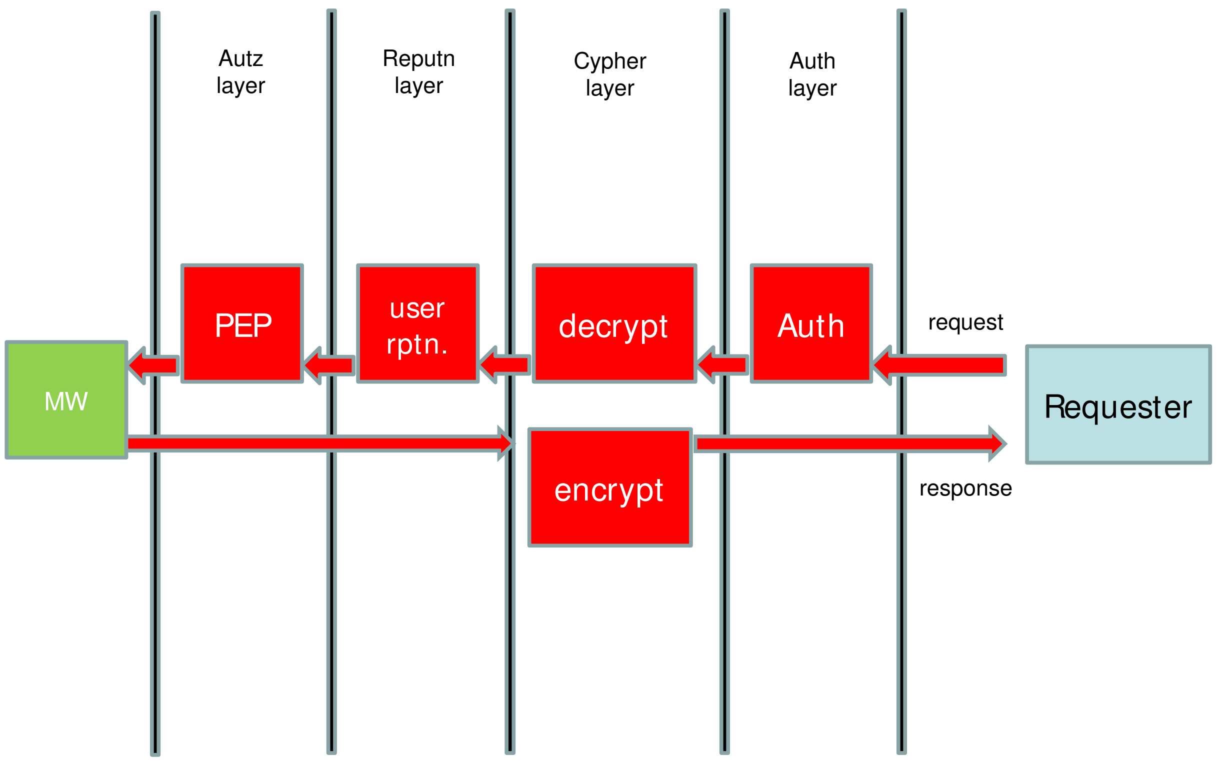


Figure 34: Inserting the evaluation of the user reputation in the chain of filters of RERUM

As the figure shows, the request that arrives the Authorization Layer Component, that is, the Policy Enforcement Point, the PEP, is not the original request, but the one that has passed previously through the Authentication, decryption and user Reputation Evaluation, in this order. And each of these steps has potentially modified the request on its own way. In concrete, the user reputation component has added a new header named 'USER_REPUTATION' with a numeric value on it. Now, all that is left is to provide a proper XACML file for the resource that we want to be influenced by the Reputation of the user. For instance, if we want to ban access to a given resource to any user with a reputation value that is lower than 2, then we could simply use the policy builder to generate the proper policy and deploy it, as shown in the following figure:

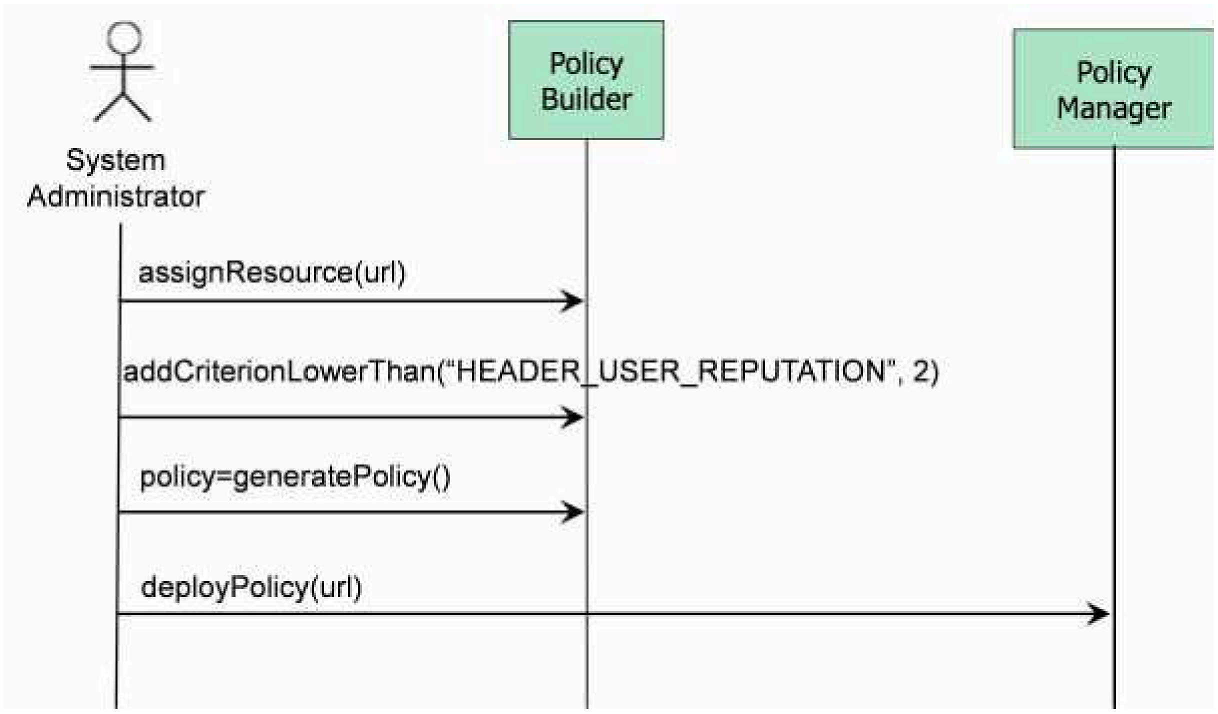


Figure 35: Creating a policy that takes into account the Reputation Evaluation of a user

As the figure shows, generating a policy file that takes into account the reputation of the user is very straightforward. It is simply a matter of creating a policy, assigning its corresponding resource and adding the condition that checks the value against the desired value and finally invoking the Policy Manager to deploy it. The only non-trivial point is taking into account the prefix HEADER for those fields that are included in an http header.

