A Framework for Knowledge Derivation Incorporating Trust and Quality of Data

By

©2013
Martin Kuehnhausen

Submitted to the graduate degree program in
Electrical Engineering and Computer Science
and the Graduate Faculty of the University of Kansas
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy.

Dissertation Committee:

Dr. Victor S. Frost, Chairperson

Dr. Gary J. Minden

Dr. Jun Huan

Dr. Bo Luo

Dr. Tyrone S. Duncan

Date Defended

The Dissertation Committee for Martin Kuehnhausen certifies
that this is the approved version of the following dissertation:

A Framework for Knowledge Derivation
Incorporating Trust and Quality of Data

Chairperson

Date Approved

Abstract

Today, across all major industries gaining insight from data is seen as an essential part of business. However, while data gathering is becoming inexpensive and relatively easy, analysis and ultimately deriving knowledge from it is increasingly difficult. In many cases, there is the problem of too much data such that important insights are hard to find. The problem is often not lack of data but whether knowledge derived from it is trustworthy. This means distinguishing "good" from "bad" insights based on factors such as context and reputation. Still, modeling trust and quality of data is complex because of the various conditions and relationships in heterogeneous environments.

The new *TrustKnowOne* framework and architecture developed in this dissertation addresses these issues by describing an approach to fully incorporate trust and quality of data with all its aspects into the knowledge derivation process. This is based on *Berlin*, an abstract graph model we developed that can be used to model various approaches to trustworthiness and relationship assessment as well as decision making processes. In particular, processing, assessment, and evaluation approaches are implemented as *graph expressions* that are evaluated on *graph components* modeling the data.

We have implemented and applied our framework to three complex scenarios using real data from public data repositories. As part of their evaluation we highlighted how our approach exhibits both the formalization and flexibility necessary to model each of the realistic scenarios. The implementation and evaluation of these scenarios confirms the advantages of the *TrustKnowOne* framework over current approaches.

Acknowledgments

I would like to thank the people who have made my dissertation possible, my advisor Dr. Frost and the committee members Dr. Minden, Dr. Huan, Dr. Luo, and Dr. Duncan.

I am forever grateful to the tremendous support, advice, and guidance I have received from Victor.

I wish to thank the University of Kansas for providing me with the opportunities and support to succeed throughout this academic journey.

I am very fortunate for all the love and support from my family and friends. Thank you so very much, Mom and Fabian. I also hope that I made my Dad proud and wish he could have been here.

Most importantly I wish to thank my wonderful Erika for making everything possible and my daughter Natalie for bringing me all the joy and happiness one could ever wish for.

Table of Contents

List of Code Listings

List of Definitions

List of Expressions

List of Figures

List of Tables

1 Introduction

In this dissertation "A Framework for Knowledge Derivation Incorporating Trust and Quality of Data" is introduced. The premise for this research is that existing approaches to knowledge derivation can be significantly improved by incorporating the assessments of data quality and trustworthiness.

Chapter 2 discusses the problems associated with current knowledge derivation processes. In particular, it highlights our approach and contributions. Context for our framework is provided in chapter 3 where we discuss related work and research areas.

The basis of our framework consists of an *abstract graph model* on which *graph expressions* are evaluated. This *model* called *Berlin* is presented in chapter 4. Chapter 5 discusses in detail how our *TrustKnowOne* framework is able to incorporate trust and quality of data aspects into knowledge derivation processes.

The application of our framework to specific scenarios is examined in chapter 6. The focus lies on showcasing how *TrustKnowOne* provides a formal and flexible approach to knowledge derivation in a variety of scenarios. The implementation of these realistic scenarios is used to confirm our claims concerning the advantages of the *TrustKnowOne* framework over the current state of the art.

Our reference implementation of the framework is discussed in chapter 7. In chapter 8 our *TrustKnowOne* framework is compared and evaluated against representative frameworks and approaches from literature. The dissertation concludes with chapter 9 that also provides an outlook for future work.

2

Problem Statement and Contributions

2.1 Motivation

All data is essentially used to make decisions. In general, these decisions are based on the assumption that the data itself is valid and useful. However, how do we determine the quality of this data, is it affected or influenced by other data, and does it change over time? Furthermore, determining the usefulness of the data is also based on the level of trust we put into the data source, especially when the data is confidential or there is a potential conflict-of-interest when reporting the data. Since our decisions are based on this data we need to understand what is correct and can be trusted, otherwise we may make wrong decisions.

This research addresses the problem of systematically and formally incorporating trust and data quality as well as time and other system dynamics into the knowledge derivation process.

2.2 Approach

We create a new framework called *TrustKnowOne* where we associate every piece of data with some model (probabilistic or deterministic) representing data quality and trustworthiness. Furthermore, we provide formalized means for determining, describing and combining these models and their parameters as well as functionality to challenge them. We utilize relationships between pieces of data and data sources to assess trust and opinions of them. Similar approaches can be found in intrusion detection and computer and social networks. However, they are often not formalized and lack a comprehensive framework that is flexible enough to deal with a wide variety of realistic scenarios. Moreover, the lack of a formal framework inhibits comparing different proposed techniques.

The following provides an overview of our approach and its benefits.

2.2.1 A Knowledge Derivation Framework

Our *TrustKnowOne* framework is divided into three components to allow for a layered approach and increased flexibility. First, *knowledge extraction* allows us to formalize a general description of data elements and their context (meta information) as measurements. This *knowledge extraction* formalization is applicable to many realistic scenarios as we will demonstrate in chapter 6. Second, *knowledge processing* deals with taking these measurements and attaching additional *relationship meta information* in order to provide beliefs and opinions about the measurements. Third, the measurements as well as the beliefs and opinions are then used by the *knowledge evaluation* component to make decisions.

One of the advantages of separating *knowledge processing* and *evaluation* is that there may be various approaches to modeling data quality and trust/opinion relationships as well as various decision engines. Current approaches [31, 54, 63, 65] often combine the trust modeling aspect with the decision problem they are trying to solve. Doing so makes the comparison of individual approaches and further improvements to them difficult.

The separation of functionality into a layered framework as presented here is a necessary step towards gaining a better understanding of the advantages and disadvantages of current methods. In some cases in literature [31, 54, 63] simply choosing a different decision engine could yield completely different results but without a framework like the one developed here it is too difficult to assess the potential improvement.

2.2.2 Formalization of Knowledge Derivation

The framework creates a new formalization approach to combining raw data with meta information on a *local* level (e.g., time, space, how it was obtained, security features) as well as a *global* level (context, attestation, expected behavior, history and ownership data) using an abstract graph model that is suitable to assessing data quality and trustworthiness. The formalized and flexible nature of our approach allows for addressing a variety of data types that may be required to support a wide range of applications as will be confirmed by the application of our framework to several realistic scenarios (chapter 6. For instance, determining the trustworthiness of Smartphone Apps (section 6.1) requires extensive modeling of heterogeneous entities and relationships which we demonstrate our framework is capable of.

2.2.3 Adaptable Quality and Trust Assessments

Our research derives quality and trust assessments for each measurement based on a rich set of data and meta information from multiple resources and contexts. This includes a rigorous process of how to derive confidence in measurements from data by incorporating and evaluating local meta information, history, expected behavior, global data, and context information. We provide a modular and extensible approach to incorporate a variety of trust and quality assessment techniques using graph *expressions*. We demonstrate this using the scenarios discussed in chapter 6: trustworthiness assessments with heterogeneous entities (section 6.1), complex assessments on individual sensors as well as groups of sensors (section 6.2), and assessments in dynamic environments (section 6.3).

2.2.4 Dynamic Reassessment of Data

Our framework enables the request of additional data or challenge of existing data when certain confidence thresholds are not reached. This approach supports weighted decision processes where one part of the system to be analyzed is more critical than others as well as time critical ones where the best decision needs to be made given certain time and context constraints. Furthermore, in contrast to other frameworks the decision engine is able to utilize both data and additional information such as trust and data quality assessments when deciding on actions to take. We use an implementation of an intrusion detection scenario to demonstrate this aspect of our framework (section 6.3). Here, trustworthiness is based on the evaluation of test messages in a dynamic environment of hosts. Depending on the confidence of a particular assessments we can adjust the difficulty as well as the rate of messages that are being sent.

2.2.5 Flexible Decision Processes

The framework incorporates a flexible decision engine which allows for estimating the trustworthiness of data based on the assessment and confidence of individual measurements, their meta information, and context. This involves deriving a decision confidence from the confidence of the measurements and particular data sources. Our framework provides these trustworthiness assessments so that they can be directly incorporated into knowledge derivation and decisions (e.g., performing analysis only on data above certain trustworthiness levels, discarding low quality data points, etc.). In order to allow for flexible comparison and evaluation, we formalize decision processes as graph *expressions* that can be reused, modified, and extended as shown in the scenarios (chapter 6). This ranges from weighting schemes including ratings, reviews, and permissions (section 6.1) to threshold-based trust classes and incorporating ownership lineage (section 6.2 to evaluating model vulnerabilities (section 6.3).

2.2.6 Analysis of Data Attacks

Our developed framework is able to handle missing data, data in error, and purposefully modified data (an information attack). The key is that we take into consideration that there are always inherent *operational system impairments* present in data that may not reflect an attack. However, when *changes in data* become correlated we are able to detect these patterns and determine the presence of attacks. Because of the formal nature of our approach using graph *expressions* we are able to assess the robustness of the individual techniques and algorithms (i.e., belief engines and decision processes) against specific attack scenarios. An implementation of an intrusion detection scenario (section 6.3) is used to demonstrate this aspect.

2.2.7 Application to Diverse Scenarios

As part of the discussion we will highlight how our approach exhibits both the formalization and flexibility necessary to model each of the realistic scenarios. These scenarios discussed in chapter 6 are used to confirm the advantages of the *TrustKnowOne* framework over current approaches. We focus our analysis on the following representative and realistic scenarios. The selected scenarios and their implementations are realistic in terms of being geographically distributed, exhibiting time dynamics, and consisting of large and diverse data sets.

First, we discuss how we can evaluate the trustworthiness of Smartphone Apps by incorporating a variety of relationship and context assessments (section 6.1). We show that this approach yields a significant improvement over current methods that are based on basic App attributes [95]. Our data set for this scenario contains a total 11326 Apps, 790940 reviews (651801 with text, 139139 without) as well as 134 different kinds of permissions captured in July 2012. For this purpose we developed a web crawler to pull the real and rich App attributes out of Google Play (Android Market). As such, our data is a diverse representation of realistic data with complex attributes and relationships.

Second, we apply our framework to distributed collaborative sensing in the domain of radiation detection (section 6.2). Here, we deal with changes in sensor values over time as well as complex relationships between them. In particular, we combine data from three data sources amounting to ≈ 2.5 million time stamped data points over the course of nine months which are geographically distributed across Japan. Two of the data sets were provided by the International Atomic Energy Agency [75] whereas the third data set from Safecast [144] represents measurements taken from thousands of people in a collaborative sensing effort. As such, the Safecast [144] data represents a challenging data set in terms of correlating related measurements, a common challenge in collaborative sensing environments. Thus, the measurements captured in the three data sets provide a realistic basis for evaluating our framework.

Third, intrusion detection provides a dynamic and challenging environment for knowledge derivation because there exist a wide variety of approaches to determine trustworthiness of system nodes. We discuss how our framework is able to formalize one approach [54] in order to be able to compare and evaluate it against a number of attacks (section 6.3). Our evaluation involves simulation of several dynamic systems with up to 60 nodes generating ≈ 9000 time stamped test messages over 75 days. The scope of this scenario is realistic for demonstrating the effects of a variety of attacks and evaluating trust assessment approaches on intrusion detection systems.

2.3 Contributions

The establishment of formal definitions for trust, quality, and other metrics as well as the manner in which they are derived from individual data elements and measurements has been mostly ignored in previous research. One intellectual merit of our framework is the formal description of these metrics using an abstract data and relationship graph model. For the first time the proposed formalization enables the comparison and evaluation of different metrics, algorithms, and approaches proposed in literature. The research establishes a unique formalized and comprehensive model for trust, reputation, and opinion approaches that is based on the metrics derived from data. This enables the analysis and comparison of models in a way that is not currently possible due to differences in the definitions of metrics and modeling aspects of the individual approaches (in particular how data is combined to derive trust, reputation and opinions). Since we establish a direct link between the metrics and models we can compare and evaluate the usefulness and impact of individual metrics, data elements, and data sources.

The structure of the framework enables better decision processes because it combines data and relationships between data with notions of quality, trust, reputation, and opinions. Furthermore, as demonstrated in this dissertation the developed framework allows for more realistic modeling of application scenarios since it incorporates context, history, and expected behavior of data. In particular, with our framework we provide a method to select the *best* belief and decision engines among several for specific real world cases. In addition, the formalization of the entire framework allows direct comparison of not only current but future approaches to various metrics (e.g., trust, data quality), models (e.g., trust, reputation, opinions) and decision processes (e.g., trustworthiness of resources, measurement impact, and usefulness).

The major contributions of this dissertation are:

- A new abstract graph modeling approach that allows the management of heterogeneous data with dynamic aspects (e.g., time, location) in a variety of application scenarios while inherently incorporating trustworthiness and data quality assessments

- A new formalization approach to describing belief engines and decision processes in the form of graph *expressions*

- A new framework for knowledge derivation that provides a flexible and extensible approach using clearly defined *extraction*, *processing*, and *evaluation* components

- The means to evaluate and compare different belief, trustworthiness, and decision making techniques in a variety of application scenarios using a formal approach

Today, across all major industries gaining insight from data is seen as an essential part of business. However, while data gathering is becoming inexpensive and relatively easy, data analysis and ultimately deriving trustworthy knowledge from it is increasingly difficult. In many cases, there is the problem of too much data such that important insights are hard to find. As we discuss in chapter 3, several frameworks have been developed that deal with large-scale data processing and analysis. Yet, the problem is often not lack of data but whether the knowledge derived from it is trustworthy. This means distinguishing "good" insights from "bad" ones based on factors such as context and reputation. Still, modeling trust and quality of data is complex because of the variety of conditions and relationships that exist in heterogeneous environments.

Table 2.1 shows how the *TrustKnowOne* framework provides significant benefits over existing state-of-the-art frameworks with respect to major aspects of the knowledge derivation process. A detailed discussion is presented in chapter 8 where the attributes of the *TrustKnowOne* framework confirmed through the implementation of three realistic scenarios are compared to the existing framework's capabilities.

The research presented in this dissertation addresses these issues by describing an approach to fully incorporate trust and quality of data with all its aspects into the knowledge derivation process. Our abstract graph model can be used to model various

Table 2.1: Comparison of major aspects in knowledge derivation processes

Aspect	TrustKnowOne	Hadoop [169, 178]	Dryad [76]	Pregel [107]	Pegasus [86]	GraphLab [104]	EigenTrust [85]	TrustRank [67]	PowerTrust [187]
heterogeneous systems	●	◒	◒	◒	○	◒	○	○	○
dynamic systems	●	◒	○	◒	○	○	○	○	◒
formal representations	●	◒	◒	◒	◒	◒	◒	◒	◒
quality and trust assessments	●	○	○	○	○	○	◒	◒	◒
flexibility in approaches	●	●	●	●	◒	◒	◒	◒	◒

○ no support ◒ partial support ● full support

approaches to trustworthiness and relationship assessment as well as decision making processes. Throughout this dissertation we describe in detail our approaches as well as compare and evaluate them using a series of realistic application scenarios.

In addition, the *TrustKnowOne* framework provides the flexibility and performance necessary for large-scale data processing. In particular, our abstract graph model can be distributed as well as partitioned using a variety of approaches such that the storage of data becomes scalable. Furthermore, processing, assessment, and evaluation approaches are implemented using graph expressions which allows for inherent parallelization and distributed computation.

3 Related Work

The complexity of dealing with heterogeneous and dynamic environments in which we want to incorporate trustworthiness and data quality assessments means that our *TrustKnowOne* framework and graph modeling approaches intersect with a variety of research areas (figure 3.1). In order to provide an overview, we focus our discussion on the three major ones:

- Trust assessment and management
- Data modeling, integration, and fusion
- Large-scale data processing

In this section, we will discuss several research topics and frameworks in these areas as proposed in literature. A detailed comparison and evaluation with respect to our approaches and in particular the *TrustKnowOne* framework described in this dissertation will be performed in a chapter 8.

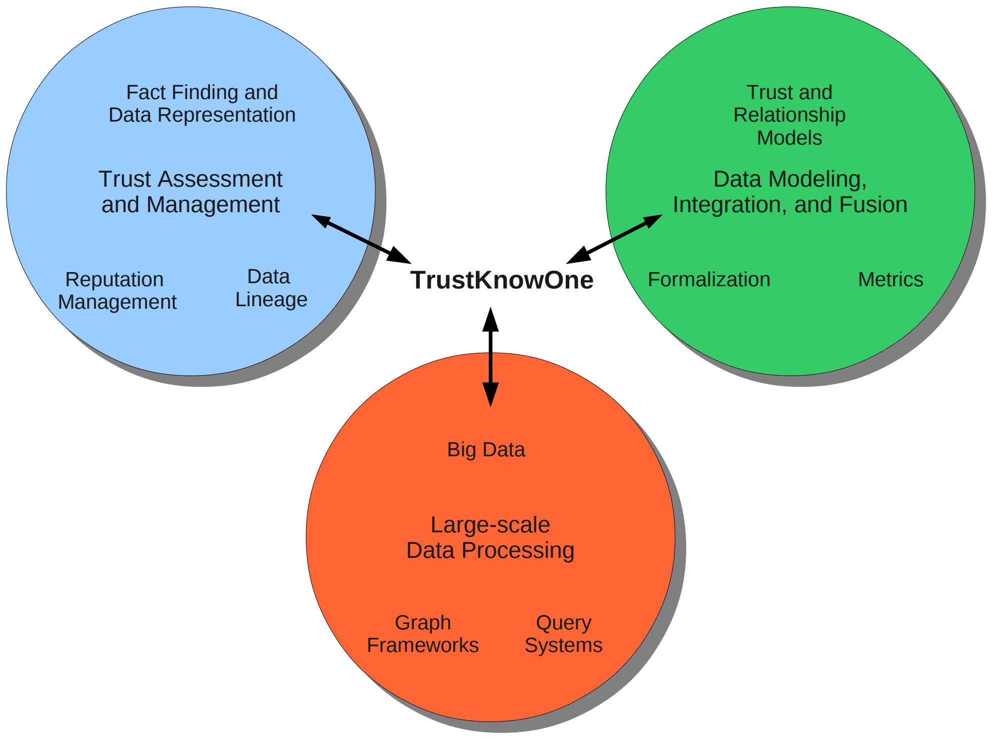

Fact Finding and Data Representation
Trust Assessment and Management
Reputation Management
Data Lineage
TrustKnowOne
Trust and Relationship Models
Data Modeling, Integration, and Fusion
Formalization
Metrics
Big Data
Large-scale Data Processing
Graph Frameworks
Query Systems

Figure 3.1: Related Work Overview

3.1 Trust Assessment and Management

Gupta and Han [65] provide an overview of current developments as well as challenges in the field of heterogeneous network-based trust analysis. In particular, the authors discuss the need for various types of information to be evaluated in terms of trustworthiness and claims to be verified or dismissed based on that evaluation. They identified several areas that are especially in need of incorporating trustworthiness.