3.10 Trust Reputation Evaluation for Devices at the Network Level

This section introduces a RERUM specific observer at network level, explaining how it fits with RERUM, the types of attacks it contemplates and providing and evaluation of its performance.

3.10.1 Relation with the Trust Model

This section presents the way to integrate the calculation of the reputation of devices in RERUM, by using an observer that monitors the network statistics together with the reliability of the measurements that the devices gather. As it can be seen in Figure 36, the Network Observer can be an additional instantiation of an observer that is connected to the RERUM reputation engine, providing trust ratings for the RERUM Devices in order to enable the calculation of their reputation. Any type of CLIPS rules and reputation rules (as described in the previous sections) can be used for the calculation of the device reputation. In the subsections below we only present the metrics via mathematical formulas, but they can be translated easily to CLIPS rules in order to be added in the trust engine of RERUM.

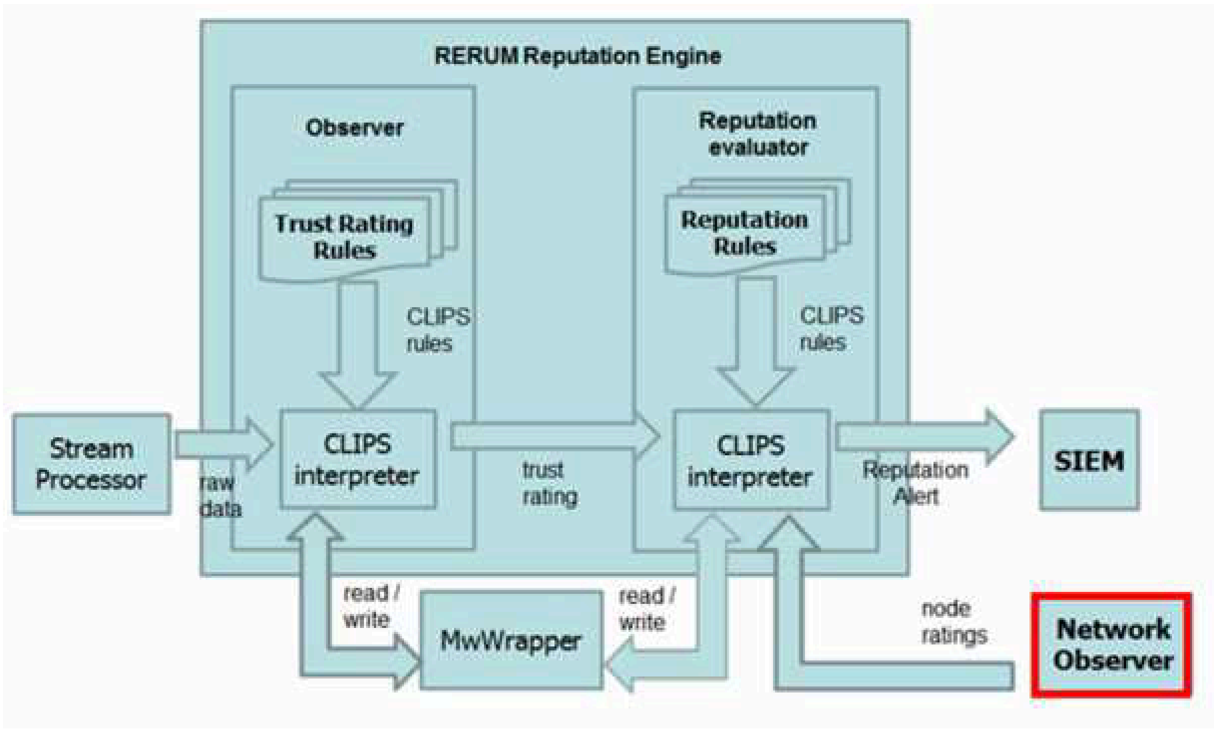


Figure 36: RERUM Reputation engine with a network observer

3.10.2 Device Reputation

In IoT, the evaluation of the reputation of a device (or node) should be based on multiple criteria that can be grouped into three categories:

a. **behavioral reputation**: it is related with the past behavior of each device as it is recorded by its neighbors, and can relate with metrics against specific types of attacks as seen in Table 1. Each of these criteria/metrics can be used for assessing the behavioral trust of the nodes. For example, when one node selects one neighbor to send the packet, then it can go into promiscuous mode to see if this node indeed forwards the correct packet, or if it alters it or drops it, and then it can evaluate the behavioral trustworthiness of this node. In RERUM's view, it can be assumed that the metrics (1), (3), (5), (6), (11) are the most important ones, while the others can be used in specific cases. Of course, different weights can be used for each one of these metrics, thus resulting in an expression like [ZAHA 10]:

$$BR_{i,j} = \sum w^s \times BC^s{}_{i,j} \text{, with } \sum_{s=1}^{n} w^s = 1 .$$

No	Trust Metric	Monitored Behavior	Attack addressed
1	Data packets forwarded	Data message/packet forwarding	Black-hole, sinkhole, selective forwarding, denial of service, selfish behavior
2	Control packets forwarded	Control message forwarding	Control/routing message dropping

3	Data packet precision	Data integrity	Data message modification
4	Control packet precision	Control packet integrity	Sybil and any attack based on routing protocol message modification
5	Availability based on beacon/hello messages	Timely transmission of periodic routing information reporting link/node availability	Passive eavesdropping, selfish node
6	Packet address modified	Address of forwarded packets	Sybil, wormhole
7	Cryptography	Capability to perform encryption	Authentication attacks
8	Routing protocol execution	Routing protocol specific actions (reaction to specific routing messages)	Misbehaviors related to specific routing protocol actions
9	Battery/lifetime	Remaining power resources	Node availability
10	Consistency of reported values/data	Consistency of sensing results, reported values (e.g. energy, humidity)	Compromised nodes
11	Sensing communication	Reporting of events (application specific)	Selfish node behavior at application level
12	Reputation	Trust value observed by third parties	Bad mouthing attack (False reputation claims)

Table 1: Neighbor behavior monitoring [ZAHA 10]

b. **communication reputation**: it is related with communication capabilities of the node and can be evaluated mainly via the link quality statistics of the nodes. As also mentioned in RERUM Deliverable D4.1, it is necessary to take into consideration a link quality metric that will be used in conjunction with the trust metric for the optimal path calculation of the routing algorithm. In the following, we use the *expected transmission count* (ETX) metric to quantify the reliability of the link between any two nodes. ETX is one of the most widely used routing metrics due to its simplicity and computational efficiency that has been proven in a large number of applications. Concretely, the ETX calculated for node *i* by node *j* is defined as:

$$ETX_{i,j} = \frac{1}{f_{i,j} \cdot r_{i,j}},$$

where $f_{i,j}$ is the forward delivery ratio, i.e. the measured probability that a packet sent from node *i* is received by node *j*, and $r_{i,j}$ is the reverse delivery ratio, i.e. the measured probability that the acknowledgement packet from node *j* will be received by node *i*. Thus, ETX expresses the average number of transmissions needed for a packet to successfully reach its destination in cases when there are transmission failures due to degradation of link quality (e.g. interference, collisions, etc.).

c. **service-based reputation**: it is related with the evaluation of the reputation of the node with regards to the measurements it gathers and transmits for each one of the services it provides. Here, the evaluation of the service trustworthiness of the node can be done by identifying the inaccuracies in the measurements reported by each node, when compared to a statistical analysis of its time series (i.e. the deviation from the average value reported in X previous timeslots) and/or compared to the measurements reported by node that are providing the same service, and are associated with the same VE as the device that we are evaluating. For this type of trustworthiness there is extended literature for i.e. outlier detection in WSNs, i.e. see the references in [GUGA 14].

What changes within RERUM with regards to the state of the art is that the nodes are assumed to provide multiple services, so when evaluating the service-based trustworthiness using any of the outlier detection techniques, the evaluation has to be done for each service separately. So, we assume to have x=1..N service-based trust ratings for each node, where N is the number of services it provides. Each of these trust values has to be evaluated separately, since inaccuracies in one service do not necessarily mean that all services provided by the node are inaccurate. If there is an association of a service to a specific sensor on board of the node, there could be a joint evaluation of the trust values of the services provided by that sensor in order to identify if this sensor is malfunctioning. The cross-evaluation of the trust values of all services can only give a hint if the node is tampered with/hacked so that it reports intentionally false measurements.

So, we can have trust metrics as below:

$OSTM_i = \sum_{S_x=1}^{N} w^{S_x} \times STM^{S_x}{}_i$, where OSTM is the overall service based trust metric and the STM is the trust metric for each one of the provided services Sx.

3.10.3 Network Model

We use the three-tier model of the RERUM architecture as depicted in Figure 37. $Tb_{i,j}$ is assumed to be some trust rating reported by each of the devices, Tb_{i_GWx} is the trust rating for the device *'i'* calculated at the GW_x and T_{br} is the respective reputation calculated at the Reputation Manager.

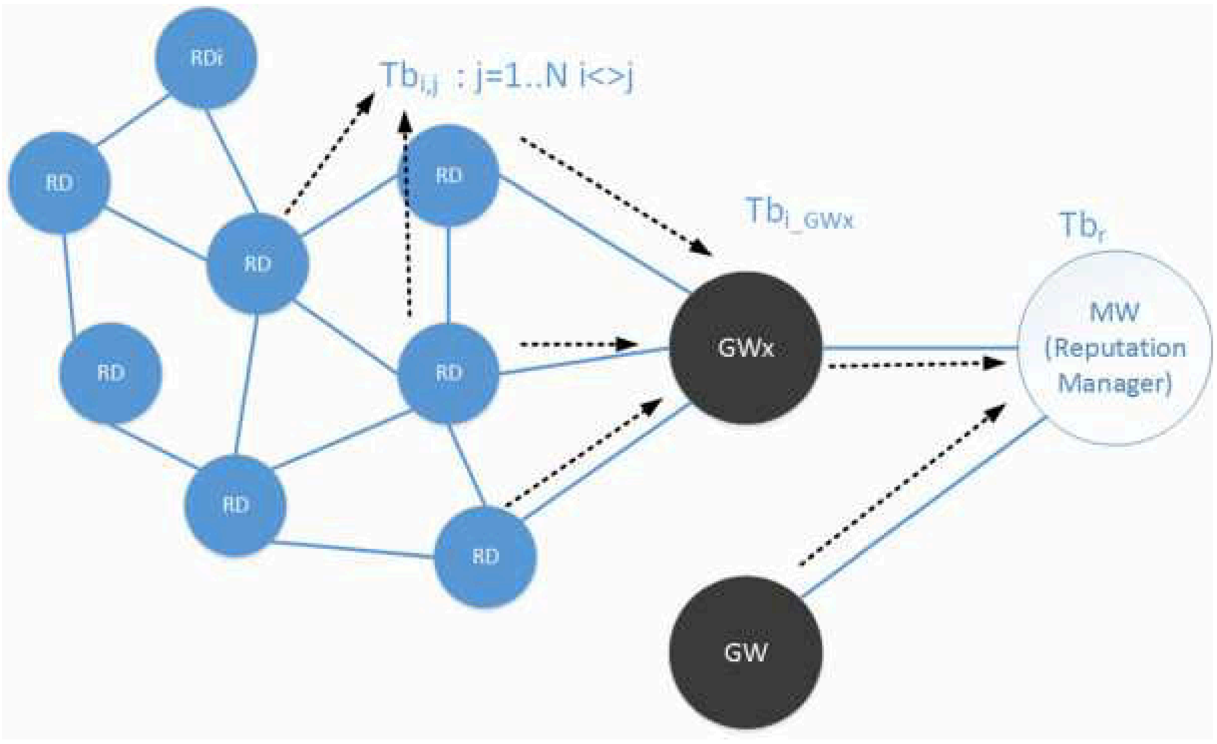


Figure 37 Three tier model architecture

The evaluation of the reputation of the devices can be done at various levels (Table 2), i.e. at node level, at Cluster Head/Gateway level or at the Middleware level where the Reputation Manager executes. In the following table, we define where each one of the above mentioned reputation values can be evaluated.

Level / trust value	behavioral	communication	Service-based
Node	X	X	-
Cluster head/gateway	X?	X?	X
middleware	-	-	X

Table 2: Trustworthiness evaluation at different levels

As it can be seen from the table above, at the node level only the behavioral and the communication reputation of its neighbor (one-hop) nodes can be evaluated and reported at the GW/middleware. This happens because these statistics can only be measured in the one-hop neighbors and only there. A rough estimation of the service-based reputation can also be done on the device, but since it is computationally expensive it should be avoided. Another reason is that only local knowledge about the measurements of the nodes can exist at each device and the results for the service-based reputation can be limited. At the Cluster Head/Gateway level the behavioral and communication metrics can be evaluated only for the nodes that are directly connected to the gateway. However, an indirect evaluation of those metrics for all the devices that are connected to the Middleware through this GW can be done, by the fusion of the measurements that are reported by the rest of the nodes. At the Cluster Head/GW the Service-based Reputation Evaluation can be done easily by both keeping a statistical analysis of the history of the measurements per service (as defined in Section 2.4 above), and per node, and by comparing the values with the respective measurements of other devices that are providing the same service and are monitoring the same VE. Of course, this service based reputation metric may be a rough estimation, since other devices providing the same service and monitoring the same VE may be connected to other gateways, thus there is a need for a higher level evaluation of the service based reputation. This higher level of evaluation is done at the MW (Reputation Manager), where all the measurements from all the devices are gathered and thus the global metric for the service-based reputation of each device can be calculated.

For the overall evaluation of the trust metrics, a hybrid scheme can be used, which exploits the benefits of the distributed and the hierarchical trust management schemes (as in [BF12]):

(i) **Intra-cluster level**, where distributed trust management can be used and each device evaluating the reputations for communication and behavior of its neighbors.

(ii) **Inter-cluster level**, where centralized trust management is used. Here, each device reports the trust values to the cluster head or the gateway, which calculates and disseminates the final values for the reputation of each node. In a scenario where GWs can cooperate with each other, we can also assume that the GWs can exchange trust rating values for the service-based Reputation Evaluation of the nodes.