Here, we discuss the most relevant with regards to our framework, *fact finding* which deals with asserting the credibility of facts as well as their sources, *reputation management* where trust is incorporating relationships and context, and *data lineage* which provides trace information about where data originated and how it was processed.

3.1.1 Fact Finding and Data Representation

Facts are statements which in the general context is considered to be true. However, identifying facts in large and complex information systems is a difficult problem. For example, news outlets may report a story slightly differently by knowingly (e.g., through subjective opinions) or unknowingly (e.g., inaccuracies) changing facts. If we want to utilize these facts in order to make decisions it becomes clear that we need to establish the correctness of information as well as the trustworthiness of sources.

Fact finding in literature [101, 143, 185] is based on three entities that are modeled as nodes in a graph. *Providers* are data sources that claim *facts* about certain *objects*. The relationship between the entities can then be described using weighted edges that provide positive (supporting facts) or negative (opposing facts) reinforcement.

In a homogeneous network, *fact* and *objects* types are the same and several basic schemes such as voting or ranking can be used to determine which *facts* are best supported by the data. However, this approach is problematic in heterogeneous networks because certain *facts* may be available for one *object* type but not others, aggregated *facts* may be conflicting, and a single *provider* often describes a variety of different *objects*. On the other hand, this variety of data elements and the more complex relationships they form is the main reason why heterogeneous networks tend to have more useful information than homogeneous networks [65].

One of the main premises of our framework is its ability to incorporate heterogeneous data. In particular, our framework utilizes a common abstract data model to address this. Additionally, basic fact finding approaches only use binary indications of true or false for *facts* which are not well suited for most scenarios. We incorporate degrees of truth where probabilities and confidence assessments are assigned to *facts* that can then be used to make decisions.

In addition, one of the disadvantages of using the three entity fact finding model is that it is often too simplistic and therefore unable to describe realistic complex relationships. On the contrary, our framework allows for a formalized representation of data which allows data transformation (i.e., creating derived *facts*) as well as relationships which are often more complex (e.g. dependencies, correlations) than simple positive and negative weights. Another important aspect we address is that data and relationships are dynamic and may evolve over time which requires complex dynamic models that incorporate ideas from dynamic Bayesian networks (DBN) [165] and Hidden Markov Models (HMM) [43, 90]. Basic fact finding approaches ignore this and depend mostly on Bayesian inference models.

3.1.2 Reputation Management

In heterogeneous environments where data may originate from a variety of sources, it is important to assess their reputation. This is particularly interesting in sensor networks where sensors can be seen as independent agents that provide measurements to a collection authority (centralized approach) or that form mesh networks and share measurements as well as information about them with each other (distributed approach). With the growing number of user generated content such as reviews on shopping websites or collaborative radiation measurements using smartphones, effective reputation and by extension trustworthiness management becomes a necessary component in the knowledge derivation process.

A survey by Challa and Momani [26] describes a variety of approaches to managing trust and reputation in different domains. We want to analyze two topics, security and trust approaches, discussed in the survey in more detail. First, across all domains the need for security is apparent. This includes secure communication protocols as well as encryption to protect data. However, security always comes with a cost and in various environments, especially resource constrained ones, it is difficult to balance the performance and security needs of applications. Furthermore, security techniques can not change the fact that data may be inaccurate to begin with due to objective challenges (e.g., environmental factors, calibration issues, time variance, etc.) and subjective challenges (malicious sensor nodes, fabrication of data, impersonation, etc.). Thus, it becomes necessary to build trust management techniques to deal with and efficiently handle different trust, reputation, and opinion issues.

Second, Challa and Momani [26] present an overview of the various methodologies used in the trust management approaches such as weighting, probabilities, Bayesian networks, game theory, and graph theory. In fact, there exist different approaches for trust management in a single domain such as sensor networks [54, 71, 80, 102] as well as there are some approaches that span multiple domains [20, 89]. The problem is that proper evaluation and comparison becomes difficult because techniques and methodologies often utilize custom and domain specific data structures as well as protocols that are hard to adjust.

However, the evolution from basic approaches using linkage of data nodes (see PageRank [20], HITS [89]) to more advanced ones in peer-to-peer networks (see EigenTrust [85], PowerTrust [187]) has led to a variety of trust management approaches that have been adapted to other, related domains, e.g., TrustRank [67] for websites, combating spam in Twitter [98], and secure code execution using commodity computers [122]. In order to improve this process our framework provides a formalization to model trust, reputation, and opinion techniques as will be confirmed through the implementation of three real-world scenarios.

3.1.3 Data Lineage

During any kind of data analysis or decision processes, we often encounter the following. We identified "good" data (i.e., accurate, recent, trustworthy, etc.) and may assume that the source must have been "good" as well. Likewise, we may have found data points that seem to be "bad" (i.e., high variance, old, not trustworthy, etc.) and would like to utilize their sources less. In addition, we face the problem that incorporating “bad” data into our decision is usually worse than missing some of the “good” data.

Therefore, it is important to track the origin of data. While this is usually not a problem in the early stages of data analysis, as the amount of data and the relationships that are formed grows this becomes increasingly difficult [36, 152]. As such, we need to enable the ability to trace individual data points at any stage of the knowledge derivation process which in literature is referred to as providing data lineage.

This means that we need to attach tracing information which includes how data is utilized throughout the entire knowledge derivation process, from the moment we capture data through various forms of processing and ultimately to decisions. It is particularly useful in cases where we have multiple conflicting data points where tracing information could be essential and help us resolve these conflicts through weighting. The problem that needs to be addressed is that by the time we perform decision processes the data has often been preprocessed, transformed or aggregated [36].

An overview of data lineage research is provided by Simmhan et al. [152]. The authors identified several areas of further research. In particular, tracing information is usually added to data management systems instead of being an integral part of them. Furthermore, many of the aspects of the data lineage systems discussed such as granularity, lineage representation, and scalability are domain specific. There is a need for systems that are flexible enough to be applicable across domains, provide varying levels of granularity, and store tracing information in a common, well-defined form. While there has been research in terms of integrating uncertainty into databases and their query systems [16, 36, 146] our framework provides a general approach that is applicable across domains.

3.2 Data Modeling, Integration, and Fusion

Data representation and subsequent processing needs to be flexible and extensible. In order to achieve these goals we need to address *formalization* which provides the ability to model different approaches within and across domains, *trust and relationship models* that need to be fully integrated into data processing, and *metrics* that enable us to evaluate and compare existing and future processes.

3.2.1 Formalization

Various formal approaches have been proposed to overcome problems with managing heterogeneous data in dynamic (time and location variant) environments. Specifically, "Protocol Buffers" [60] and "Thrift" [154] allow data to be structured and efficiently serialized while generating custom interfaces for several programming languages. Still, the primary goal of these approaches is to provide flexible data structures for specific application scenarios. As such their use for describing large scale evolving environments is limited.

The problem of describing data from a variety of sources and combining it is partially addressed by the "Dataset Publishing Language" [62]. However, only one source format (comma separated value text file) is specified. A more flexible approach is the "Data Format Description Language" [135] which allows data formats to be formally described. This formal description works well for structured data but is problematic for unstructured data such as text. Furthermore, the description of meta information is often limited and there is no formal approach that defines relationships between data elements. However, as discussed above, meta information and relationships are important for assessing trustworthiness and quality of data. They need to be incorporated into any formal framework from the beginning instead of added later on.

For sensor networks, researchers [32] proposed a replicated dynamic probabilistic model approach where data produced at each sensor and data consumed at every data collector is modeled probabilistically. The problem is that data collected in such a way will only be bound by the accuracy of the probabilistic models used and hence we lose the original raw data and the ability to assess its trustworthiness.

A similar approach [64] is performing a distributed regression in which the sensor nodes model their local regions and together they fit a global function that represents the sensor data. The authors [64] also point out that sensors which are close to each other often show similar readings. Since the data collected by each sensor is modeled, transmission can be reduced to cases where the actual data read exceeds the predicted data by a certain threshold. Furthermore, each sensor node is able to detect data outliers easily because of its local model. Guestrin et al. [64] also point out that in their modeling approach a single sensor node actually stores the regression coefficients not just for itself but for the entire network.

The basic problem with these modeling approaches is that the data is only approximated and that adaptive data modeling is necessary [64] to deal with natural changes in the environment. However, correctly distinguishing between natural model changes and important events becomes increasingly difficult. Our research provides solutions to these problems by incorporating exact data and relationships as well as changes over time and space.

3.2.2 Trust and Relationship Models

The integration of trustworthiness assessments and relationship models into the data processing component of knowledge derivation is essential. However, there is often a balance that needs to be found between attaching meta information in order to enable trust assessments and the need for high performance processing.

Gupta and Han [65] have identified the following trust analysis research problems and while some of them have been addressed by others [31, 54, 63] we will discuss how our framework addresses important ones that remain [129, 176]. Non-cooperative data sources may or may not provide the useful and trustworthy data that we are looking for [74, 166] and some data sources provide better trust and relationship assessment information about one class of data than others (*cluster-based fact finding*). Furthermore, a piece of data is usually associated with multiple trust and quality assessments (*consensus learning* [55]) and individual assessments are often related to multiple data pieces (*generalization of facts*).

Our framework identifies these relationships as complex but also extremely valuable for trust analysis and incorporates them into the *knowledge extraction* component of the framework. Note that, in order to reduce the increased complexity that comes with heterogeneous relationships some approaches have focused on transforming heterogeneous information networks back into homogeneous networks [8, 183]. However, the two techniques that are generally used have several problems. First, one could determine common attributes that every data pieces contains and only use them thus cutting off information that is potentially valuable (intersection approach). Second, data elements could be extended to include other attributes even if they do not have values for them which leads to both performance and complexity issues (union approach). Since our framework is capable of dealing with heterogeneous data and relationships this transformation is unnecessary thereby eliminating the problems that come with the intersection and union approaches as will be confirmed through the implementation of the *TrustKnowOne* framework.

The assessment of trust relationships is necessary in several areas such as sensor networks [63], intrusion detection [54], and data mining [65]. Several research problems have been stated in surveys [63, 65] which include assessing the correctness of information and trustworthiness of data sources. However, one of the most prevalent issues that arises in existing approaches is that there is no formal approach to specifying trust and relationship models such that they can be compared and evaluated. Our framework presents an approach that allows this formalization through the definition of *metrics* using *expressions* that are evaluated on standardized graph components. Specifically, we provide means to implement algorithms that embody belief engines [124, 127] and decision processes [134, 153].

3.2.3 Metrics

In order to evaluate and compare approaches we have different options. The most commonly used one is to evaluate data processing, trust techniques, and decision engines based on common data sets. However, there are several issues with this. First and foremost, evaluating on common data sets and comparing the performance of results treats evaluation as black-box testing. Thus, the impact of specific components such as trust approaches or decision engines cannot be determined. Furthermore, this option is highly sensitive to implementation, data structures, and the combination of trustworthiness techniques with decision processes.

As such, it is better to integrate mechanisms to evaluate individual components separately. This can be seen as white-box testing where performance and complexity metrics are intrinsic. Our framework implements this approach for various reasons. First, it enables the evaluation of various trust assessment and relationship models (i.e., belief engines) as well as decision processes (i.e., decision engines) separately. Second, specific combinations of belief engines with decision engines can be compared with each other which overcomes one of the biggest problems seen in literature where the results of good trust algorithms are decreased by bad decision engines. Likewise, this allows us to identify bad belief engines whose results are skewed by good decision processes. Therefore, implementing measurable aspects into data processing needs to be seen as an integral part of the entire knowledge derivation process as it enables the performance evaluation and comparison of different approaches effectively.

In addition, one of the main problems remains the variety of metrics for trust analysis. For instance, the most commonly used terms *trustworthiness*, *reputation* and *opinion* are often defined similarly yet utilized differently in literature [30, 31, 54, 63, 88, 114, 122]. Furthermore, researchers introduce additional complexity by breaking down terms [23, 52] such as credibility into reputed credibility, surface credibility and expected credibility. The same complexity issues exist for factors like context, popularity, direct experience and others [58] that influence these trust metrics. How we define all of these not just by using a textual description but actual data relationships is not addressed in literature. Therefore, our proposed approach provides a formalized and data-driven framework for defining trust metrics, how factors affect trust metrics, as well as the computation of trust metrics.

3.3 Large-scale Data Processing

In general, there are two distinct data processing areas that are related to the research performed in this dissertation: *big data* which deals with large-scale data processing and analysis, and *graph frameworks* which model processing of data as directed or undirected graphs. Only recently systems such as GraphLab [104] and Pregel [107] have been proposed to combine these two areas, that is systems intended to perform efficient large-scale graph processing in distributed environments. However, with respect to our approach of integrating trust and quality assessments into the knowledge derivation process, these systems often do not consider relationships, meta information, and trustworthiness assessments. A third research area deals with the need for flexible yet efficient large-scale *query systems* in order to determine relevant data, describe how it should be processed, and provide mechanisms for evaluation.

3.3.1 Big Data

The area of "big data" refers to both distributed and cloud computing. In particular, it aims to address several problems that are inherent in trying to derive knowledge from large amounts of data. These problems span across a multitude of areas and as such often require complex solutions such as distributed file storage and bandwidth-efficient data query systems that are not necessary for smaller data sets.

There are several systems that focus on highly scalable, distributed computing such as Hadoop [169, 178] and Dryad [76] using various approaches. Hadoop is actually a collection of several research areas (e.g., distributed file storage, job scheduling, etc.) that provide an implementation of the MapReduce paradigm [37–39]. The basic idea is that complex tasks are broken down into sub-tasks with each one *mapped* in form of key-value pairs. These mappings can be hierarchical, non-hierarchical, and nested. A list of these sub-tasks are then *reduced* with the results being associated with the respective key. The advantage is that computations can easily be distributed and performed in parallel. A detailed comparison of MapReduce to parallel database systems is also provided by Pavlo et al. [123].

Dryad [76] takes a different approach and provides a distributed execution engine that defines data flows where nodes represent computational processes and egdes communication channels. It automatically deals with the scheduling of tasks in dynamic environments where resources may become available, unavailable or fail. Similar research includes Orleans [22] which models computation in terms of distributed components.

While the described systems provide excellent approaches to dealing with large-scale data processing, they do not integrate trustworthiness approaches and have problems modeling graph structures and complex data dependencies. Note that this is primarily because of the focus on scalability and performance. However, while our framework emphasizes scalability to handle "big data", it also focuses on aspects of flexibility, extensibility, and reusability in order to provide mechanisms for trustworthiness and quality of data assessments as well as decision processes.

3.3.2 Graph Frameworks

There are various techniques for graph model processing, but all have their limitations [25]. Pregel [107] represents a computational model that is based on message passing between nodes of a graph. The focus is on sparse graphs and single types of nodes. Therefore the approach is unsuited for trustworthiness assessments in heterogeneous environments that involve meta information and relationships. DEX [109] describes an approach where data from multiple sources is incorporated into a graph database querying system, but it does not address distributed processing.

Frameworks that focus on machine learning include GraphLab [104], Distributed GraphLab [105], and Orleans [22]. In general, their approach is to provide abstraction layers for algorithms such that distributed processing, parallelism, and scheduling are taken care of by the respective frameworks. Approaches such as Pegasus [86] and SCOAL [40] exploit context knowledge (e.g., many graph mining algorithms can be expressed as matrix multiplications) but are not flexible enough for applications across different domains. Others are limited to a subset of machine learning areas such as the correlation of time-stamped events [174] or provide custom implementations for parallel data analysis (Green-Marl [70]). In general, these frameworks often do not address scenarios with dynamic graphs as there is no easy way to add new data sources, extend the graph model, or change computational processes.

There have also been extensions built on top of existing large-scale processing frameworks such as Hadoop. In particular, Mahout [121] and GBASE [87] aim to provide flexible and generic graph processing approaches. However, using Hadoop as a basis makes it difficult to overcome its limitation when dealing with heterogeneous data, a large number of interdependencies, and dynamic graphs. Furthermore, many of the present graph frameworks focus on dealing with algorithms in homogeneous environments that can be easily scaled using large-scale processing approaches such as Hadoop and parallel database systems. The implementation of complex graphical models like

Bayesian networks [69, 157, 165], Markov models [4, 100, 130], and factor graphs [2, 92] as well as probabilistic reasoning in graphs [44, 124, 179] remains problematic if the underlying processing model (e.g., Hadoop) cannot be adapted or exchanged. Our framework provides extensive flexibility in terms of enabling different input, output, storage, and processing paradigms.

3.3.3 Query Systems

Efficient query systems are important when it comes to analyzing large amounts of data. In general the focus is on performance and expressiveness. A number of higher level query languages exist that run on top of Hadoop [169]. In particular, Pig Latin [118, 172] is a domain specific language for performing queries in Hadoop. Extensions have been developed to perform predictive analysis on Twitter [103]. The approach here is to provide a higher level of abstraction than writing code but more control than declarative languages such as SQL. However, Hive [170, 173] provides this exact functionality where SQL-like queries are compiled into MapReduce instructions.

In similar fashion, several higher level query languages such as DryadLINQ [186] and SCOPE [24] which use a SQL-like syntax have been developed for Dryad [76]. Note that other frameworks have proposed integrated solutions (see Green-Marl [70]). However, unless we choose the same underlying large-scale data processing framework some of the query systems are not available. In addition, none of these existing query systems supports any notion of trustworthiness and relationship assessment. As discussed before there is also no clear distinction between algorithmic modeling in terms of data processing (i.e., belief engines) and decision processes.