(iii) **MW level**, where centralized trust management is used (Reputation Manager). Each GW reports the trust ratings for each of the devices to the MW, which takes over in order to evaluate the overall service-based reputation of the devices. The behavioral and communication reputation metrics are only stored into the MW because the final calculation of the respective metrics is done at the GW level.

3.10.4 Behavioural Statistics

The trust model that we use is built on the evaluation of trust degree values, based on the observed behavior of a node by its neighbor's. Actually, the model described here is inherited by the trust model described in Section 2, where all each node also acts as an observer of its neighbours. All nodes observe and record a neighbor's abnormal behavior each time there is an interaction between them, while keeping track of the total amount of incoming packets. For the sake of simplicity, and in order to keep the overhead low, two statistics that capture the forwarding behavior of a node are used: (i) *packet drop rate* (PDR) and (ii) *packet modification rate* (PMR), defined as follows:

$$PDR = \frac{\text{\# of dropped packets}}{\text{total \# of received packets}}$$

$$PMR = \frac{\text{\# of modified packets}}{\text{total \# of forwarded packets}}$$

Assume a receiver RD 'j'. Each time a packet is received by 'j', each neighbor 'i' of 'j' overhears forwarding behavior of 'j' and update accordingly the values of $PDR_{i,j}$ and $PMR_{i,j}$. Then, the aggregate *misbehavior rate* (MBR) of RD *j* as perceived by RD *i* is calculated as:

$$MBR_{i,j} = w \times PDR_{i,j} + (1-w) \times PMR_{i,j}$$

where $w \in [0,1]$ is a user-defined weight controlling the balance between the behavioral statistics.

3.10.5 Dishonesty of Malicious Nodes

Apart from their malicious acts in terms of the forwarding behavior, the trust model that we describe considers nodes that provide dishonest reports about the behavioral statistics of the neighbors they observe. Particularly, bad-mouthing (BM), and good-mouthing (GM) attacks are considered.

A bad-mouthing attack of a malicious node 'i' against the trustworthiness of a legitimate node 'j', is performed by deliberately increasing the observed misbehavior rate of that node 'j'. Here, the BM misbehavior rate $MBR^{BM}_{i,j}$ is calculated as:

$$MBR^{BM}_{i,j} = \min\left((1+w_{BM}) \times MBR^{BM}_{i,j}, 1\right)$$

where $w_{BM} \in [0,1]$ is a weight controlling the severance of bad-mouthing.

Respectively, a good-mouthing attack of a malicious node 'i' in favor of the trustworthiness of a malicious node 'j' is performed by deliberately decreasing the observed misbehavior rate of node 'j'. Here, we define the GM misbehavior rate $MBR^{GM}_{i,j}$ as:

$$MBR^{GM}_{i,j} = w_{GM} \times MBR^{GM}_{i,j}$$

where $w_{GM} \in [0,1]$ is a weight controlling the severance of good-mouthing.

3.10.6 Outlier Detection

Outlier detection is generally an important step in trust evaluation routine. An outlier in a dataset is an observation that is considerably dissimilar and inconsistent from the rest of the data. This is regarded as either a "nuisance" that has to be quickly identified and eliminated for the sake of increased trust evaluation robustness, or as an effective way to detect abnormal phenomena in the area of outlier appearance.

In this deliverable, we propose the use of outlier detection in order to detect dishonesty of malicious nodes, i.e. report of falsified observed behavioral statistics to the Reputation Manager. In this way, we claim that it is possible to improve the reliability in combination of MBRs of each node, as they are observed by their neighbors.

In particular, we adopt a well-known distance-based definition of an outlier, namely the distance from the k-th nearest neighbor (KNN) [RARA00]. The outlier detection scheme is applied in the Reputation Manager on the MBR values reported for each node. Consequently, outlier values are omitted in belief computation and combination according to the Dempster-Shafer theory, as described in the next section.

3.10.7 Performance Evaluation

We model the overlay network as a bi-directed graph G(*V*,*E*) where *V* is the set of nodes, including a single sink node that plays the role of the Reputation Manager, and *E* is the set of edges representing wireless links between nodes. The sink node is assumed to be more powerful in terms of processing, memory, and energy. We assume that each node has a limited communication range, thus can directly communicate only with the nodes that reside inside this range, namely its *neighbors*. This is of course something quite reasonable in IoT deployments due to the fact that devices want to utilize low transmission power in order to consume less energy and prolong their lifetime. Communication with the sink/gateway node is possible for each node in a multi-hop fashion with each one following a path calculated through a path calculation algorithm at the sink.

3.10.7.1 Detection Probability of the Malicious Nodes

In order to evaluate the proposed trust management scheme we assume a sensor network deployment that consists of 100 nodes deployed uniformly at random in an area of 100mx100m. We choose the Reputation Manager to be at the center of the area, and we conduct simulations for data aggregation rounds. Routing is performed by using the trust-based routing scheme proposed in [TRACA 15], where malicious nodes are ignored by the routing scheme. Trust belief update interval is set to 20 rounds, while we choose $c = 0.1$, $h = 0.5$, $r = 0.05$ and $w = 0.5$. We select the distance from the 3-rd nearest neighbor as the outlier definition distance metric.

We assume two scenarios, namely *highly malicious behavior*, and *low malicious behavior*. In the first case, the malicious nodes exhibit PDR and PMR with values in [0.8,1], while in the second one, the values are in [0.3,0.5]. Likewise, we assume two scenarios, with regards to node dishonesty, namely *severe dishonesty*, where w_{BM} varies in [0.8,1], and w_{GM} varies in [0,0.2], and *light dishonesty*, where w_{BM} varies in [0.3,0.5] and w_{GM} varies in [0.5,0.7]. Finally, the number of the malicious nodes varies in [5,40]. For each scenario combination, we execute 20 Monte Carlo runs, each consisting of 2000 data aggregation rounds.

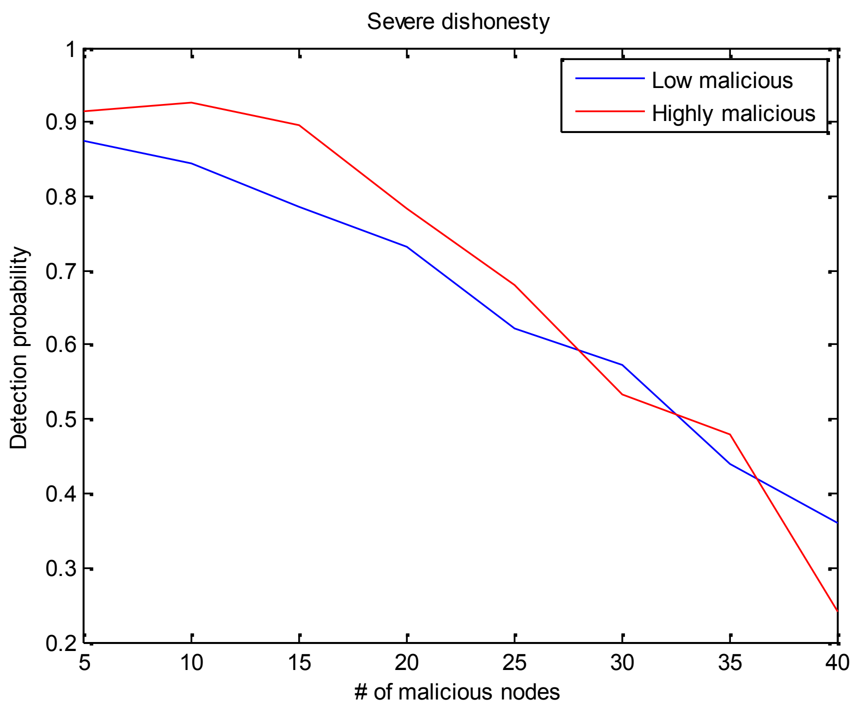


Figure 38 Detection probability of the malicious nodes (Severe dishonesty)

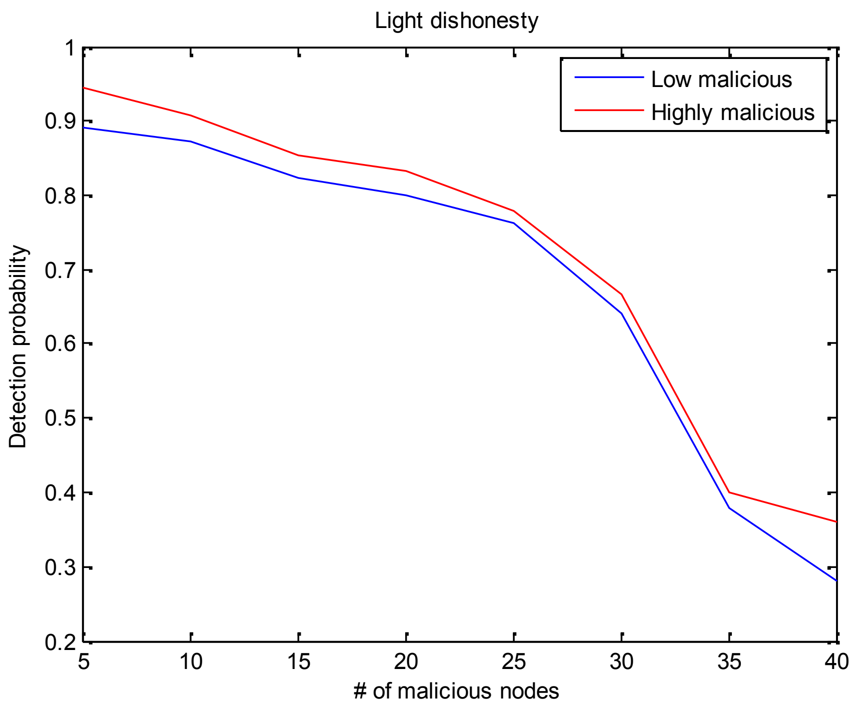


Figure 39 Detection probability of the malicious nodes (Light dishonesty)

Figure 38 displays the detection probability of the malicious nodes vs their total number in case of the severe dishonesty. As expected, detection probability decreases as the number of the malicious nodes increases, since the abnormal behavior of the malicious nodes prevails over that of the legitimate ones. Additionally, highly malicious behavior is more easily detected than low malicious behavior, especially for a small number of malicious nodes. This is because the algorithm can more easily identify the malicious behaviors.

Accordingly, Figure 39 displays the detection probability vs the number of the malicious nodes for the case of the light dishonesty. Same conclusions as in former case can be drawn. By comparing Figure 38 and 9 we should note that in the case of the light dishonesty, the detection probability remains high for a larger number of malicious nodes, compared to the severe dishonesty case. For example, for 25 malicious nodes and severe dishonesty, the detection probability is around 65%, while in case of the light dishonesty, the detection probability is almost 80%. It is possible that this happens because for the light dishonesty, it is easier to detect malicious nodes, since bad-mouthing and good-mouthing attacks cannot severely deceive the outlier detection algorithm.

After a node is characterized a malicious, the DEC_INT_DEV_REPTN (Decrease Internal Device Reputation) alert can be raised so the specific node is isolated from the network.

3.10.7.2 Dempster-Shafer Conflict Computation

Here, we investigate the effectiveness of the behavioral trust fusion mechanism based on Dempster-Shafer (D-S) theory. In particular, we present the values of probability mass related to conflict in fusion of neighbors' individual trust beliefs that expresses the level of discrepancy in the observed misbehaviors.