Our approach provides a flexible and scalable query system by modeling queries, data processing, and decision processes as graph expressions. These graph expressions utilize relationships (i.e., mathematical, logical, etc.) between data elements that are clearly defined as will be discussed in chapter 4. This formalizes the overall knowledge derivation process and in particular enables evaluation and comparison of different methodologies. Furthermore, our framework allows these formalizations to be described in a variety of ways. (e.g., low-level application programming interface (API), extensible markup language (XML)).

3.4 Chapter Summary

The contribution of this effort is related to a variety of research areas. Here we focused our discussion on the most relevant ones. First, the main premise of our framework is to incorporate trust assessments and management into the knowledge derivation process. As such, a large part of the effort is related to extending *fact finding* approaches to include properties like high accuracy, recency, and usefulness. This is not a trivial task and becomes even more complex when considering that we need to factor in *reputation management* which deals with assessing relationship and data context. Additionally, in order to provide full transparency we also need to make sure that all components of our framework fully incorporate *data lineage*. This means keeping track of where data comes from and how it is processed.

Second, our framework focuses on flexibility rather than performance to be applicable to as many application scenarios as possible. Furthermore, we overcome the common problem in literature of being unable to properly evaluate and compare different approaches through the *formalization* of *trust and relationship models* as well as *metrics*. Our framework utilizes an abstract graph model on which graph expressions are evaluated. Approaches for trust, reputation, and opinion (i.e., belief engines) will be modeled using these expressions. Decision processes such as weighting schemes and Bayesian inference (i.e., decision engines) are described in similar fashion.

Third, scalability as well as extensibility are becoming more and more important in the era of large-scale data processing. Therefore, we need to make sure that our framework meets these requirements to be able to deal with *big data* problems. In order to address this our framework uses a flexible, abstract graph model which allows it to express complex heterogeneous data and their relationships. This leads us to relate our approach to various other *graph frameworks* where data processing is modeled using directed or undirected graphs. However, one of the biggest problems that often remains is the ability to provide an scalable and expressive *query systems*.

This chapter discussed several approaches discussed in related literature. In particular, it highlighted some of their shortcomings. The main benefit of the *TrustKnowOne* framework is that trustworthiness and relationship assessments are directly incorporated into a flexible and scalable knowledge derivation process. An in-depth evaluation of our framework and a comparison to existing approaches described here is performed in chapter 8.

4

Berlin - An Abstract Graph Model for Knowledge Processing Using Graph Expressions

The foundation of the framework developed here is *Berlin*, an abstract graph model on which processing and inference is performed. There are several reasons for using an abstract graph model. First, describing data in a uniform and standardized manner allows for a systematic and clear approach to processing. Second, all processing, inference and decision making can be made using graph operations. Third, dynamic data is managed simply by the addition or removal of nodes and edges.

We define our graph model as follows. A basic description of a piece of data and its attributes is an *element*. A particular *element* with attribute values is an *element node*. Each *element node* consists of multiple *element instances* which are timestamped. In order to be able to deal with dynamic graphs, each piece of information needs to be associated with a particular time instance. This allows for ordering of values in time series data. We have several options for achieving this. First, we can designate a particular data attribute to be the time instance. Second, if there is no such data attribute, we can choose the time of import into the graph model as the time instance. Third, we can specify a certain time independently.

Because *element instances* may contain only values for specific attributes, an *element node* can be seen as a sparse matrix that contains attribute values ordered by time where values that do not change do not need to be stored. Furthermore, every *element node* is uniquely identifiable through an *identifier* which may be an attribute of the *element node* or explicitly assigned. This solves the problem of having to check the entire graph whenever new information is added.

The connection between two *elements* is described through a *relation* which can be defined implicitly or explicitly and may contain attributes, e.g., weights or location information. A *relation edge* connects two *element nodes* and contains timestamped *relation instances* in a similar manner to *element instances.*

As such, our framework deals with two types of graphs. The *element description graph* keeps track of the basic descriptions of *elements* and *relations* as well as implicit meta information (i.e., basic *metrics* and *dimension models*). The *element instance graph* contains the actual *element nodes* and *relation edges* with all their values and instances.

In order to incorporate local trust aspects for attribute values such as deteriorating sensor accuracy over time, our approach associates *dimension models* with individual *attributes.* These *dimension models* express confidence and trust assessments for attributes in a probabilistic or deterministic manner. As such, we evaluate how, among other dimensions, time and location can affect values in the abstract graph model.

Belief engines representing trust and quality of data assessments as well as decision processes are implemented using graph *expressions.* These *expressions* range from straightforward mathematical computations to complex relationship-based techniques and can be combined hierarchically to make them flexible and extensible.

In the following sections, we will discuss the use of all of these graph abstractions in more detail.

4.1 Graph Components

In this section, we will describe the basic components of our abstract graph model *Berlin*. Specifically, we extend basic graph theoretical approaches in order to incorporate the ability to model trust and relationship assessments.

4.1.1 Elements

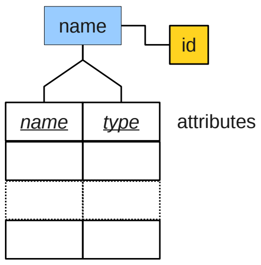


Figure 4.1: An *element* description which includes a uniquely identifiable name, an id reference, and a list of attributes and their types

Elements represent descriptions of the basic pieces of information that inference is made upon. They can be thought of as different types or classes defining *element node* objects. This means that we are able to deal with heterogeneous data fusion applications, overcoming limitations of other homogeneous graph models that only consider one type of node. Each *element* as shown in figure 4.1 is uniquely identifiable by its *name* and contains an id reference and definitions of its attributes in the form of name-type pairs.

Definition 4.1 Attribute

We define an *attribute* as

$$a = \{name, type, \{\varphi_1 \dots \varphi_n\}\}$$

where *name* is the name of the attribute, *type* the class of possible attribute values and $\{\varphi_1 \ldots \varphi_n\}$ an optional set of *dimension models* describing the attribute's value range, distribution, and constraints.

Our framework provides a flexible type system for these attributes in which a set of common well-known types is provided but can be easily be extended by custom definitions. Furthermore, we are able to attach time, location, and value *dimension models* to attributes. These can be used by the belief engines to determine trustworthiness and quality aspects of attribute values. Hence, we can formally define an *element* as follows.

Definition 4.2 Element
Let $A = \{a_i, \ldots, a_n | a_i.name \neq a_j.name \ \forall a \in A\}$ be a set of attributes then an *element* is defined as

$$E = \{name, ID, A\}$$

where ID is a function which is able to uniquely identify *element nodes* derived from the *element* E.

Note that we need to specify how individual *element nodes* derived from an *element* are identified. The reason is that meta data and additional information has to be correctly correlated throughout the knowledge derivation process. This is especially important for handling dynamic graphs in our framework where we need to check whether nodes that are being added already exist in the graph. Note that we provide several options for performing this identification. It can be done explicitly through a serial id that is assigned to each new *element node* or implicitly by having one or a combination of attributes represent its identity.

4.1.2 Element Nodes

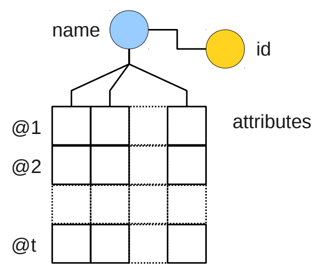


Figure 4.2: An *element node* consisting of a specified id reference and timestamped attribute value instances

Particular objects in our graph model that contain data values are *element nodes*. As shown in figure 4.2 they contain a *sparse* table of attribute values where each row is an *element instance* that is identified by a particular time instance.

> **Definition 4.3** Attribute-value pair
>
> Let a be an attribute and $value$ the specified value for the attribute, then their *attribute-value pair* is defined as:

$$av = a \cup \{value\} = \{name, type, \{\varphi_1 \dots \varphi_n\}, value\}$$

As we incorporate more information over time the table will grow. However, note that we only need to store information that changes from one instance to another thus saving space and inherently making the table *sparse*.

This *sparse* table approach has several advantages over creating new nodes for every *element instance*. First, it makes time series analysis straightforwards as we keep related information close together. Second, space complexity is reduced since values that do not change do not require additional storage space. Third, we do not have to perform any additional graph operations in order to perform time series analysis and correlation. Furthermore, it simplifies the management of the abstract graph model since it keeps the number of nodes and edges in a dynamic graph low (compared to a graph containing separate nodes for each *element instance*).

Note that, *element instances* may contain additional "non-descriptive" (not previously described) attributes such as derived information for which values can be added on-the-fly. This gives our approach more flexibility and leaves room for enhancements to the graph model in later stages.

Definition 4.4 Non-descriptive attribute-value pair
A "non-descriptive" *attribute-value pair* is a *attribute-value pair* without a specified *type* and *dimension models*

$$av' = \{name, value\}$$

We can combine the previously defined and the "non-descriptive" *attribute-value pairs* to define an *element instance*.

Definition 4.5 Element instance
Let *av* be a particular *attribute-value pair*, *e.A* the set of attributes for the *element node e*, and *av′* an additional "non-descriptive" attribute of the *element instance*, then an *element instance* is a collection of attribute values at a specific time instance *t* defined as

$$e_t = \{\{av_1 \dots av_n | av \in e.A\}, \{av'_1 \dots av'_m\}\}$$

The collection of *element instances* makes up the *sparse* table of attribute values.

Definition 4.6 Element instance collection
The collection of *element instances* can be defined as

$$ei = \{e_1 \dots e_T\}$$

where $ei(t) = e_t$ acts as a mapping function from a time instance t to the specific *element instance* e_t. Furthermore, we define the ordered set of time instances ei_t as follows

$$ei_t = \{t_0 \dots t_T | t_i < t_{i+1}\}$$

Note that *element nodes* are derived from *element* descriptions. Thus they contain all possible attributes specified in the respective *element* and the mandatory id follows from the description as well (i.e., can be explicit or inferred attribute reference, combination of attributes, auto generated).

Definition 4.7 Element node

Let E be a particular *element*, then an *element node* can be defined as

$$e = E \cup \{id, ei\} = \{name, ID, A, id, ei\}$$

where $id = ID(ei)$ is the result of the identification function applied to all attribute values since the specific *identifier* of the *element node* is either attribute based or explicitly defined and ei the collection of *element instances*.

4.1.3 Relations

Two *elements* are connected if there exists a *relation* consisting of a defined source and target *element* between them as shown in figure 4.3. Because *elements* may share more than one *relation* it is necessary to group or organize them by defining unique *names*. Furthermore, we are able to describe more complex *relations* by attaching *metrics* to them. Formally, a *relation* is:

Definition 4.8 Relation

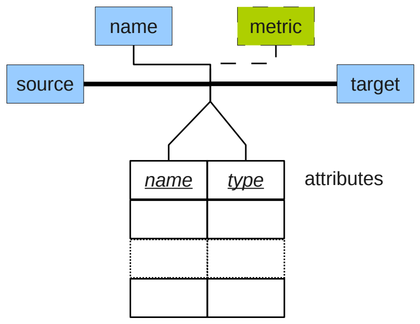


Figure 4.3: A *relation* which is defined by a unique name, information about the two elements that it connects, and a list of attributes and their types. This may include an optional *metric* determining existence of the relation.

Given the set of all element descriptions $\hat{E} = \{E_1 \ldots E_I\}$ let $S, T \in \hat{E}$ be source and target *element* definitions, A a set of attributes, and M an optional *metric*, then a *relation* is defined as

$$R = \{name, S, T, A, M\}$$

In the case where we specify a *relation* without defining a *metric*, the derived *relation edges* will always exist. In case there is a *metric*, a *relation edge* exists only if the *metric* evaluates to `true` and does not exist if it evaluates to `false`.

Definition 4.9 Relation existence

The *relation* R always exists if the *metric* M does not exist since it represents an optional qualifier of existence for each *relation*. If a *metric* M is specified, then the *relation* only exists if the application of the *metric* to the *relation*, noted as $M(R)$, yields `true`.

However, when we describe *relations* in the *knowledge extraction* phase of the framework we need to take the following into consideration. Since there is no additional or meta information available yet, we can define only basic relationships such as equality comparisons between attributes (e.g., same owner, same sensor type).

In the *knowledge processing* stage we have more information available. Therefore, we can establish two additional types of relationships. First, there are *explicit relations* that are defined based on context and meta information (e.g., temperature ranges, rankings). Second, *implicit relations* can be derived during *knowledge processing* by analyzing the data in the graph model using belief engines. For example, relationships between *element* attributes may be discovered using correlation techniques [79, 127, 130].

4.1.4 Relation Edges

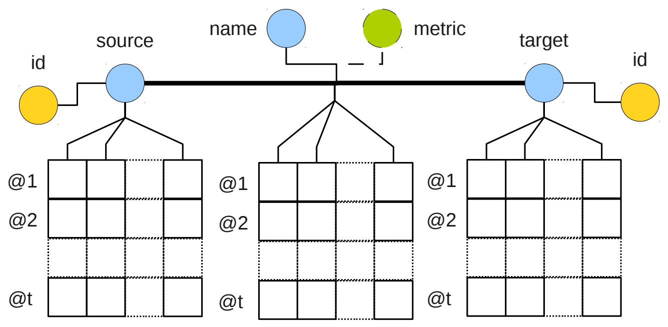


Figure 4.4: A *relation edge* consisting of a name, the *element nodes* it connects which are identifiable by their type, and timestamped attribute value instances. An optional *metric* may be attached to allow more complex relationships to be defined.

Particular instances of *relations* are *relation edges* which describe the relationship between two *element nodes*. As shown in figure 4.4 the source and the target *element nodes* are specified by their id. Note that these *element nodes* are only valid if the *element* type is the same as specified by the *relation*. Since *relations* have unique names this allows us to easily group *element node* neighbors by "type" (similar, same owner, etc.) based on the name of a *relation*. A *relation edge* also maintains attributes in the form of a *sparse* table where each time instance refers to a specific *relation instance*. This is similar to the way attribute values are stored in *element instances* which means that the *relation instances* may contain additional "non-descriptive" attributes as well.

Definition 4.10 Relation instance

Let *av* be a particular *attribute-value pair*, $r.A$ the set of attributes for the *relation edge* r, and av' an additional "non-descriptive" attribute of the *relation instance*, then an *relation instance* is a collection of attribute values at a specific time instance t defined as

$$r_t = \{\{av_1 \dots av_n | av \in r.A\}, \{av'_1 \dots av'_m\}\}$$

Thus, the *sparse* table of attribute values becomes

Definition 4.11 Relation instance collection

The collection of *relation instances* can be defined as

$$ri = \{r_1 \dots r_T\}$$

where $ri(t) = r_t$ acts as a mapping function from a time instance t to the specific *relation instance* r_t. Furthermore, we define the ordered set of time instances ri_t as follows

$$ri_t = \{t_0 \dots t_T | t_i < t_{i+1}\}$$

The collection of *relation instances* makes up the *relation edge.* As such, it is derived from the formal definition of a *relation.*

Definition 4.12 Relation edge

Let R be a particular *relation*, then a *relation edge* can be defined as

$$r = R \cup \{s, t, ri\} = \{name, S, T, A, M, s, t, ri\}$$

where s and t are *element nodes* matching *element* descriptions S and T

respectively and *ri* the collection of *relation instances.*

Because of the ability to model dynamic graphs we need to realize that a metric specifying a particular relationship may yield `true` for some *relation instances* and `false` for others. In our approach, we keep track of these time and location variant relationships.

Definition 4.13 Relation edge existence
Let *ri* be the collection of *relation instances* for the *relation edge* r and $M(r_t)$ the result of the metric M applied to the *relation edge* at time t then we say that in general the *relation edge* exists if

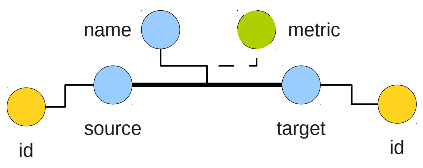
$$\exists r_t \in ri | M(r_t) = \texttt{true}$$

We can utilize this notion of a relationship to allow belief engines to infer properties such as strength and connectivity in dynamic graphs where relationships may change over time.

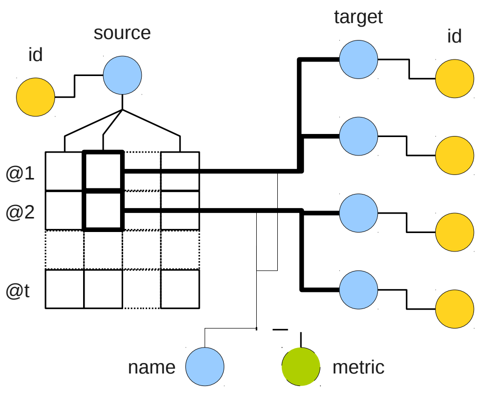
Definition 4.14 Relation edge strength
Let *ri* be the collection of *relation instances* for the *relation edge* r and $M(r_t)$ the result of the metric M applied to the *relation edge* at time t then the strength of the *relation edge* is defined as

$$r_{strength} = \frac{|\{r_t | r_t \in ri, M(r_t) = \texttt{true}\}|}{|ri|}$$

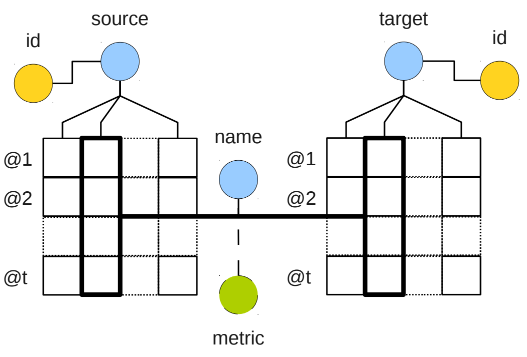
In general, we model relationships as edges between two *element nodes.* This approach balances the need for relationship detail with efficient computation and modeling aspects. In particular, by using this approach we are able to incorporate the following detailed relationships that would otherwise require more complex solutions (see figure 4.5).