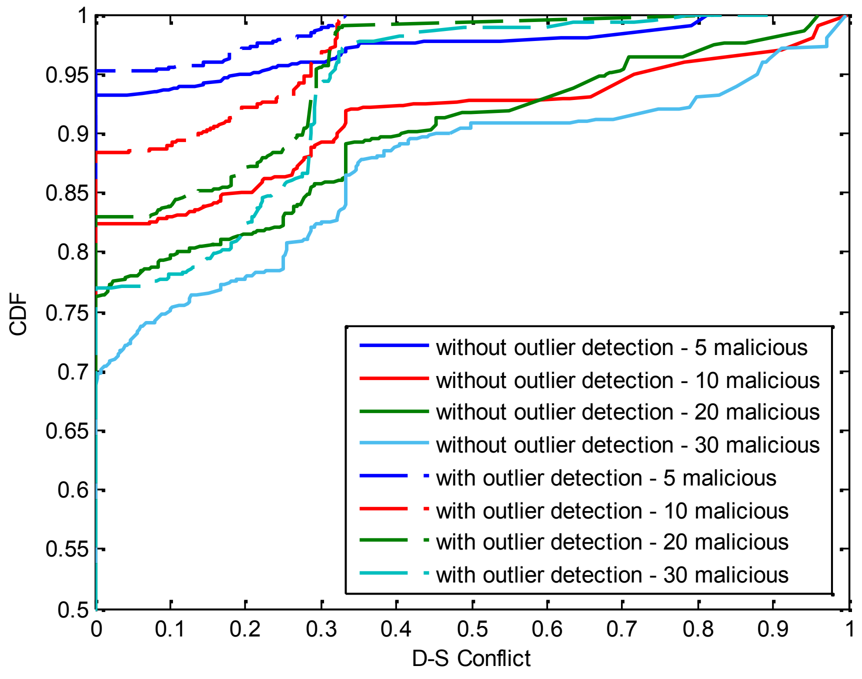


Figure 40 CDF of D-S conflict for severe dishonesty

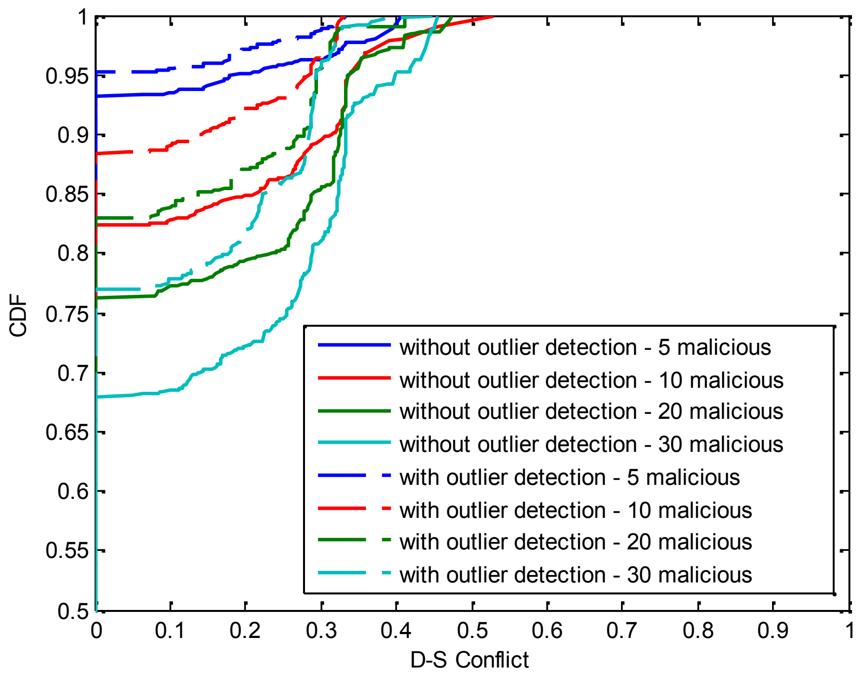


Figure 41 CDF of D-S conflict for light dishonesty

Figure 40 and Figure 41, illustrate the cumulative density function (CDF) of D-S conflict for the case of severe dishonesty and light dishonesty, respectively. It is obvious that as the number of malicious nodes increases, D-S conflict increases for both cases. This is because more individual beliefs (corresponding to malicious nodes) are corrupted, increasing the level of discrepancy during fusion. Furthermore, outlier detection algorithm has a positive effect on reducing conflict, especially for increased number of malicious nodes. For example, observe that in case of severe dishonesty and 5 malicious nodes there is only almost 1.5% increase in belief fusions that admit conflict lower than 0.1, while in case of 20 malicious nodes there is an increase of more than 6% for the same conflict level.

4 Conclusions and Future Work

In this document we present a generic trust model for the IoT, which has two different parts: a Trust Evaluation Model for evaluating the trust and reputation of the services and devices, and a Reacting Model for stating the reaction to the evaluations generated in the Trust Evaluation Model.

Due to the dynamic nature of Internet, and consequently of the IoT, it is not possible to provide a model that fits every possible need of the IoT. For this reason, what we have instead provided is a set of essential criteria together including tools to configure custom rules and reactions. As a consequence, future research is still needed to create new rules, for observers, reputation engines and trust query evaluations adapted to different applications in the IoT, which will be an ongoing process.

Regarding the Trust Evaluation Model, more research is needed to create or adapt generic process models such that they can be used in streaming mode by constrained devices (close to the data generators) and that can be easily calibrated for different environments.

In this document we did not deal with social trust management techniques only because of privacy problems, but also because it is difficult to trust anonymous users. Perhaps methods related to bitcoins (distributed trust, anonymous rewards for correct information) could be used to enable privacy-preserving social trust methods in IoT. This can be another field of research in the future, too.

The document also introduces the concept of reacting to the Reputation Evaluations produced in the Trust Evaluation Model, and presented the most frequent reactions. Among these frequent reactions, it also included the ability to feed the authorization components with those evaluations to improve the authorization process.

Regarding the Integration of RERUM with the Trust Model, this document also included a first design with this goal in mind. Some of the required components were not only designed but also implemented because of their importance as POCs or because they were committed in previous deliverables. These POCs are:

- Basic engine for executing the Trust Evaluation Model,
- Basic Reaction to the results of the model, and
- Incorporation of the Reputation Evaluation of Users to the Authorization Process.

The provided design of the documents cover the complete list of components to be adapted, that is:

- Interception of RERUM Data;
- Collection of Data from External Services;
- Managing Context Data;
- Processing of Reputation;
- Correlating Reactions to Reputation Alerts;
- Processing Reactions;
- Incorporating the Reputation Access Profiles to the Authorization Process;
- Processing of reputation of users; and
- Evaluation Reputation of Devices at network level.

Regarding the compliance with the Trust Model Requirements, it is depicted in Table 3: Summary of Compliance with Trust Model Requirements below in this section.

For the trust Reputation Evaluation at the network level, the work presented a framework for assessing the reputation of IoT devices at various levels (according to the design choice) and using various network statistics and data reliability values. An example instantiation of this framework was tested via simulations to show its performance. There, we used the *packet drop rate* and the *packet* *modification rate*, for computing the misbehaviour rate of each node and we collected the statistics at the Reputation Manager, which fused them using the D-S algorithm, to estimate the probability that supports the malicious nature of a node. For all scenarios we showed that the proposed framework can identify quickly potential "outliers" or malicious devices and prevent them from affecting the reliability of the overall system.

As a final summary, the table below lists the set of requirements, the section or sections that covered them, and whether they have been included in any of the POC prototypes of the task.

Table 3: Summary of Compliance with Trust Model Requirements

Requirement	Section	Comments	Included in POC
TM1: Efficiency	2.4 Trust Evaluation Model and 2.5 Reacting Model (ATOS)	This requirement involves the whole processing of the data and hence it is not covered in a single section. it was necessary to balance performance and functionality in some sections, such as using an UDP stream instead of a full queuing system or remote procedure calls for the Trust evaluation Evaluator	No
AM2: Support Evaluation of Reputation at Service Level	2.4.3 Trust for services jointly with 2.4.5, 2.4.6, 2.4.7 and 2.4.8	The Reputation Manager collects all trust ratings and produces a single composite value with the value for the different evaluation criteria. Sections 2.4.5 to 2.4.8 defines the mechanisms to locate when a concrete value does not seem to match an expected distribution	Yes
AM3: Support Querying Trust at Service Level	2.4.3 Trust for services	The Observers produce a trust rating for the services whose values they are inspecting	Yes
AM4: Support Querying Trust at Device Level	2.4.4 Trust of single data value	Integrity checks assess the reputation at device level	Yes
AM5: Support Evaluation of Trust at Device Level	2.4.4 Trust of single data value	Integrity checks assess the reputation at device level	Yes
AM6: Support Reputation for multiple evaluation criteria	2.4.3 Trust for services	The Reputation Manager allows for the evaluation of multiple evaluation criteria according to the rules that	Yes

		configure it	
AM7: Support for multiple Observers	2.4.3 Trust for services	The Reputation Manager collects all trust ratings and produce a single composite value with the value for the different evaluation criteria	Yes
AM8: Produce Alert only for Relevant Changes	2.4.9 CLIPS rules	CLIPS rules can be set up to decide when the changes are relevant	Yes
AM9: Ability of redemption of devices and services	2.4.4 Trust of single data value	Provided an option to recover from zero trust.	No
RM10: Ability to react to Reputation Evaluations	2.5.2 Generic Reacting System	Provided a set of possible reactions and a model for executing them	No
RM11: Generic Reaction system	2.5.2.4 Generic Execution Mechanism	Provided a set of the main generic execution mechanisms suitable of IoT	Yes
RM12: Support Integration with an Authorization Mechanism	2.5.4 Heterogeneous Access Control and Profiles	Provided several options for incorporating Reputation Evaluation to the authorization process	Yes, but only for reputation of users

References

[ARM-IoT 15] TrustZone, SecurCore. Available at: https://www.arm.com/products/security-on-arm/index.php. last accessed: July 2015.