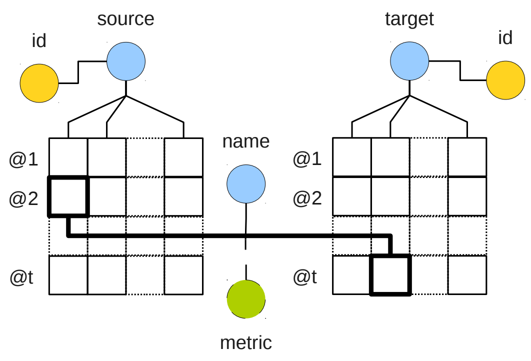
(a) *element node* to *element node*



(b) *attribute* to *element node*



(c) *attribute* to *attribute*



(d) *value* to *value*

Figure 4.5: Relationship types that can be inherently modeled using *element node* to *element node* definitions.

Element node - element node Two *element nodes* are related (as connected by a *relation edge*) by some definition that incorporates their *element* types and attribute values as well as potentially other relations (figure 4.5a). This is the most general type of relationship and can, for example, be used to determine that two sensors tend to have similar temperature readings by defining a *metric* that incorporates both temperature attributes and a specific range (an example of such a *metric* is discussed in expression 4.9).

Attribute - element node A particular attribute of an *element node* can be related to another *element node* (figure 4.5b). For example, if ownership of a sensor is modeled as an attribute of a sensor *element* then we could define a relationship “same owner” between sensors with the same ownership attribute value. Note that this aspect can be modeled on an *element node* to *element node* basis using a metric that determines if the ownership attributes are the same.

Attribute - attribute Correlations between two attributes are often of interest (figure 4.5c). In the case of a sensor, we may be interested to know if there is a relationship between a sensor’s location and radiation level measurements. Furthermore, attributes of two different *elements*, e.g., different sensor types (one measures only temperature and the other measures only rainfall) can be correlated as well. By using the *element node* to *element node* approach we are able to determine these relationships by defining *metrics* that incorporate attribute values from a variety of *element nodes* at different levels of detail.

Value - value A specific value may be the result of a sequence of circumstances (figure 4.5d) such as when sensors tasked with the monitoring of cargo can cause an event chain of alerts. In this case, it is important to be able to model data provenance (lineage) which, in our abstract graph model, can be achieved using *metrics* that take into consideration attribute values across time instances.

The key here is that inference is primarily made on the *element nodes* and that particular relationships types such as attribute - element node, attribute - attribute and value - value can be seen as describing certain aspects of relationships between *element nodes*. Doing so decreases the complexity of the abstract graph model while maintaining the ability to model complex relationships between various pieces of diverse data.

4.2 Graph Types

As part of our framework we distinguish between two types of graphs. On the one hand, the *element description graph* represents a blueprint of all the *elements* and *relations* that need to be modeled for a specific application scenario. On the other hand, the *element instance graph* contains the actual *element nodes* and *relation edges* with their respective data. Note that while both types of graphs can be dynamic, usually only the *element instance graph* encounters changes in values, *element nodes*, and *relation edges*.

4.2.1 Element Description Graph

The first step in our framework is the formal description of *elements* and their *relations* that make up the abstract graph model. This information is stored in the *element description graph*.

Definition 4.15 Element description graph

Let E be an *element*, R a *relation*, M a *metric*, and Φ a *dimension model* then the *element description graph* is defined as

$$EDG = \{\{E_1 \dots E_I\}, \{R_1 \dots R_J\}, \{M_1 \dots M_K\}, \{\Phi_1 \dots \Phi_L\}\}$$

4.2.2 Element Instance Graph

Particular values and graph element instances are stored in the *element instance graph*. This graph can be augmented and extended as more information becomes available and is incorporated into the knowledge derivation process.

Definition 4.16 Element instance graph

Let e be an *element node*, r a *relation edge*, m a *metric instance*, and φ

a *dimension model instance* then the *element instance graph* is

$$eig = \{\{e_1 \ldots e_i\}, \{r_1 \ldots r_j\}, \{m_1 \ldots m_k\}, \{\varphi_1 \ldots \varphi_l\}\}$$

This *element instance graph* is fully dynamic where every data value is timestamped as described earlier.

4.2.3 Graph Transformations

While the topic of graph transformations has been extensively covered by [6, 91, 164], we briefly discuss one possible approach using our framework here. In particular, a *transformation* consists of two main parts, a *pattern* and a *replacement*. We can utilize a set of *metrics* to determine if a certain part of the graph matches a particular *pattern*. Furthermore, since *metrics* are expressions describing information in a graph we can apply them to the matched *pattern* forming a respective *replacement* sub-graph.

Definition 4.17 Graph transformation
Given the set of all *metrics* $\hat{M} = \{M_1 \ldots M_K\}$ let $P, RE \in \hat{M}$ be a *pattern* and *replacement* then a *transformation* can be defined as

$$T = \{P, R\}$$

Note that because *metrics* can be nested we can essentially replace a *pattern* with multiple *replacements* as well, which means that we can apply a *transformation* as follows.

Definition 4.18 Graph transformation application
Let $P = \{M_1 \ldots M_n\}$ be a *pattern* consisting of a set of *metrics* then a

particular subgraph G matches the pattern if

$$P(G) = M_1(G) \wedge \cdots \wedge M_n(G) = \texttt{true}$$

Let $RE = \{M_1 \dots M_m\}$ be a *replacement* consisting of a set of *metrics* such that

$$RE(G) = \{M_1(G), \dots, M_m(G) | M \in RE\}$$

then the *transformation* $T = \{P, RE\}$ is applied to the subgraph G as

$$T(G) = \begin{cases} RE(G) & \text{if } P(G) = \texttt{true} \\ G & \text{if } P(G) = \texttt{false} \end{cases}$$

In the following, we discuss two possible applications of graph transformations in our framework.

Elements of Interest Note that while all relevant *elements* are stored in the *element description graph*, instead of considering all *elements* and *relations* in a graph for inference purposes, we may choose to select a subset within a domain or application scenario. This can be achieved by designating *elements of interest* and using graph transformations to convert attributes to *elements* as well as to fold *elements* that are not of interest into attributes. In particular, additional data not of interest to the application can be treated as an attribute of an *element*. For example, if we are interested in sensor values but not ownership, we may relegate owner to be an attribute of the *element* sensor as shown in figure 4.6. In addition, if *elements* have been "reorganized" into *elements of interest*, the *element instance graph* will reflect this as well.

Element Nodes of Interest Furthermore, other subgraphs that can be extracted by similar graph transformations include time specific graphs (by slicing across groups of particular instances with a specific time instance) and hierarchical graphs (logical or

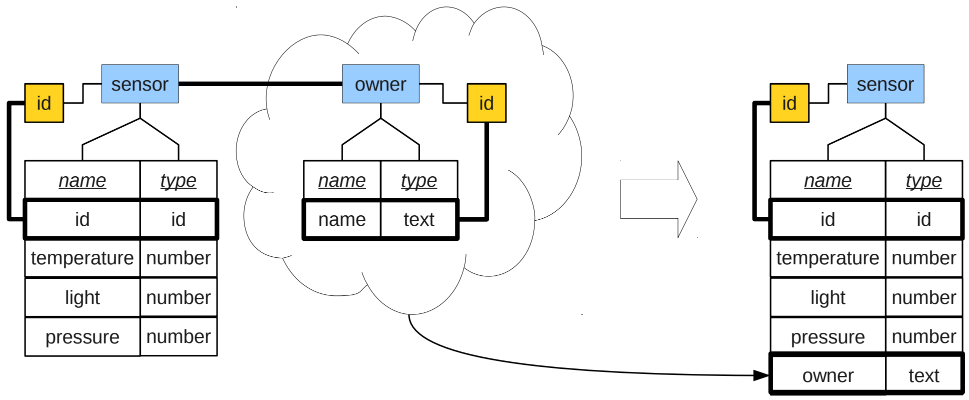


Figure 4.6: An example of defining "sensors" as elements of interest and relegating "owner" *elements* to attributes using a graph transformation.

physical groupings of *element nodes*).

4.3 Dimension Models

As discussed above, attributes can be associated with a set of *dimension models* describing the attribute's value range, distribution, and constraints. The purpose of *dimension models* is to provide *belief engines* with information to assess trustworthiness and data quality on a "local" attribute level. They may be probabilistic (e.g., values following certain distributions) or deterministic (e.g., at a specific times of the day the location is "office" and otherwise "home" or "in transit") as well as time and location variant. This allows us to model applications where the meaning of values is different depending on some dimension.

There exist a variety of dimensions such as time, location, and other attribute values that could affect the trustworthiness assessment of an individual attribute. For instance, we can assess slowly degrading sensors where the accuracy of measurements taken is reduced over time dynamically. Within a dimension, we describe particular instances as contexts, e.g., specific dates for time dimensions and places for location dimensions. This approach has the advantage that we do not have to rely on static error models.

Here, we discuss the definition of a *dimension model*.

Definition 4.19 Dimension model

Let θ be a specific context within a dimension Θ then a *dimension model* is defined as the mapping function

$$\Phi(value|\theta) = \vartheta$$

where the value ϑ is derived for the specific attribute *value* given the context θ.

In general, we can categorize *dimension models* into the following types. First, consider cases where the context is drawn from a finite set of contexts (e.g., specific locations, sensor types, owners).

Definition 4.20 Discrete dimension model

Let Θ be the set of contexts θ within a dimension defined as

$$\Theta = \{\theta_1 \dots \theta_d\}$$

then we can define the mapping functions φ

$$\varphi_i(value|\theta_i) = \vartheta$$

where the value ϑ is derived for the attribute *value* given the specific context θ_i. One option is to define a discrete *dimension model* $\Phi_{discrete}$ as the set of such mapping functions and a default function $\varphi(value) = \vartheta$ which does not require any context

$$\Phi = \{\{\varphi_1 \dots \varphi_l\}, \varphi\}$$

such that its application to a particular attribute *value* is defined as

$$\Phi_{discrete}(value|\theta) = \begin{cases} \varphi_i(value|\theta = \theta_i) & \text{if } \theta \in \Theta \\ \varphi(value) & \text{otherwise} \end{cases}$$

The other options is to specify a discrete mapping function φ in the form of

$$\varphi(value|\theta) = \vartheta$$

where the value ϑ is derived for the specific attribute *value* given the discrete context variable θ. The discrete *dimension model* $\Phi_{discrete}$ in this case is

$$\Phi_{discrete}(value|\theta) = \varphi(value|\theta)$$

Second, there are models such as range constraints that are independent of the context. Our approach is to treat this as a special case of the discrete *dimension model.*

Definition 4.21 Static dimension model
The special case of a *dimension model* for an attribute *value* that does not depend on any particular context θ concerns a single mapping function. Thus, given the general definition of a discrete *dimension model* $\Phi_{discrete}$ we only need to specify the default function φ such that

$$\Phi_{static}(value) = \varphi(value)$$

Third, instead of specifying a discrete set of contexts, we can define a continuous model for all context values. There are two options for doing so. On the one hand, we can extend the discrete *dimension model* to include an interpolation (i.e., smoothing or regression) function which allows us to derive values in between contexts. On the other hand we can simply ignore discrete contexts and define a specific mapping function that incorporates all contexts.

Definition 4.22 Continuous dimension model

Given a discrete *dimension model* $\Phi_{discrete}$ we need to carefully define the range of contexts Θ. We require that the set of contexts Θ be ordered in the manner $\theta_i < \theta_{i+1}$. This aspect allows us to interpolate between different contexts for which we require an additional parameter Ξ, the interpolation (i.e., smoothing or regression) function. Furthermore, the first element θ_1 and the last element θ_d represent boundaries for the range of the continuous *dimension model*, i.e., $[\theta_1, \theta_d]$. A such, a continuous *dimension model* can then be defined as

$$\Phi_{continuous}(value|\theta) = \begin{cases} \Xi(value|\theta) & \text{if } \theta_1 \leq \theta \leq \theta_d \\ \varphi(value) & \text{otherwise} \end{cases}$$

If we choose not to base our continuous *dimension model* off a discrete *dimension model* then we must define the mapping function φ in the form of

$$\varphi(value|\theta) = \vartheta$$

where the value ϑ is derived for the specific attribute *value* given the continuous context variable θ. The continuous *dimension model* $\Phi_{continuous}$ is then defined as

$$\Phi_{continuous}(value|\theta) = \varphi(value|\theta)$$

In the following, we focus our discussion on some of the major dimensions and provide example *dimension models* accordingly. Note that since the static *dimension models* are independent of the particular context we provide a separate example here.

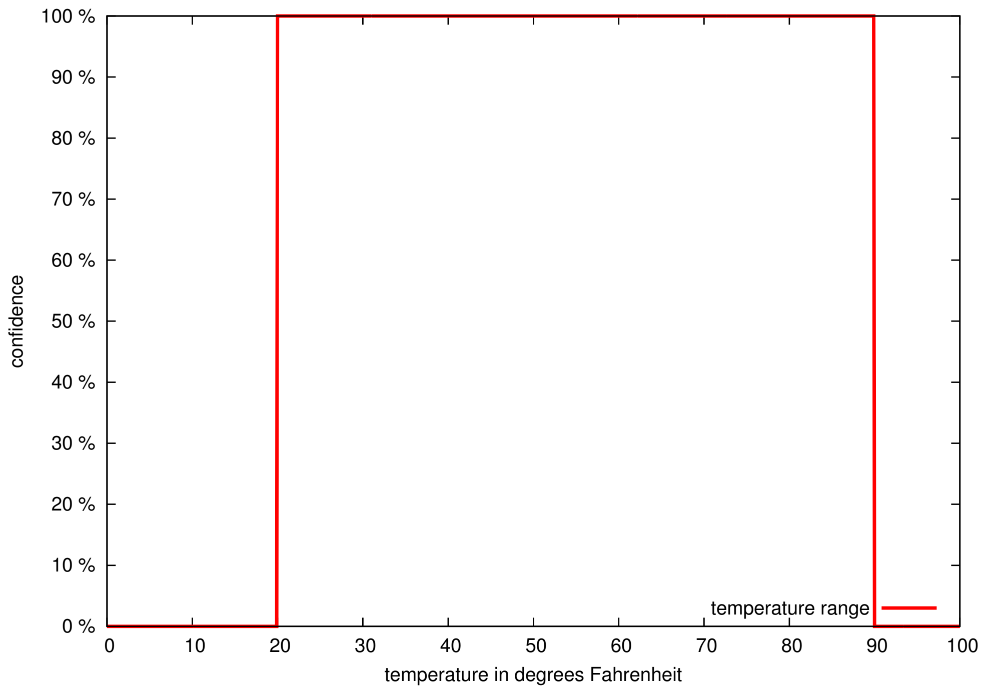


Figure 4.7: The confidence in a temperature measurement based on a specified range of valid temperatures, i.e., $T_{min} = 20$ and $T_{max} = 90$

Static Constraint This reflects the most general case where trustworthiness aspects are "local" but do not change over time and are not dependent on other context. An example would be to constrain the range of valid temperature measurements as shown in figure 4.7 which we could define as follows.

Let T_{min} be the minimum and T_{max} the maximum temperature a particular sensor is designed for then the confidence (from 0% to 100%) in the temperature range can be defined in terms of the default function φ

$$\varphi(value) = \begin{cases} 0\% & \text{if } value < T_{min} \\ 100\% & \text{if } T_{min} \leq value \leq T_{max} \\ 0\% & \text{if } value > T_{max} \end{cases}$$

where *value* represents a particular attribute value. Note that in this case there is no

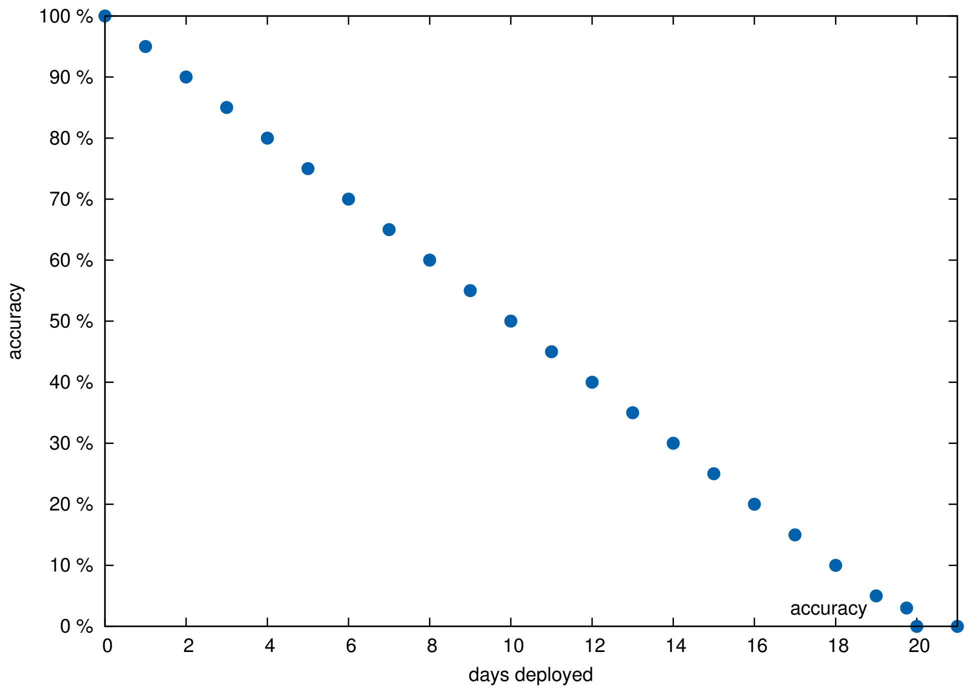


Figure 4.8: The accuracy of a sensor's measurements based on the number of days deployed

context θ necessary.

4.3.1 Time

The accuracy and by extension trustworthiness of an attribute value may depend on the time context at which it is evaluated. Here, we choose as an example a sensor scenario and describe several use cases for particular *time dimension models.*

Discrete Time We can model a sensor with slowly degrading accuracy using a discrete *dimension model.* For example, consider sensors that report temperature values once a day. Furthermore, let us assume the quality of the sensors is low such that each day their accuracy decreases by 5%. This means that on the first day accuracy is 100% and 0% on the 21st day and beyond (figure 4.8).

Hence, the set of contexts Θ can be defined as the number of days deployed

$$\Theta = \{\theta_1 = 0, \theta_2 = 1, \dots \theta_{20} = 19\}$$

where the mapping functions φ follow the pattern

$$\varphi_i(value|\theta_i) = 100\% - 5\% \times \theta_i$$

accordingly. We then express the default function for the remainder of the days as $\varphi(value) = 0\%$.