[Bailey 03] James Bailey, George Papamarkos, Alexandra Poulovassilisy, and Peter T Woody. An event-condition action language for XML. Department of Computer Science, University of Melbourne, School of Computer Science and Information System, Birbeck College, University of London. March 2003.

[Bao 12] Fenye Bao and Ing-Ray Chen. Trust management for the Internet of Things and its application to service composition. Workshop Paper 978-1-4673-1238-7 IEEE, Department of Computer Science, Virginia Tech. June 2012.

[Bernabe 15] Jorge Bernal Bernabe, Jose Luis Hernandez, Ramos Antonio F., and Skarmeta Gomez. TACIoT: Multidimensional trust-aware access control system for the Internet of Things. Focus Soft Computing, pp.1-17, Springer. 2015.

[Bissmeyer 12] Norbert Bißmeyer, Sebastian Mauthofery, Kpatcha M. Bayarou, and Frank Kargl. Assessment of node trustworthiness in VANETs using data plausibility checks with particle filters. Fraunhofer SIT, Uni Darmstadt, Uni Ulm. 2012.

[Boglaev 04] Yuri Boglaev. Interchange of ECA rules with Xpath expressions. Ameritrade. Available via www.w3.org/2004/12/rules-ws/paper/7/. 2004.

[Chang 12] Kai-Di Chang and Jiann-Liang Chen. A survey of trust management in wireless sensor networks, Internet of Things and future internet. KSII Transactions on Internet and Information Systems, (1976-7277). January 2012.

[Chen 11] Dong Chen, Guiran Chang, Dawei Sun, Jiajia L1, Jie Jia1, and Xingwei Wang: TRM-IoT: A trust management model based on Fuzzy Reputation for Internet of Things. Computer Science and Information Systems, Issue: 20, pp.1207-1228. 2011.

[Chen 14] Ing-Ray Chen, Jia Guo, and Fenye Bao: Trust management for service composition in SOA-based IoT systems, Virginia Tech, 978-1-4799-3083-8/14, IEEE, pp.3486-3491. 2014.

[Cho 11] Jin-Hee Cho, Swami, A., and Ing-Ray Chen: A survey on trust management for mobile ad hoc networks. in Communications Surveys & Tutorials, IEEE (Volume:13 , Issue: 4), pp.562 – 583. 2011.

[CLIPS-RefMan-6.30-2015] CLIPS Reference Manual. CLIPS Basic Programming Guide version 6.30. (http://clipsrules.sourceforge.net/documentation/v630/bpg.pdf). last accessed: March 17th 2015.

[Daubert 15] Joerg Daubert, Alexander Wiesmaier, and Panayotis Kikiras. A view on privacy & trust in IoT. In IOT/CPS-Security Workshop, IEEE International Conference on Communications, ICC 2015, London, GB, June 08-12, IEEE. 2015.

[Douceur 02] J.R. Douceur and J.S. Donath. The Sybil attack. Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), pp.251–260. 2002.

[ENISA 13] ENISA European Union Agency for Network and Information Security: On the security, privacy and usability of online seals. An overview Version. December 2013.

[Fritsch 11] L. Fritsch, A.-K. Groven, and T. Schulz. On the Internet of Things, trust is relative. Norwegian Computing Center, in R. Wichert, K. Van Laerhoven, J. Gelissen (EDS.): AmI 2011 Workshops, CCIS 277, pp.267-273, Springer-Verlag. 2012.

[Gamma 95] E, Gamma, R. Helm, R. Johnson, and J. Vlissides: Design patterns - elements of reusable object-oriented software. Addison-Wesley. 1995.

[Gomez 08] F. Gomez Marmol and G. Martınez Perez. Providing trust in wireless sensor networks using a bio-inspired technique. In: Proceedings of the networking and electronic commerce research conference, NAEC'08. Lake Garda, Italy. September 2008.

[GUGA 14] Manish Gupta, Jing Gao, Charu Aggarwal, and Jiawei Han. Outlier detection for temporal data. Synthesis Lectures on Data Mining and Knowledge Discovery, Vol. 5, No. 1 , pp.1-129. March 2014.

[Gustafsson 10] Fredrik Gustafsson: Particle filter theory and practice with positioning applications. Senior Member IEEE tutorial. 2010.

[Intel-IoT 15] Enhanced Privacy Identity (EPID). Available at: http://www.trustedcomputinggroup.org/files/static_page_files/93061BAE-1A4B-B294-D0F3EBD27DB68FAB/IOT_Security_Architects_Guide_TCG.pdf. July 2015.

[IOT-A D4.2] Concepts and solutions for privacy and security in the resolution infrastructure., D4.2, Section 4.5 Trust and Reputation Architecture (TRA), http://www.iot-a.eu/public/public-documents/d4.2/at_download/file.

[ITU-T Y.2060] Overview of the Internet of things. Available at: http://www.itu.int/ITU-T/recommendations/rec.aspx?rec=y.2060. International Telecommunication Union. Study Group 13. June 2015.

[Javanmardi 14] Saeed Javanmardi, Mohammad Shojafar, Shahdad Shariatmadari, and Sima S. Ahrabi. FR TRUST: A fuzzy reputation based model for trust management in semantic P2P grids. 2014.

[Javed 12] Nauman Javed and Tilman Wolf. Automated sensor verification using outlier detection in the internet of things. In ICDCS Workshops, pp.291–296. IEEE Computer Society, http://www.ecs.umass.edu/ece/wolf/pubs/cpns2012.pdf. 2012.

[Joesang 07] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. Decision Support Systems, 43(2):618–644. 2007.

[Jelia 06] J. J. Alferez, F. Banti, and A.Brogi, An event-condition-action logic programming language. CENTRIA, Universidade Nova de Lisboa, Portugal, Dipartamento di Informatica, Universitâ de Pisa, Italy. Available via www.di.unipi.it/~brogi/papers/JELIA06.pdf. 2006.

[Kantere 06] Verena Kantere, Iluju Kiringa, and John Mylopoulos. Supporting distributed event-condition-action rules in a multidatabase environment. International Journal of Cooperative Information Systems, August 2006.

[Kamvar 03] S. Kamvar, M. Schlosser, and H. Garcia-Molina. The EigenTrust algorithm for reputation management in P2P networks. Budapest, Hungary. May 2003.

[KeWi06] Ketil Stølen, William H. Winsborough, Fabio Martinelli, Fabio Massacci. 'Trust Management', from 4th International Conference, iTrust 2006, Pisa, Italy, May 16-19, 2006.

[Leister 12] Wolfgang Leister and Trenton Schulz._Ideas for a trust indicator in the Internet of Things. Norsk Regnesentral; SMART 2012: The First International Conference on Smart Systems, Devices and Technologies. IARIA. 2012.