Continuous Time As the number of contexts to be modeled grows it is often a better approach to choose a continuous *dimension model*. For example, if we wanted to extend the simple accuracy model from above to be able to determine accuracy for particular temperature values gathered throughout a day instead of only daily (figure 4.9).

This would require us to specify a large number of contexts and mapping functions. Here, we use the continuous *dimension model* that solves the problem by defining a mapping function φ for a continuous context variable θ

$$\varphi(value|\theta) = max\left(0\%, 100\% - 5\% \times \frac{\theta}{60 \times 60 \times 24}\right)$$

where θ represents the time a sensor has been deployed in seconds.

4.3.2 Location

The location of sensors has a direct impact on the accuracy of measurements. For instance, consider a radiation detection sensor as will be discussed in section 6.2.

Discrete Location In order to be able to compare radiation levels of two sensors we need to establish common measurement parameters. Safecast [144] (as used in

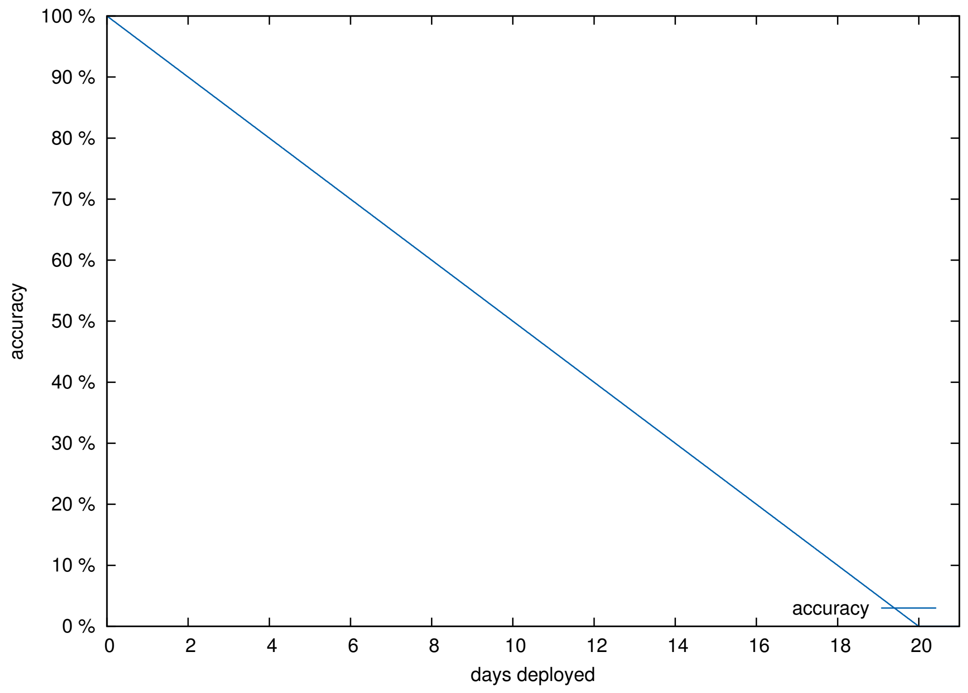


Figure 4.9: The accuracy of a sensor's measurements based on a slowly degrading continuous function

section 6.2) discusses the problem of inaccuracies based on the height of the sensor during measurements. In particular, radiation levels are different at various heights. For instance, Safecast [144] suggests 1m above the ground instead of ground level to determine radiation levels. Furthermore, there are differences depending on whether measurements are taken inside or outside of buildings.

Hence, let the set of contexts Θ represent the approximate height in meters at which the sensor took measurements

$$\Theta = \{\theta_1 = 0m, \theta_2 = 1m\}$$

then we can define the following mapping functions φ to determine accuracy of the

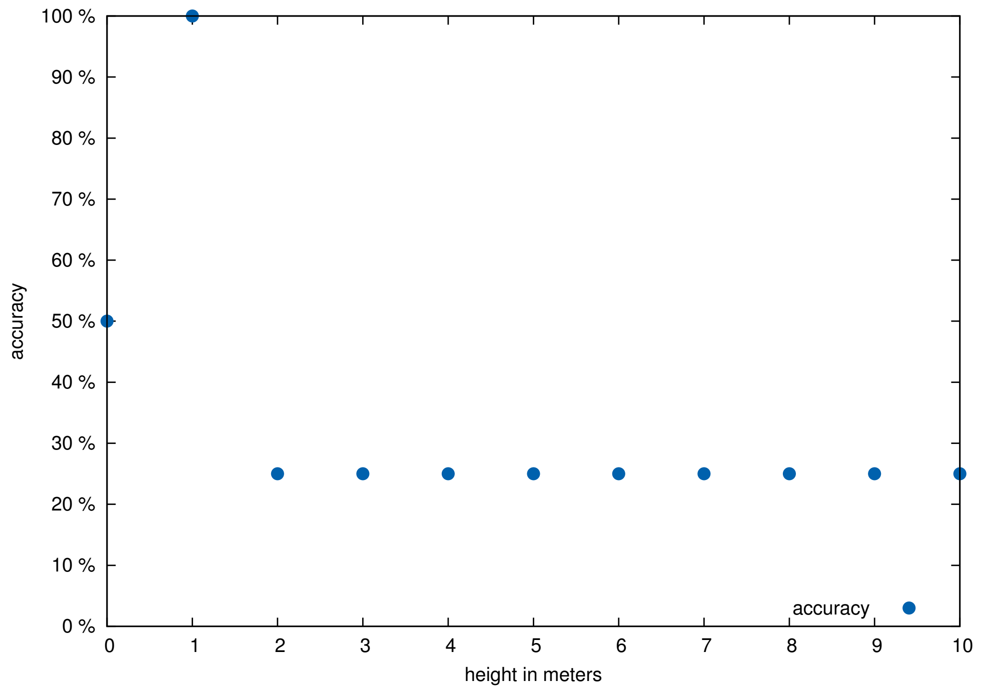


Figure 4.10: The accuracy of a sensor's measurements based on the approximate height in meters

measurements

$$\varphi_1(value|\theta_1) = 50\%$$

$$\varphi_2(value|\theta_2) = 100\%$$

accordingly. We then express the default function for other possible heights as $\varphi(value) = 25\%$. The resulting discrete *dimension model* is shown in figure 4.10

Continuous Location Some sensors are very sensitive to environmental factors. While many stationary sensors can be calibrated in a way that reduces noise from these factors, mobile sensors need to continuously adapt. Therefore, the accuracy of measurements of mobile sensors should carefully evaluated. Here, assume that we are tracking cargo that is transported by rail (a scenario described in [53, 93, 94]). We could

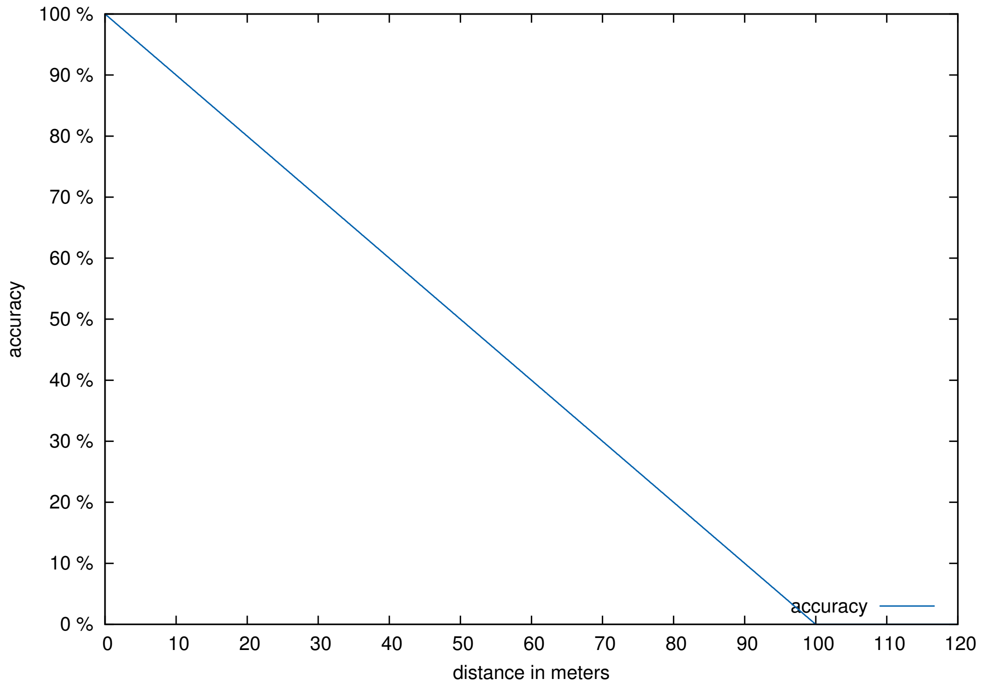


Figure 4.11: The accuracy of a sensor's location based on the estimated distance from tracks

define a location accuracy assessment by determining how far measurements coordinates are from train tracks (figure 4.11).

In this case we would employ a continuous *dimension model* with a context variable θ representing measurement coordinates such as

$$\varphi(value|\theta) = max\,(0\%, 100\% - 1\% \times \texttt{distance from tracks in meters})$$

4.3.3 Value

Dimensions other than time and location can be chosen as well. For example, the trustworthiness of a particular attribute may depend on other related attributes. In the following we give some examples for this case.

Discrete Value There exist various sensor types with a variety of advantages and disadvantages. Furthermore, sensors are often deployed by different entities for application specific scenarios. Here, we want to model the level of confidence we have in a particular sensor's measurements based on the owner.

Let Θ represent several potential owners

$$\Theta = \{\theta_1 = government, \theta_2 = public, \theta_3 = private\}$$

and define confidence as a set of the following mapping functions φ

$$\varphi_1(value|\theta_1) = 90\%$$
$$\varphi_2(value|\theta_2) = 50\%$$
$$\varphi_3(value|\theta_3) = 70\%$$

Furthermore, let the default function for all other owners be $\varphi(value) = 50\%$. The resulting discrete *dimension model* represents an example of a deterministic model for "local" trust assessment.

If we are interested in assessing confidence in temperature measurements and have other information available such as a basic description of the conditions (e.g., sunny, cloudy, rainy, snowing), we can express this confidence using a fuzzy approach [15, 126, 141]. As such, we associate every weather condition with a particular temperature range model (see figure 4.12).

Hence, we could define the context Θ as the set of possible weather conditions

$$\Theta = \{\theta_1 = snowing, \theta_2 = rainy, \theta_3 = cloudy, \theta_4 = sunny\}$$

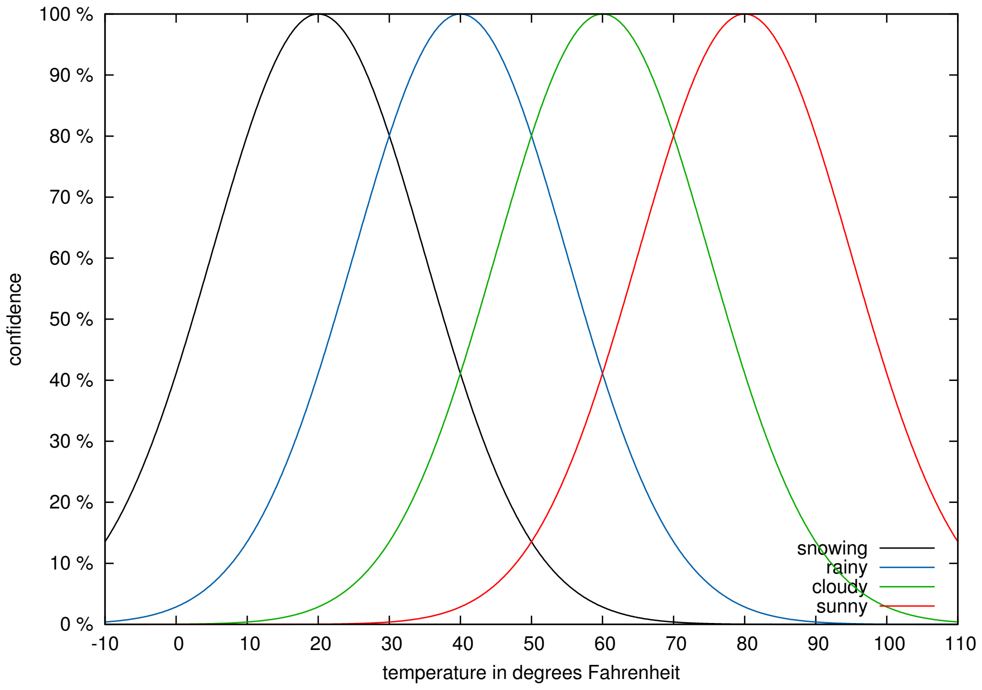


Figure 4.12: The confidence in a temperature measurement based on the observed condition

and the mapping functions φ

$$\varphi_1(value|\theta_1) = N_{ratio}(value|20)$$

$$\varphi_2(value|\theta_2) = N_{ratio}(value|40)$$

$$\varphi_3(value|\theta_3) = N_{ratio}(value|60)$$

$$\varphi_4(value|\theta_4) = N_{ratio}(value|80)$$

where $N_{ratio}(value|\mu)$ is the ratio of probability densities

$$N_{ratio}(value|\mu) = \frac{N(s|\mu, 15)}{N(\mu|\mu, 15)}$$

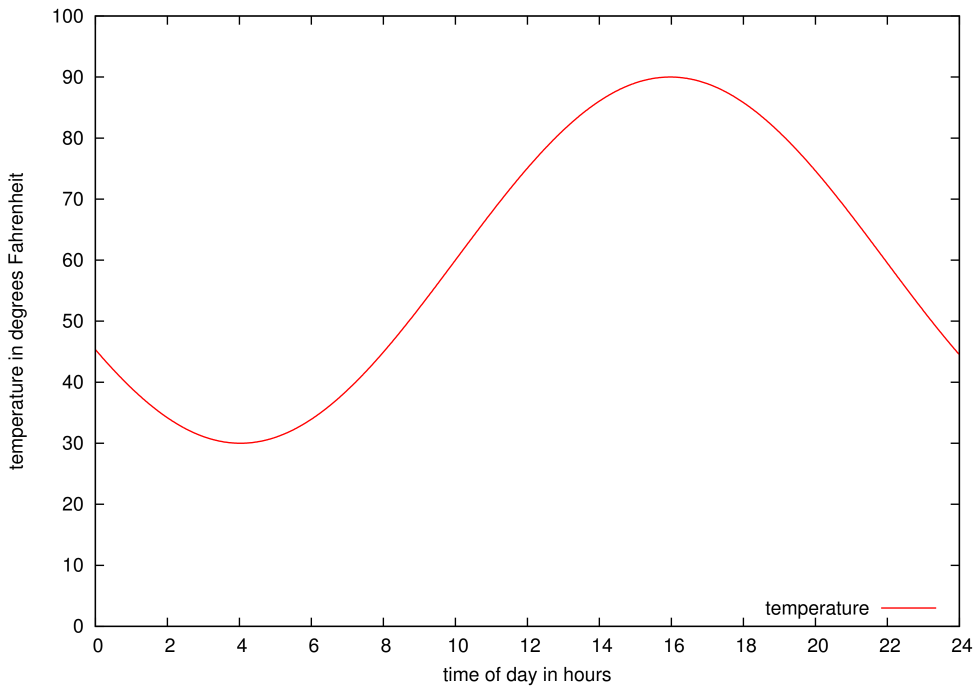


Figure 4.13: An example of a temperature cycle that can be used to model temperature confidence using a *dimension model*

of the Normal distribution defined as

$$N(x|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Continuous Value Context can be very useful in determining trustworthiness. Another example would be knowing something about the temperature cycle in a given region. The measurements of a sensor should roughly reflect a daily pattern such as the one shown in figure 4.13.

In particular, it should be cooler during the night than it is during the day. Hence we can compare actual temperature measurements against expected ones in order to determine confidence. Here, we would use a continuous *dimension model* with a context variable θ for the expected temperature during a particular time of day and define a continuous mapping function φ similar to

$$\varphi(value|\theta) = max\,(0\%, 100\% - 1\% \times \texttt{difference to expected temperature in degrees})$$

4.4 Graph Expressions

Given *elements*, *element nodes*, *relations* and *relation edges* in the abstract graph model described above, we can perform a variety of computations on them. We call these computations *graph expressions.* During the *knowledge extraction* phase, we do not have additional meta information which means that *graph expressions* can only perform a limited number of actions. However, they can be used to describe relationships between elements using comparison operations. For example, we are able to define a *graph expression* that limits the source and target to be sensor *elements* and checks whether both of the owner attributes are the same.

In the *knowledge processing* phase, we are able to include specific values, context, dimensions, relationships and external information which allows the *graph expressions* to become far more powerful. This means that we are able to specify ranges (e.g., temperatures, critical values, etc.) and incorporate time or location specific information (e.g., trends using time series analysis, no received heartbeat from sensor in 5 mins, sensor is within 1km of other sensors, etc.) as well as meta data (e.g., rankings, third-party assessments, etc.). Furthermore, utilizing *graph expressions* we are able to perform various aspects of data transformations such as conversion (e.g., smoothing, scaling), combination (e.g., aggregation) and filtering (e.g., outlier detection).

In order to maintain flexibility and reusability, we identify *graph expressions* by a unique name that can be referenced throughout the entire knowledge derivation process. A *metric* formally represents a computable value derived from the abstract graph model which consists of an *expression tree* and one or more graph references on which the *expression* is evaluated on. These references may refer to any of the graph components described above. Here, we discuss the parts that make up *graph expressions* and provide formal definitions for them.

4.4.1 Expressions

The most basic computational aspects within our framework can be captured using *expressions*. For example, determining whether two attribute values are the same is represented by the *equal expression*. These *expressions* include basic mathematical, logical, and comparison functionality that do not require context or additional information. In our framework, more complex *expressions* are defined as *model expressions*.

In order to achieve computational flexibility while allowing a formal definition of individual *metrics*, we define *expression trees* which describe the necessary computations.