[Llanes 11] Marc Llanes et Al. D2.1.1 Scenario Requirements for MASSIF project. Available via http://www.massif-project.eu/list_deliverables. May 2011.

[Maatjes 07] M.C. Maatjies. Automated transformations from ECA rules to Jess. University of Twente. available via http://essay.utwente.nl/559/. 2007.

[Marmol 09] Felix Gomez Marmol, Gregorio Martınez Perez, Security threats scenarios in trust and reputation models for distributed systems. University of Murcia, Elsevier, computers & security 28. pp.545 – 556. 2009.

[Marti 06] S. Marti, H. Garcia-Molina, Taxonomy of trust: categorizing P2P reputation systems. Computer Networks 50 (4) pp.472–484. Elsevier Science. Available via http://zoo.cs.yale.edu/classes/cs457/fall13/TaxonomyOfTrust.pdf. 2006.

[Maybeck, Peter S.: Stochastic models, estimation, and control. Mathematics in Science and Engineering, vol 141, , Academic Press, Inc. 1979.

[Momani 09] Mohammad Momani and Subhash Challa. Survey of trust models in different network domains. UTS, Sydney, Australia, University of Melbourne, Australia. 2009.

[Nitti 12] Michele Niţti, Roberto Girau, Luigi Atzori, Antonio Iera, and Giacomo Morabito. A subjective model for trustworthiness evaluation in the social Internet of Things. In 23rd Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), doi: 978-1-4673-2569-1/12, pp.18–23. 2012.

[Oracle 1] Oracle Corporation. Introduction to Rules Manager. Available at http://docs.oracle.com/cd/E11882_01/appdev.112/e14919/exprn_intro.htm#EXPRN001.

[Oracle 2] Oracle Corporation. The Java EE 6 Tutorial: Filtering Requests and Responses. Available at https://docs.oracle.com/javaee/6/tutorial/doc/bnagb.html. 2013.

[OHara 14] K. O'Hara, H. Alani, Y. Kalfoglou, and N. Shadbolt. Trust strategies for the Semantic Web. In: ISWC'04 Workshop on Trust, Security and Reputation on the Semantic Web. 2004.

[Probst 06] Matthew J. Probst and Sneha Kumar Kasera. Statistical trust establishment in wireless sensor networks. Utah University, 2006.[RARA00] Ramaswamy S., Rastogi R., and Shim K. Efficient algorithms for mining outliers from large datasets. ACM SIGMOD Conf.. 2000.

[Rani 14] V. Uma Rani andK. Soma Sundaram. Review of trust models in wireless sensor networks. World Academy of Science, Engineering and Technology; International Journal of Computer, Electrical, Automation, Control and Information Engineering Vol.8, No.2. 2014.

[Raya 08] Maxim Raya, Panagiotis Papadimitratos, Virgil D. Gligory, and Jean-Pierre Hubaux. On data-centric trust establishment in ephemeral ad hoc networks. School of Computer and Communication Sciences; EPFL, Switzerland; Department of Electrical and Computer Engineering; Carnegie Mellon University, USA, EU project SEVECOM. 2008.

[RD2.4] G.Moldovan. 'Smart Object Middleware'. RERUM deliverable D2.4. April 2014

[RD3.1] D. Ruiz Lopez (Ed.) et al., 'Enhancing the autonomous smart objects and the overall system security of IoT based Smart Cities', RERUM Deliverable D3.1. March 2015.

[RD3.2] R. Staudemayer (Ed.) et al., 'Privacy enhancing techniques in the Smart City applications', RERUM Deliverable D3.2. September 2015

[RD5.2] A. Liñán, M. Fabregas (Ed.) et al., 'Smart object and application Implementation', RERUM Deliverable D5.2. September 2015

[Richardson 03] M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic web. University of Washington, IBM Research, Proceedings of the Second International Semantic Web Conference. pp.351-368, Sanibel Island, FL: Springer. https://homes.cs.washington.edu/~pedrod/papers/iswc03.pdf. 2008.

[Saranya 15] V.S. Saranya and R. Saminathan: Reputation and Plausibility Verification based System for Providing Secure Vehicular Networks International Journal of Computer Applications (0975 – 8887) Vol.128 – No.3, Tamil Nadu, India. October 2015.

[Schukat 15] M. Schukatand P. Cortijo. Public key infrastructures and digital certificates for the Internet of things. 26th Irish Conference in Signals and Systems (ISSC)pp.1-5, 24-25 June 2015.

[Sicari 13] Sabrina Sicari, Alberto Coen-Porisini, and Roberto Riggio. Dare: evaluating data accuracy using node reputation. In Computer Networks, volume Issue 15, p.3098. Elsevier Science. doi: 10.1016/j.comnet.2013.07.014. October 2013.

[Soleymani 15] Soleymani et al.. Trust management in vehicular ad hoc network: a systematic review. EURASIP Journal on Wireless Communications and Networking. DOI 10.1186/s13638-015-0353-y. 2015.

[STORM 15] Apache Storm project available via http://storm.apache.org/index.html

[TCG-IoT 15] Trusted Computing Group: Architect's Guide: IoT Security. Available at: http://www.trustedcomputinggroup.org/files/static_page_files/93061BAE-1A4B-B294-D0F3EBD27DB68FAB/IOT_Security_Architects_Guide_TCG.pdf. July 2015.

[TRACA 15] E. Tragos, P. Charalampidis, A. Fragkiadakis, G. Lambropoulos, and S. Kyriazakos. Improving the trustworthiness of ambient assisted living applications, in Proc. of WPMC, Hyderabad, India. December 2015.

[WISER 15] Project Wiser available via https://www.cyberwiser.eu/. June 2015.

[WWIS_Mun-2011] World Weather Information Service – Munich. June 2011.

[Xiong 04] Xiong L. and Liu L. PeerTrust: supporting reputation-based trust in peer-to-peer communities. IEEE Transactions on Knowledge and Data Engineering. 16(7):843–57. 2004.

[Yan 08] Z. Yan and S. Holtmanns. Trust modelling and management: From social trust to digital trust. In R. Subramanian, editor, Computer security, privacy and politics: current issues, challenges and solutions IGI Global, p.290. 2008.

[Yan 11] Z. Yan and C. Prehofer. Autonomic trust management for a component based software system. IEEE Transactions on Dependable Secure Computing, 8:810. 2011.

[Yosra 13] Yosra Ben Saied, Alexis Olivereau, Djamal Zeghlache, Maryline Laurent. Trust management system design for the Internet of Things: A context-aware and multiservice approach. Elsevier; computers & security, pp.1-15. 2013.

[Zan 14] Z Yan, P Zhang, AV Vasilakos: A survey on trust management for Internet of Things. Journal of network and computer, Elsevier, Vol.42, pp.120–134. June 2014.

[ZAHA 10] Zahariadis, Theodore, et al. Trust management in wireless sensor networks. European Transactions on Telecommunications 21.4. pp.386-395. 2010.

[Zheng 14] Zheng Yan, Peng Zhang, and Athanasios Vasilakos. A survey on trust management for Internet of Things. Journal of Network and Computer Applications, (42):120. March 2014.