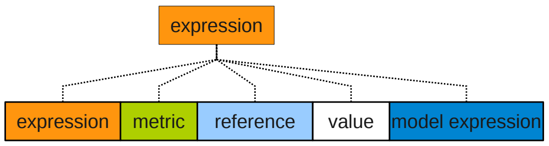


Figure 4.14: An *expression tree* node consisting of any number of optional child elements such as *expressions*, *metrics*, *references*, *values* and *model expressions*

As such, an *expression* may have any number of child elements (figure 4.14):

- other *expressions* representing a basic computation
- *metric* references to enable reusable computational definitions
- references to another graph components such as particular *element nodes* or lists thereof
- specific *values* which can be used for constants, scenario parameters, and critical values
- *model expression* references that enable the incorporation of more complex belief engines and trust assessments

Hence, we define an *expression* as follows.

Definition 4.23 Expression

Let m be a *metric*, ref a *reference* to a graph component, v a specific *value* and *mexp* a *model expression* then an *expression exp* is recursively defined as the collection

$$exp = \{\{exp_1 \dots exp_I\}, \{m_1 \dots m_J\},$$
$$\{ref_1 \dots ref_K\}, \{v_1 \dots v_n\}, \{mexp_1 \dots mexp_L\}\}$$

The key here is that by formalizing every *expression* we can reduce the number of ambiguous or biased interpretations of trustworthiness assessment approaches suggested in literature. Furthermore, this allows us to analytically evaluate the impact of parameter choices in trust models as part of our framework as we can adjust elements of the *expression tree* while keeping the rest unchanged.

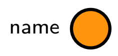
An *expression* consists of a particular tree structure where all child nodes are fully specified. As such, all tree nodes need to be fully resolved which means that they refer to specific *expressions*, *metrics*, *references*, *values* and *model expressions*. In particular,

- *expressions* refer to specific operations such as addition, summation, etc.
- referenced *metrics* consist of fully defined *expression trees*
- *references* are resolved to particular *element nodes* and *relation edges* or described as relative such as the source and target *element nodes* of a *relation edge*
- *model expressions* have all their parameters specified

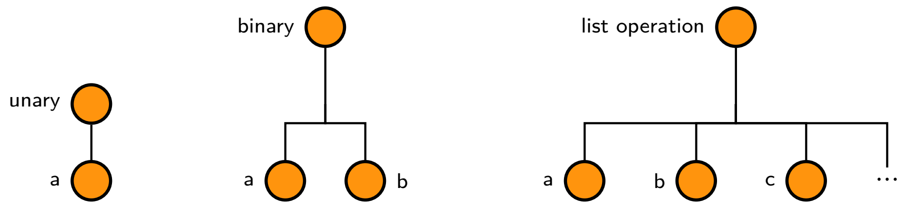
In order to describe *expression trees* we introduce the following graphical representation.

Expression 4.1 Expression

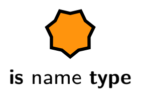
We represent an *expression* as[1]



where name is the type of expression (e.g., add, subtract, equal, etc.). Note that since there are a variety of operations with a different number of operands we choose the following convention



where *expressions* are operations which are performed on the particular child *expressions* (i.e., unary operation evaluates a, binary operation evaluates a and b, etc.). A special type of the basic *expression* is a constraint. We can use it to limit the types of *element nodes* and *relation edges* by their unique type name. For example, this can be used to restrict the application of an *expression* to only the sensor type. We note this special *expression* as



where name refers to a particular *element* or *relation* name.

4.4.2 Values

For every computation there may be values necessary that need to be incorporated. For instance, many formulas require constants, function factors, and probability distributions critical values. Furthermore, our framework facilitates the analysis and

[1]Note that *expressions* always have some graphical representation which we will not denote by a separate figure number. A list of the *expressions* is part of the table of contents.

simulation of a variety of scenarios. Specifically, in our approach *expressions* are able to include *system parameters* which are defined by simulation configurations (e.g., weights, *model expression* parameters) such that the same scenario can be evaluated in a variety of ways using a different parameters.

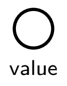
Definition 4.24 Value

A *value* is a term that can be used within an *expression tree.* There are two types of *values*, *constants* that do not change (e.g., factors, scales, mathematical constants) and *system parameters* that depend on the configuration of a particular scenario.

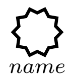
Within our graphical representation we express these concepts as follows.

Expression 4.2 Value

We represent a *constant value* as



where value is the term of the constant. On the other hand, *system parameters* are specified as



where name refers to a scenario configuration variable which actual value will be determined during a particular simulation run. Note that both of these *values* are usually leaf nodes in the *expression trees.*

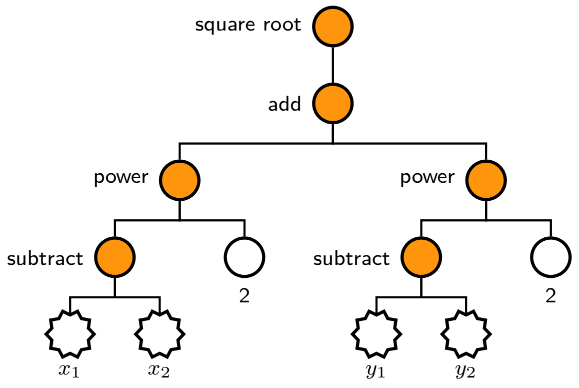
We are now at a point where we can define a variety of basic computations such as expressing formulas. Here, we briefly discuss how the Euclidean distance between two points in a two dimension coordinate system could be modeled using only *expressions* and *values.*

Expression 4.3 Euclidean distance

The Euclidean distance between two points is defined as

$$distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

where x_1 and y_1 are the coordinates for the first point and x_2 and y_2 for the second one. Since there are only basic computations involved the *expression tree* is a direct representation of the mathematical computations required for the solution.



4.4.3 References

In order to incorporate components of the graph model into *expressions trees* we need to specify what can be referenced and how. In particular, *element nodes* and *relation edges* contain attribute values that we need to be able to refer to because they form the value basis of our trustworthiness assessments. Furthermore, there are cases when we need to distinguish between what a *reference* applies to. For instance, an *expression* that yields related graph components of a particular *element node* could yield the connected *element node* or *relation edges.*

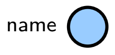
Definition 4.25 Reference

We define *reference* as an item that expresses the notion of a particular graph component. This component could be a particular attribute of an *element node* or *relation edge*. However, a *reference* can also express a specific subset or list of *element nodes* and *relation edges*. Furthermore, we utilize *reference* to distinguish between the source and target nodes of a *relation edge*. As such, since we apply an *expression tree* to graph components a *reference* represents the variable part that is different depending on the actual graph component.

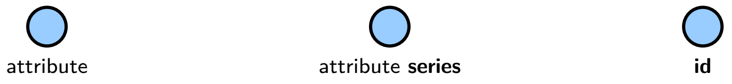
In terms of a graphical representation, we introduce the following.

Expression 4.4 Reference

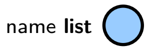
We represent a *reference* as



where name is the particular graph component referred to. For attributes of graph components we need to distinguish between several cases. First, we need to be able to reference the most recent attribute value. Second, for certain computations it is necessary to deal with the entire time series of attribute values. Third, in order to uniquely identify a particular graph component we have to be able to relate to its derived id attribute.



In order to express a subset of graph components we can use a list reference



where name could represent a subset or list of graph components. Furthermore, since *relation edges* consist of two *element nodes* we need to be able to distinguish between in *expression trees*. As discussed earlier, our approach is to refer to one as the source and the other as the target node.

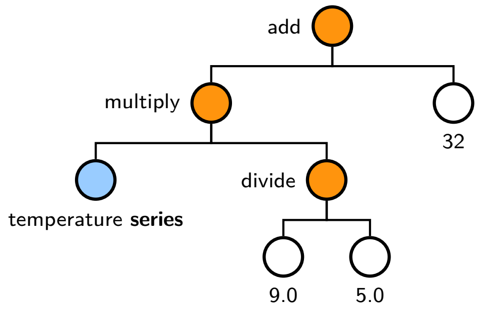
Together with the *expressions* and *values* discussed above we can formalize computations that depend on graph components and attributes thereof. As an example, we showcase the conversion of the temperature attribute for a particular sensor.

Expression 4.5 Temperature conversion

We can convert degrees Celcius to Fahrenheit using the following formula.

$$F = \left(C \times \frac{9}{5}\right) + 32$$

where C is the temperature in degrees Celcius to be converted. Applying this conversion formula to a particular sensor *element node* with a temperature attribute is then represented as



which would convert all values in the temperature attribute time series accordingly.

4.4.4 Model Expressions

In order to provide flexibility for implementing complex belief engines and decision processes our framework provides *model expressions.* These can be use within an *expression tree* to model complex algorithms and approaches that require parameters. In particular, without *model expressions* complex expressions would require specific implementations for each parameter value. This is clearly not feasible.

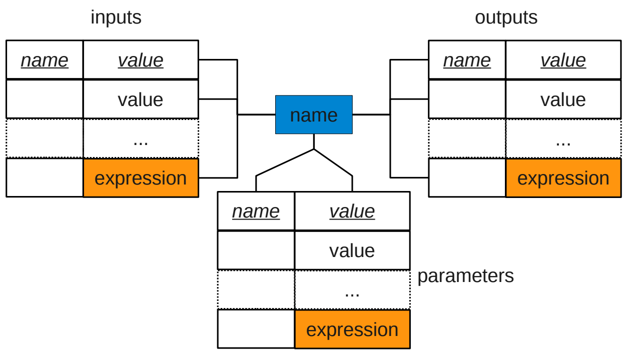


Figure 4.15: A *model expression* consisting of a unique name and sets of inputs and outputs as well as model parameters. Note that all values could be specified as either *value* or *expression*.

Definition 4.26 Model expression

Let $I = \{i_1 \ldots i_n\}$ be a set of inputs, $O = \{o_1 \ldots o_m\}$ a set of outputs, and $P = \{p_1 \ldots p_k\}$ a set of parameters where each i, o, and p could be a particular value or an *expression* then a *model expression* is defined as

$$mexp = \{name, I, O, P\}$$

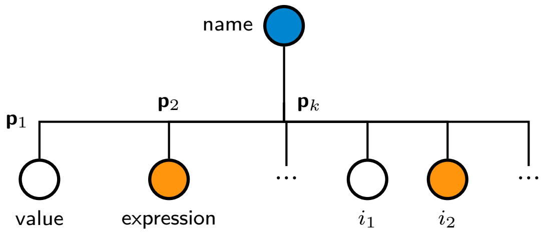
where name is a unique identifier such that the *model expression* can be properly referenced in *expression trees*. Here, we include all potential inputs and outputs in the sets. However, this does not mean that give a specific set of parameters all inputs are used and all outputs will be created by the *model expression*. The mapping from inputs to outputs is based on the parameters.

Note that the distinction between *expressions* and *model expressions* allows us to define flexible and reusable models that can be applied in various application scenarios. *Model expressions* can be graphically represented as follows.

Expression 4.6 Model expression

We represent a *model expression* as

where name is the unique identifier for a specific *model expression.* We follow the same convention that we use for the *expressions* such that child nodes of the *model expression* represent input. However, in order to specify parameters and differentiate them from inputs we label the edges accordingly. Hence, *model expressions* follow the format



where the edge labels $p_1 \dots p_k$ lead to parameters and the inputs $i_1 \dots i_n$ do not contain an edge label. Note that both can be specified as *values* or *expressions.*

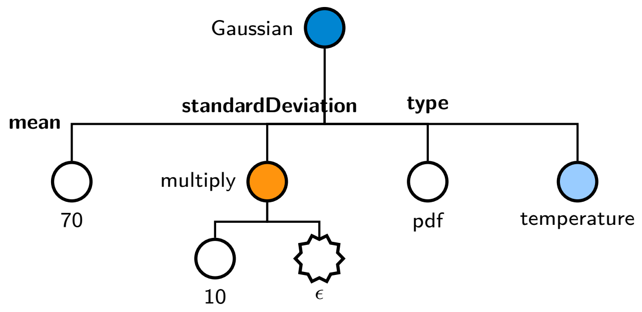
We can utilize *model expressions* to describe complex algorithms and approaches that require parameters. Here we show an example of incorporating a probability distribution into an *expression tree* to model the relative likelihood of a particular temperature value.

Expression 4.7 Temperature likelihood

Let the expected temperature follow a Normal distribution where the probability density is defined as

$$N(x|\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The *model expression* requires a set of parameters, in particular, mean, standard deviation, and type. Here, the type refers to the probability density function but others are possible (e.g., cumulative distribution function, sample, etc.).



Note that the *Gaussian model expression* follows the convention described above where parameters are marked with edge labels and the inputs are not. The ϵ shown in the *mean expression* represents the possibility of including *system parameters* in a variety of *graph expressions*.

4.4.5 Metrics

A *metric* represents a computable value that can be derived from the abstract graph model by evaluating an *expression* on a particular graph component. These *metrics* are referenced by a unique name and can be utilized in other *expressions* as discussed above.

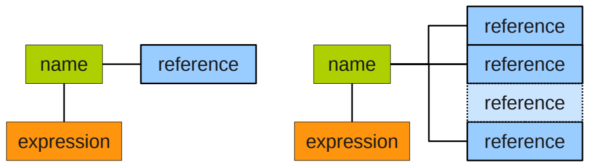


Figure 4.16: A *metric* consisting of a uniquely identifiable name, an *expression* and one or more *references*.

As shown in figure 4.16 *metrics* can be applied to single *element nodes* or *relation edges* as well as lists of them. It is important to note that specific *metrics* only exist if they are applicable. Therefore, when a *metric* is applied to a graph component it will only yield a result if all components of the *expression* can be evaluated on it. For instance, if we try to determine the average temperature over the last 24 hours but the sensor being evaluated has no measurements in the time period it would not result in a *metric*. An advantage of this approach is that the *metric* acts as both a filter (exists only if prerequisites are fulfilled) and processing instruction (compute a value based on the information referenced in the *expression*) at the same time thus simplifying query processing.

Using the flexible definitions of *expression trees* we are able to derive computable values from graph components which we refer to as *metrics*. It could be argued that the notion of *metrics* could be incorporated into *expressions*. However, the purpose of an *expression* is to define computational processes much like formulas whereas a *metric* embodies the application of *expressions* on various graph components.

Definition 4.27 Metric

Let ref be a *reference* to a graph component and *exp* an *expression* then a *metric* is defined as

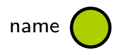
$$m = \{name, exp, \{ref_1, \ldots, ref_k\}\}$$

where *name* is the unique identifier of the *metrics*, *exp* the root of the *expression tree*, and $ref_1, \ldots, ref_k$ the referenced graph components the *expression* is applied to. As such, $m(ref_1, \ldots, ref_k) = \{exp(ref_1) \ldots exp(ref_k)\}$ is the application of the *metric* on the *references* performed by evaluating the *expression* on each individually.

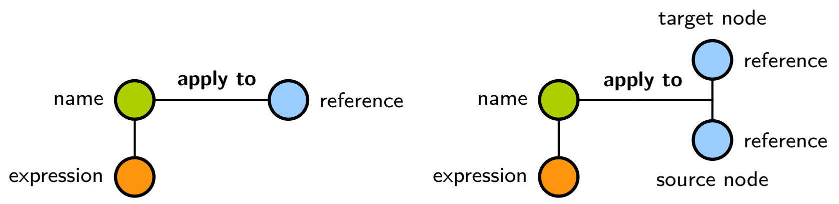
The graphical representation includes the *expression* and the *references* accordingly.

Expression 4.8 Metric

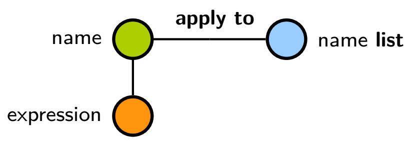
We represent a *metric* as



where name uniquely identifies the *metric* within our framework. There are several ways we represent the application of an *expression* to a particular graph component. The main difference is between the application of an *expression* on a *element node* and a *relation edge* where we need to specify source and target nodes.



Furthermore, *references* can be single graph components as well as lists of graph components which are represented as



where the name of the list needs to be specified (e.g., all sensor *element nodes*).

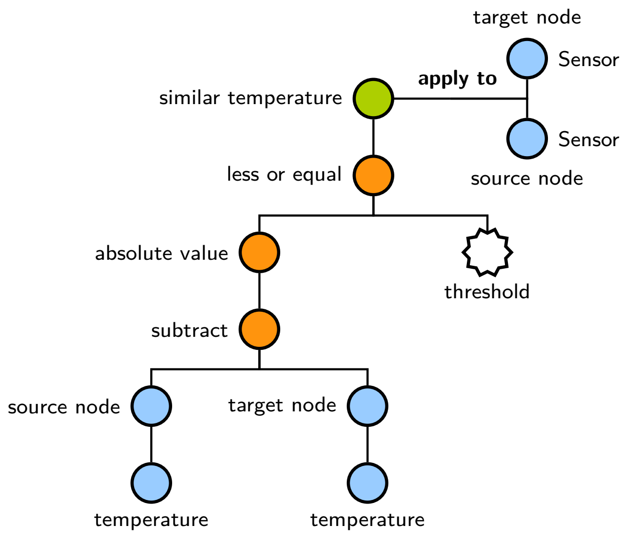
With all *graph expressions* formally defined we will show some examples of how *metrics* are fully specified and utilized within the framework.

Expression 4.9 Similar temperature metric

The process of determining whether two sensors have similar temperatures based on a threshold temperature difference can be defined as

$$\text{similar temperature} = \begin{cases} \texttt{true} & \text{if } abs(t_1 - t_2) \leq threshold \\ \texttt{false} & \text{otherwise} \end{cases}$$

where t_1 and t_2 are two temperatures accordingly. We model the *metric* using a combination of *expressions*, *values*, and *references* and apply it to a *relation edge* consisting of two *sensor element nodes* as

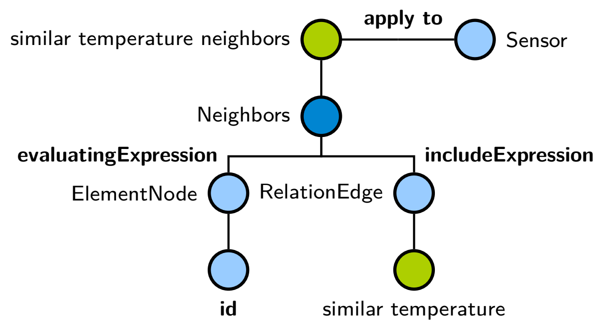


where the temperature difference threshold is modeled as a *system parameter*. This allows the *metric* to be as flexible and reusable as possible.

We can then utilize this *metric* in a variety of ways. For instance, given a particular sensor we can determine all related sensors based on whether or not they have a similar temperature.

Expression 4.10 Similar temperature neighbors metric

In order to determine related *element nodes* we can use the *neighbors model expression*. It takes two parameters. First, an *include expression* that filters existing relationships based on whether the specified *expression* yields `true`. Second, an *evaluating expression* which is applied to the remaining graph components that were not filtered out. Here, we express a *metric* that when applied to a *sensor element node* gives a list of ids for the related neighbor *element node* for which the *relation edge* determined a similar temperature.



4.5 Chapter Summary

In this chapter, we presented our approach to modeling heterogeneous data as well as belief engines and trustworthiness assessments using an abstract graph model called *Berlin*. Specifically, we discussed how to formalize data and express it in terms of flexible and extensible graph components. These graph components consist of *elements* and *relations* which provide the descriptions for data used in our framework. The actual data is then stored and processed in *element nodes* and *relation edges* that keep track of values in terms of time instances.

This leads to two different types of graphs. First, the *element description graph* contains *elements* and *relations* but no actual data. Second, the *element instance graph* contains data instances in the form of *element nodes* and *relation edges*. Furthermore, we discussed how we can utilize *graph expressions* to transform the graphs such that they model particular application scenarios better by specifying *elements* and *element nodes* of interest.

Attributes can be associated associated with *dimension models* that enable to modeling of “local” trust aspects such as accuracy and confidence. *Dimension models* may be probabilistic or deterministic representations of trustworthiness. They depend on the context within a dimension such as days within the time dimension or particular areas within the location dimension. Note that, while time and location variant *dimension models* are most prevalent, we show that *dimension models* are flexible enough to model other dimensions as well (e.g., ownership, set of weather conditions, etc.).

Implementations for knowledge processing that incorporate belief engines and knowledge evaluation through decision processes are formally modeled using *graph expressions.* These *expressions* are evaluated on graph components which forms the basis for the knowledge processing and knowledge evaluation phases of our framework. We use *metrics* to express computable values that apply specific *expressions* to a single or a set of graph components. These *expressions* are organized in a recursive tree hierarchy that may include other *expressions* (i.e., basic computations like add, subtract, and count), *model expressions* (i.e., complex computations that require parameters), *references* (i.e., to graph components), *values* (i.e., constants or *system parameters*, and *metrics* (i.e., named references to other *expression trees*).

5

The TrustKnowOne Framework for Incorporating Trust and Quality of Data into Knowledge Derivation

Most decision processes operate by evaluating available information and deriving knowledge or insight from it. While this seems straightforward, in reality we face a variety of problems which make knowledge derivation and decision making complex and difficult.

In particular, decisions are often influenced by factors such as past experience with similar or related problems. This causes decision processes to exhibit subjective rather than objective and reproducible behaviors. Furthermore, depending on the situation or context, given the same information different decisions could be made. Constraints also impact decision processes. Specifically, there are always resource constraints which

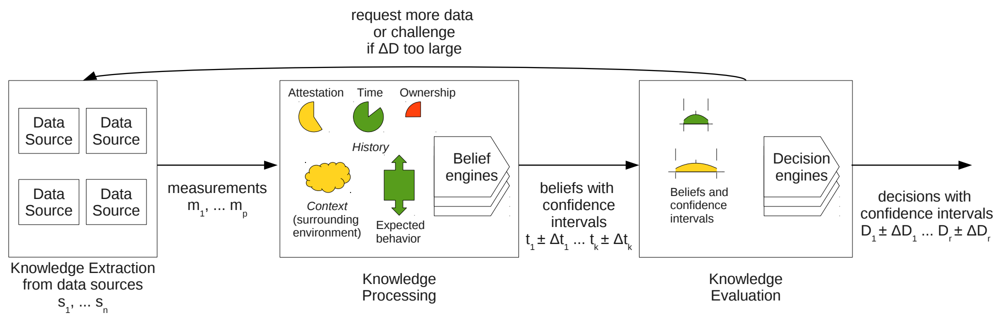

request more data or challenge if ΔD too large
Data Source
Data Source
Data Source
Data Source
Knowledge Extraction from data sources $s_1, \ldots s_n$
measurements $m_1, \ldots m_p$
Attestation
Time
Ownership
History
Context (surrounding environment)
Expected behavior
Belief engines
Knowledge Processing
beliefs with confidence intervals $t_1 \pm \Delta t_1 \ldots t_k \pm \Delta t_k$
Beliefs and confidence intervals
Decision engines
Knowledge Evaluation
decisions with confidence intervals $D_1 \pm \Delta D_1 \ldots D_r \pm \Delta D_r$

Figure 5.1: Knowledge Derivation framework

require trade-offs. A classic example of this is the need to balance time (i.e., speed, performance) and space complexity (i.e., storage space, memory, network bandwidth, etc.) found in almost all applications that require extensive computational power.

It is important to realize that decisions are based on the assumption that the data we are evaluating is useful and trustworthy. Data quality refers to properties such as accuracy, completeness, validity, and timeliness which are often assumed to be inherent [84, 117, 133, 175]. However, these assumptions may not be correct and as such could have a dramatic impact on decisions being made. Furthermore, pieces of data are often related or dependent on each other. We need to consider these dependencies, correlations, and trust relationships between data elements. Trustworthiness represents the perceived level of confidence we have that a particular data source is collaborative and behaves according to specification [31, 33, 54, 63, 88, 114, 122, 184]. The problem is that determining trustworthiness is hard for reasons such as dependencies between pieces of data, changes in data and resources over time, and resources potentially conspiring with each other [17, 57, 114].

The *TrustKnowOne* framework presented as part of this dissertation addresses these issues as will be described in this chapter. The main focus of our framework is to provide a formalization of approaches to quantify trust and data quality aspects throughout the knowledge derivation process and provide a variety of confidence and trustworthiness assessments for decisions. Figure 5.1 shows an overview of the main components of our layered framework.

5.1 Architectural Principles

In order to fully utilize available knowledge for making decisions, we present a layered architecture with models for various aspects of trust and quality of data. We will discuss how among others things, context, expected behavior, and relationships of data can be incorporated to improve knowledge derivation and to allow for better decisions to be made. In this section, we present an overview of the framework components and discuss the architectural design principles governing our framework.

At the core, we deal with three different entities that form the basis for decisions: data, data quality, and trust. Here the term data refers to the structured or unstructured information we gather from various sources. To achieve our goal of improving knowledge derivation several challenges must be addressed. Data may be incomplete or inaccurate, or even worse, someone might have intentionally altered it, i.e., attacked the data. For some of these problems there are solutions (e.g., digital signatures to prevent modification) while for others there are not (e.g., how do we know we received all data?). A further complication is the fact that decisions must usually be made in a finite amount of time which means we often need to make a decision before all data has been obtained. The point is that we need to account for these factors when we process and utilize data.

To overcome these challenges it is critical to determine the quality of data using auxiliary knowledge such as the information source, historical data, and location information. Hence, quality is not something that is absolute but rather relative, changing over time, and dependent on our knowledge of the context at a particular point in time. Furthermore, data quality and data trustworthiness are two distinct things. We may determine that the quality of data is high enough (e.g., based on evaluating the context and the dimension models), but it might come from an untrustworthy source.

Sometimes, low quality data that we can trust might prove to be more useful in our decision making process than perceived high quality data that may be tampered with some probability. A premise of the our approach is that the combination of the data itself, its perceived quality, and the trust we put into it and its source will allow us to make better decisions.

Decision making processes vary in complexity depending on the application scenario. The reason is that there are essentially two components, the approach on how a decision is made (decision engine) and the techniques used to process the data and its context on which the decisions are based (belief engines). Note that our approach makes this important distinction which is often ignored in literature. Furthermore, we enhance the overall decision making process by incorporating perceived quality and trust of data into these belief and decision engines. While this increases the complexity of the framework, it has the advantage of making the framework more flexible and useful. In particular, with the same data being available, we can employ different decision methods ranging from simple (e.g. voting, ranking) to the more complex ones (e.g. Bayesian inference [44, 69, 100, 126, 128, 158], Dempster-Shafer theory [59, 108, 147, 148, 155], weighting schemes [63, 108, 113, 146, 182]) to arrive at a decision. In addition, the separation allows us to evaluate multiple belief and decision engines in order to determine the best possible decision given all available knowledge. Based on this evaluation process we can improve on our decision making process in the future.

To address the complex problems described above we break down our framework into three layers or phases. Each of the phases represents a model of a specific set of tasks that need to be accomplished throughout the knowledge derivation process.

- The *knowledge extraction* phase models data sources and their integration
- The *knowledge processing* phase models data processes and incorporating quality and trust relationship aspects
- The *knowledge evaluation* phase models decision making processes

The focus of these models lies on the development of *measurable factors* that can be used to determine the effectiveness and performance of different techniques and approaches given certain application scenarios. These factors are especially important for evaluating how these models are affected by knowledge attacks that attempt to modify data, belief engines, and decisions processes.

Our approach to provide a formalization of the entire knowledge derivation process and incorporate data quality and trust aspects is based on the *abstract graph model Berlin* discussed in chapter 4. In particular, we model every piece of information as a *graph component* in order to allow for a flexible, standardized, modifiable, and interoperable data management foundation of the framework. Furthermore, data processing approaches, quality and trust assessments in terms of *belief engines*, and *decision processes* are also represented by *graph expressions*. As such, our *TrustKnowOne* framework provides a formal and flexible approach to knowledge derivation where each layer addresses specific aspects of the overall process.

Knowledge extraction models the task of formally describing how data is transformed from data source into *graph components* of the *abstract graph model*. In addition, it considers meta information and context in order to provide "local" quality and trust assessments. The resulting *abstract graph model* and these assessments serve as input to the *knowledge processing* component where we incorporate more complex quality and trust assessments that take into consideration context and relationships. Specifically, we evaluate "global" meta information that depends on aspects such as scenario specifics (e.g., temperature ranges, dangerous radiation levels) and trust relationships (e.g., sensors in the same location should have similar measurements). *Knowledge evaluation* has access to all knowledge modeled in the first two phases. The *knowledge evaluation* phase then models decision making as *decision processes* that are also represented by *graph expressions*.

Note that this separation of "local" and "global" information is important for several reasons. First, it clearly separates what type of context is incorporated in the *knowledge* *extraction* and *knowledge processing* components. Second, the process of transforming data from data sources into *graph components* needs to be very specific due to constraints (i.e., type, format). By incorporating "local" assessment into this transformation we are able to derive meta information in a more meaningful manner. Third, complex quality and trust assessments should not depend on how data is incorporated into application scenarios. In particular, our framework provides an *abstract graph model* as the basis on which these assessments are derived in terms of *belief engines*. This avoids implementing techniques that depend on subjective interpretations of the raw data. Finally, while there exist various quality and trust assessment approaches it is often difficult to improve them or apply them to a different application scenario. In our framework, approaches modeled as *belief engines* are represented by formal *graph expression* definitions. Because they are all evaluated on the formal *abstract graph model* provided by *knowledge extraction* component it makes it easy to modify, exchange, and reuse them in a various scenarios and not just the application scenario they were originally defined for.

An important benefit of the *TrustKnowOne* framework is the separation of the process of performing quality and trust assessments from the process of making decisions. Current trust assessment approaches often combine these two which makes their evaluation, comparison, and improvement difficult. Furthermore, our framework allows for fast prototyping of new approaches as well as evaluating them against existing approaches because of the formalized nature of the *abstract graph model*. Another problem seen in existing literature results from the coupling of assessment approaches with decision processes. By merging the two, you become tied to particular approaches which may result in a good assessment technique being paired with sub-optimal decision making or vice versa. Instead, our framework decouples *belief engines* from *decision processes*. This allows for the evaluation of various combinations of assessment and decision making approaches as well as a more detailed analysis to find the optimal pair resulting in an overall better knowledge derivation processes.

In general, our framework derives multiple decision options for every scenario. These decisions include representations of confidence and trustworthiness assessments. Specifically, the framework provides an assessment of how confident it is in a particular decision given the knowledge derived from data as well as "local" and "global" assessments of the knowledge utilized in arriving at the decision. Furthermore, based on user defined thresholds (e.g., in case confidence in a decision it deemed to low) the *knowledge evaluation* component can either try to incorporate additional data by requesting it from *knowledge extraction* or attempt to improve existing assessments by reevaluating knowledge and context.

An additional benefit of the *TrustKnowOne* framework lies in its ability to analyze and evaluate knowledge attack scenarios. Because our framework incorporates trust and quality of data as well as formalizes knowledge derivation, we are able to assess the impact of attacks on data and meta information. For instance, malicious nodes in sensor networks could provide incorrect measurements (i.e., attacks data) and collaborate in an attempt to give other malicious nodes higher trustworthiness (i.e., attacks meta information). We can evaluate the robustness of trust assessment approaches and decision processes using several techniques. First, we can simulate scenarios with increasing levels of particular attack activity (e.g., percentage of data compromised) and compare the assessment results of the trust approaches to a baseline. Second, since attacks materialize themselves in terms of changes in data and context we are able to define *graph expressions* capable of determining the existence of these changes.

In the following sections, we discuss in detail the three components of the framework.

5.2 Knowledge Extraction

The first of the three stages of our *TrustKnowOne* framework is *knowledge extraction.* While this phase is the most straightforward one should not overlook its importance. Data is often captured in order to be utilized as a basis for decision processes. However, when dealing with data we often face problems such as what kind of data is available or what format is my original data in and does it need to be converted or transformed? These questions can often be trivially answered if the amount of the data and consequently the amount of knowledge derived from that data is small. However, the need for a framework like the one developed here arises when we have to deal with large and complex data as is the case in the radiation detection scenario discussed in section 6.2.

In order to address issues with managing dynamic heterogeneous data, we developed a framework that is capable of dealing with the various aspects of data modeling and in particular knowledge extraction from data. Our framework provides a common abstract data model based on graph theory with *nodes* representing elements of the data model and *edges* the relationships between them as described in chapter 4. Hence, this graphical data representation is able to store information and allows for *knowledge extraction* through the definition of patterns that can be matched onto the data graph.

As a first step in the overall knowledge derivation process, we need to clearly define how data and meta information about the data can be incorporated into our framework. For this purpose the *knowledge extraction* component utilizes *data adapters* which are responsible for extracting knowledge from data sources as well as providing the framework with the information necessary to assess its trustworthiness and quality. It is important to note that at the *knowledge extraction* stage only "local" meta information is available. In particular, aspects such as context and expected behavior used to assess quality and trustworthiness are limited to considering meta information about individual data elements and sources but not their relationships to each other. This clearly distinguishes the *knowledge extraction* stage from the *knowledge processing* stage.

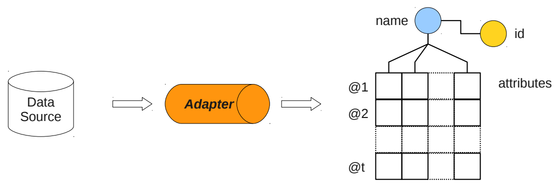
The main purpose *knowledge extraction* is the extraction of data elements from the data source and their transformation into equivalent *graph components*. This component also performs "local" assessments by incorporating "local" context and expected behavior through the use of *dimension models* and "local" *belief engines*. The input to the *knowledge extraction* component consists of information about the raw data as well as its data sources. The result of the *knowledge extraction* phase is the data modeled as *graph components* as well as “local” quality and trust assessments of data and their sources.

To keep *knowledge extraction* flexible and abstract we want to be able to incorporate the implementation of a variety of algorithms and techniques. Hence, we discuss the following aspects of the *knowledge extraction* component in detail throughout this section.

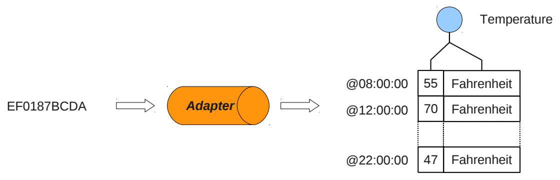
- Providing a formal description of data elements, data sources and their mapping onto *graph components*
- Formally describing “local” quality and trust assessments in terms of *dimension models* and *belief engines*
- Maintaining flexibility for a variety of data acquisition approaches
- Providing the ability of incorporate structured and unstructured data into the knowledge derivation process
- Incorporating dynamic context such as time and location into trust assessments
- Including data lineage by keeping track of how knowledge was derived from a particular data source as well as how it is processed
- Determining the cost associated with data acquisition and transformation thereby enabling evaluation and comparison

5.2.1 Knowledge Extraction: Architecture

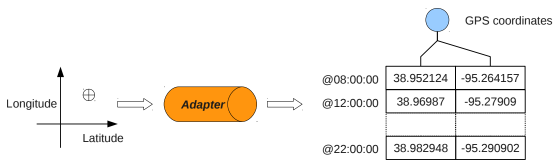
Approaches such as [51, 62, 135] describe data and relations between data elements. However, as discussed in chapter 3, they all lack the comprehensiveness to include various aspects of knowledge derivation necessary such as trust relationships, local value models, as well as time and location dynamics. Furthermore, one of the goals of our



(a) Generic *knowledge extraction* from data using an adapter



(b) Temperature from binary data source example



(c) GPS coordinates from location example

Figure 5.2: Data element definition from data source via adapter

approach is to enable the combination of data with meta information such as context, expected behavior, the process of how it was obtained, and security features (e.g, certificates, signatures) to further improve quality and trust assessments. Hence, we provide a formal approach that enables the modeling of data and knowledge derivation in a flexible and extensible way.

A key aspect of our approach is maintaining flexibility by enabling the addition of new data and data formats. Therefore, we introduce a *data element* that represents the notion of a basic piece of data and its context. This *data element* may have any number of attributes. In order to become part of the data model, we define *adapters* that capture the particular data and provide a common abstract view of it in terms of *graph components* (see chapter 4). This part of the process is shown in figure 5.2.

As part of the *knowledge extraction* component we need to define the following.

Data Elements The type of information which is being used in our framework needs to be formally defined in terms of *graph components*. This means that data will be described by a combination of *elements* and *relations* each with their respective attributes. In addition, meta information such as "local" context, known expected behavior, and known value constraints is incorporated in *dimension models* and associated with *graph components*.

Data Sources We treat data sources as providers of information in its most basic form. As such, they may provide values, "local" relationship information, as well as context for *graph components*. Since there exist a multitude of data formats it is important to note that our framework does not provide implementations for each one of them. Instead, our framework designates this task to be addressed by scenario specific *adapters*. However, the key point to keep in mind here is that the main task of the *knowledge extraction* component is to provide a unified and formalized representation of information relevant to application scenarios in terms of the *abstract graph model* discussed in chapter 4.

Knowledge Extraction Mapping With both descriptions available (data and data source), the *knowledge extraction* phase comes down to establishing mappings from the data source to the respective *graph components* in our *abstract graph model*. This task is performed by *adapters* whose implementation can range from simple mappings of values to more complex transformations and extraction approaches. Note that our framework provides flexibility in this regard as extracted values can be transformed as part of the *knowledge extraction* phase or by utilizing *belief engines* during the *knowledge processing* phase which we will discuss later.

Consider the following example where temperature measurements are captured by a set of sensors in a custom binary format. Let us assume we are only interested in analyzing historical information for temperature trends. Hence, all time series temperature measurements are provided to us in a single file. First, we describe a new *graph component* that contains time series information of temperatures. Seconds, we model the format of the data stored in the file. Third, the transformational mapping is implemented as the *adapter* which is defined specifically for this binary format and provides an abstract view of the temperature values that were captured.

Many real work scenarios require an online (dynamic) as opposed to offline (static) knowledge derivation process because of changes in data and relationships that need to be incorporated in real time. As such, our framework and in particular the *abstract graph model* supports the dynamic definition and extension of *graph components* as well as adding, modifying, and removing data modeled as *element nodes* and *relation edges*. By providing this dynamic graph model, we are able to address a variety of application scenarios in which data and relationship structures are constantly changing as will be shown in chapter 6. The advantage of our framework is that after this initial *knowledge extraction* phase we are able to utilize data in a flexible and common manner within our *abstract graph model*.

The second key task of the *knowledge extraction* component deals with performing "local" quality and trustworthiness assessments. As the process of extracting knowledge is only an early step in knowledge derivation, we only have limited information available. While specific context such as possible temperature value ranges for sensors are available, more complex relationships such as comparing sensors based on temperature similarity requires additional "global" context. The main approach in this phase is to assess data on a "local" level with no complex "global" context such as similarity ranges, distribution parameters, and scenario specific meta information.

In our framework, there are two options for performing these "local" assessments. First, we can associate *graph components* with *dimension models* that provide context specific to the type, origin, and value range of data. Second, basic *belief engines* which are discussed in more detail in the *knowledge processing* phase can be used to provide aspects such as expected behavior assessments (based on analyzing time series information) and comparative information (e.g. same sensor location, lower temperature). Note that everything that requires some sort of parameter requires “global” context and thus needs to be part of the *knowledge processing* component.

5.2.2 Knowledge Extraction: Data Acquisition

Our framework supports a variety of data acquisition models. Here, data acquisition refers to the process of making data available in the form of data sources. There are two major types to consider. First, the most common knowledge derivation process requires only a *static* data source where data and meta information made available does not change. This makes *knowledge extraction* a one-time process. Second, in the case where data changes within a data source, the acquisition and thus *knowledge extraction* needs to be *dynamic*. Specifically, whenever new data becomes available, we allow it to be incorporated into the *abstract graph model* (push approach). Furthermore, there are cases, such as low confidence in trustworthiness assessments or decisions, where we may require additional information to be acquired by the data source (e.g., sensor, monitoring process) (pull approach). These are important factors in making the framework presented here flexible and extensible.

Note that a specific use case for dynamic data acquisition involves multi-agent systems that perform work independently of each other. This allows the framework to be utilized in application scenarios such as mobile applications, sensor networks, and intrusion detection systems which are discussed in detail in chapter 6.

5.2.3 Knowledge Extraction: Data Integration

The *knowledge extraction* component needs to be able to deal with structured and unstructured data. While structured data may be mapped into *graph components* more efficiently, our framework provide the means to effectively incorporate unstructured data as well. In particular, we acknowledge the fact that data may be incomplete and incorrect. For instance, data collected by sensor networks is often clearly defined. However, free text such as reviews of mobile applications (as discussed in section 6.1), travel experiences, and medical information often do not adhere to a specific structure but rather consist of a subset of a large corpus of possible terms (i.e., look and feel, performance, permissions; hotels, flights, food; medications, symptoms, diagnoses). Having to completely define all these terms is impractical.

As such, our *abstract graph model* supports "non-descriptive" attributes to be dynamically added to *graph components* (chapter 4). This aspect makes our approach flexible and extensible as will be demonstrated in various scenarios (chapter 6). Specifically, it allows the framework to model a data warehouse approach where "raw" unstructured data can be stored using a limited set of attributes and features can be derived from them as part of belief engines during the *knowledge processing* stage. It should be noted that for standardization and compatibility reasons, the focus should remain on formalizing as much of the data and data sources as possible.

Since our *abstract graph model* approach provides a flexible solution to managing heterogeneous data, *TrustKnowOne* is able to overcome issues that arise from data integration and information fusion. For instance, we can model various sensor types with different attributes without having to choose between modeling only common or all possible attributes (see chapter 3). Furthermore, information fusion is performed by formalized belief engines during the *knowledge processing* phase. This allows for a better approach as meta information such as context and "local" relationships can be incorporated into the knowledge derivation process.

5.2.4 Knowledge Extraction: Time and Location Dynamic Data

One aspect that is considered secondary in many data processing frameworks (chapter 3) is the fact that data and context is often dependent on some dimension such

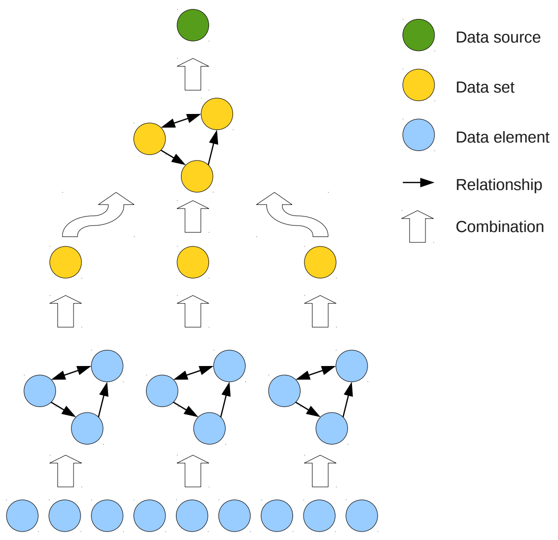


Figure 5.3: Data model organizational hierarchy overview with different layers of abstraction

as time or location. Take for instance the case where a temperature sensor's accuracy slowly degrades over time. Furthermore, the possible range of temperature values may be impacted by the location of the sensor. Our framework enables the formalization of *dimension models* that can be associated with *graph components* to model the impact of dimensions such as time and location. This provides a more realistic approach to evaluating data and assessing its quality and trustworthiness.

5.2.5 Knowledge Extraction: Data Lineage

We can group and organize various *data elements* into *data sets* by using *tags*. The term *tag* here loosely refers to any type of grouping. This may be ordering *data elements* by location or time but it could also be used to create logical groupings such as correlations or dependencies.

Figure 5.3 shows that *data elements* can form relationships at various degrees of abstraction. Specifically, *data elements* can be associated with particular *data sets* which are provided by *data sources*. For example, a set of sensors provides temperature measurements. These measurements can be grouped into location data sets by some weather monitoring authority. Note that in general, data will be organized hierarchically as shown in Figure 5.3 because this allows clear levels of abstraction. However, since the relationship between pieces of data can be quite complex, knowledge can also be derived by combining data in non-hierarchical fashion.

All of this means that, providing data lineage is often a complex process. Nevertheless, our approach incorporates lineage throughout the knowledge derivation process by associating data with the context of where it originated and how it has been processed. As part of the *knowledge extraction* stage, data is automatically tagged by adding an additional source attribute to each *graph component*. Because *knowledge processing* as well as *knowledge evaluation* is formalized using *graph expressions* that are evaluated on the *graph components* provided by the *knowledge extraction* component, our framework enables the tracing of data lineage and processing.

5.2.6 Knowledge Extraction: Cost Assessments

One of the focus areas of *TrustKnowOne* is the trustworthiness assessment of data sources. For this purpose, our framework maintains meta information about these sources throughout the knowledge derivation process. First, we can associate data sources with certain data acquisition costs. Here, cost reflects aspects such as timeliness, completeness, and accuracy. Second, we need to incorporate the cost of the process of adapting data from data sources into equivalent *graph components* performed by the *adapters*. Third, while the process of "local" assessments is usually performed in parallel with the transformation of data into *graph components* we need to account for it. Thus, the total knowledge acquisition cost is thus a sum of the raw data acquisition costs, the necessary transformations into *graph components*, and performing "local" assessments.

$$cost_{knowledgeextraction} = cost_{acquisition} + cost_{transformation} + cost^{local}_{assessment}$$

Incorporating these cost factors into the knowledge derivation process has several advantages. First, one may have a variety of options available to acquire the necessary knowledge in order to come up with a decision for particular scenarios. Our framework provides a formal mechanism for determining the cost of each of these options and as such enables evaluation, optimization, and comparison. Second, sometimes it is necessary for decisions to be made even with incomplete data (i.e., not all data being available) due to time or resource constraints. By providing a cost context for data and data sources we can incorporate data based on the best value or highest quality. Third, in dynamic scenarios where over time more data is incorporated into the knowledge derivation process, it is important to evaluate cost factors as well. Specifically, our framework (i.e., *knowledge evaluation* component) allows for additional data collection or reevaluation of existing data if certain confidence criteria are not met (figure 5.1). This reinforcement learning [83, 163] approach of *exploration* and *exploitation* is inherently cost-based since the decision of which steps to take depends on the ratio of their expected return compared to the costs.

The output of the *knowledge extraction* phase is the data modeled as *graph components* as well as "local" quality and trust assessments of data and their sources. This becomes the input to the *knowledge processing* phase.

5.3 Knowledge Processing

When processing information we often encounter questions such as

- Are certain data pieces correlated?

- How can one derive knowledge from individual pieces of data?
- Does the combination of multiple pieces of data lead to more knowledge?

In our framework, the *knowledge processing* component provides way of answering these questions by formalizing data processing as well as quality and trustworthiness assessments. In particular, we use various techniques to transform data modeled as *graph components* and determine aspects such as perceived data quality and trustworthiness. The *abstract graph model* discussed in chapter 4 provides a flexible and extensible approach to describing algorithms and techniques for *knowledge processing.*

The *knowledge processing* component provides the ability for processing *graph components* using *metrics* represented by *graph expressions.* Furthermore, in this stage, we incorporate "global" meta information such as expected behavior, history, and other context to derive additional quality and trust assessments. The knowledge and "local" assessments derived from *knowledge extraction* are taken to the *knowledge processing* component. The result of the *knowledge processing* phase is that the graph model is now augmented with the "processed knowledge", such as the results of transformations and the evaluation of *graph expressions*, as well as additional quality and trust assessments based on "global" meta information, context, and relationships.

In this section, we discuss the following aspects of the *knowledge processing* component. Note that these will also be demonstrated in detail in throughout the scenario analysis (chapter 6).

- Providing a formal description of data processing techniques in terms of *graph expressions*
- Describing "global" quality and trust assessment approaches formally using *belief engines*
- Incorporating knowledge from sources with different trust aspects
- Assessing trust aspects from "global" context and relationships

- Maintaining flexibility in terms of how to perform processing and assessment computations
- Including processing lineage through the formal definition of processing and assessments as *graph expressions*
- Determining costs for processing and assessment approaches thereby enabling evaluation and comparison

5.3.1 Knowledge Processing: Architecture

The second stage of the *TrustKnowOne* framework is *knowledge processing.* There is often the need to transform data in order to derive the knowledge we seek. For instance, individual sales transactions are grouped by product, time, or location which allows strategic business decisions to be made on a higher level of abstraction. Our framework enables the use of a wide array of processing approaches through the evaluation of *graph expressions* on *graph components.* As such, simple mathematical approaches can be incorporated in the same manner that more complex ones can. Furthermore, our *abstract graph model* is flexible enough to support a large number of existing techniques for a variety of application scenarios such as sensor networks and intrusion detection systems as well as future ones because of its extensible *graph expressions* approach.

By modeling data and how it is processed as *graph components* and *graph expressions*, we provide a unified view of knowledge derivation. The advantage here is that instead of having one approach for data management and another one for processing, our framework enables the use of a single paradigm, our *abstract graph model.* A detailed description of how *knowledge processing* can be performed using our *abstract graph model* is discussed in chapter 4.

In addition to providing an effective way to model the processing of data, we enable the derivation of complex "global" quality and trust assessments. We can often associate meta information with data elements. The framework is able to correlate information by using meta information (see [124, 147]). Specifically, we are interested in spatial and time series data as well as information about the process by which the data was obtained (expected behavior and context) and security features such as certificates and signatures. One of the major tasks for the *TrustKnowOne knowledge processing* stage is to evaluate data modeled by the *knowledge extraction* component and combine it with additional meta information. In particular, we incorporate expected behavior [145] such as range, mean, and variance by associating a distinct probabilistic or deterministic *dimension model* with each data element as described in chapter 4.

It has been noted [12, 72, 97] that context awareness such as the semantic meaning and distribution of data values can often enhance knowledge derivation. Hence, our framework provides the possibility to evaluate surrounding data elements on temporal (similar changes over time, sliding windows [11, 27]) and spatial (co-located measurement entities) as well as physical (same data source, dependencies) and logical (ownership, groupings, signatures, vouchers) levels. Similar approaches discussed in literature include local structure inference [4, 100] and Markov blankets [130]. Our framework accommodates these techniques which can be implemented as *graph expressions* which enable their reuse, modification, and extension of them in a formal manner. Furthermore, since trust assessment techniques such as Bayesian inference [44, 69, 100, 126, 128, 142, 158], Dempster-Shafer theory [59, 108, 147, 148, 155] and weighting schemes [63, 108, 113, 146, 182] can be described using graph theoretic constructs, we are able to map them directly onto our *abstract graph model.*

Given these approaches, the combination of data, meta information, and trust assessments enables us to derive confidence levels for individual data elements that describe attributes such as data quality, accuracy, and trustworthiness. Our framework establishes formalized *belief engines* to assess quality and trustworthiness aspects of data and data sources that can be included in the *knowledge processing* of data. For example, data below certain quality or accuracy thresholds could be ignored during processing. Similarly, relationships among data elements (e.g., overlapping and conflicting data) can be used to determine trust aspects.

5.3.2 Knowledge Processing: Trust Aspects in Data Fusion

Maintaining trust aspects during the combination of heterogeneous data, often referred to as data fusion, is a complex problem on its own. In order to derive meaningful knowledge we not only need to consider the combination of data elements but also incorporate any trust aspects associated with them. In particular, Dalvi and Suciu [36] discusses the problems related to data integration while also having to address trust issues. First, integrating heterogeneous data with different trust levels means dealing with potentially conflicting data and trust assessments. Second, trust approaches need to be flexible enough to allows for future types of data where trust may not have been clearly defined. We address both of these problems with our *TrustKnowOne* framework through the use of *graph expressions*.

The flexibility of implementing trust approaches as *graph expressions* enables the incorporation of a variety of approaches mentioned in literature such as trust level fusion [114] and confidence levels of trust [141]. Note that, while there have been several approaches [108, 113, 148] directly focused on assessing quality and trustworthiness, the combination of homogeneous and heterogeneous data with trust aspects remains problematic. In our approach, processing approaches have direct access to trust assessments since they part of the *abstract graph model* and vice versa. This allows for trust aspects to be incorporated in a way that is not possible in other frameworks. For instance, one could weight data differently when performing data fusion based on its assessed trustworthiness.

We can incorporate existing as well as future approaches to trust assessments because of the flexibility of *graph expressions*. This allows us to perform fact finding in “safe” environments where data sources are cooperative and data is of high quality and trustworthiness as well as in "dangerous" environments where data sources have diverging interests and provide incomplete and conflicting data. Data processing frameworks found in literature can often only handle the first scenario since they do not incorporate trust with data fusion. What distinguishes our *TrustKnowOne* framework is that it can also handle the "dangerous" environments. Some of the application scenarios that our framework is able to address are discussed in chapter 6.

5.3.3 Knowledge Processing: Trust Relationships Between Data

While assessing quality and trust aspects "locally" can be difficult, considering relationships between data elements is even more complicated due to interdependencies and growing complexity. However, [10, 79, 156] make use of a social network approach to determine trusted resources based on types of relationships and frequency of interaction. This provides a good basis for our framework on ways to adjust trust levels and confidence intervals.

In particular, we formulate the idea to establish "checks and balances" between the data elements on a global level that evaluate their relationships. One goal of the *knowledge processing* component is then to model relationships between data elements including "global" meta information while incorporating a variety of trust models. In particular, *graph expressions* define how trust *metrics* are processed into data quality and trust assessments. Because of our *abstract graph model* approach to managing data, other potential data dependencies (causality) [15, 44, 125, 127, 128, 151, 158–160]) and correlations [10, 78, 79, 128, 130, 156] that provide additional knowledge can be explored.

In addition, clustering often provides insight to the relationships between seemingly disparate data elements. Given the flexibility in transforming *element instance graphs* using transformations as discussed in section 4.2.3, we can employ a variety of clustering [181] and biclustering [106] algorithms to discover these relationships. Note that we are especially interested in temporal and spatial ones [28, 34, 73, 138] as will be used in section 6.2. Another technique that we are able to directly map is pattern matching [78]. In particular, *graph expressions* can be used to incorporate statistical (i.e., certain amount of features match), syntactical (i.e., structural, hierarchical based) and template matching (i.e., assign pattern to closest template that matches) approaches.

5.3.4 Knowledge Processing: Computational Aspects

Note that our framework in contrast to others does not dictate how processing and assessment is performed (see chapter 3). We discuss our reference implementation in chapter 7. Instead, the approaches are modeled as *graph expressions* that are evaluated on *graph components* representing data. As such, our framework can work in a distributed way based on the individual and parallel evaluation of *graph expressions* hence overcoming limitations often seen in centralized systems. In the case where *graph expressions* include relationships and interdependencies our *abstract graph model* basis allows for various forms of clustering to be performed in terms of graph transformations (see section 4.2.3) to provide clear definitions of computational boundaries.

Furthermore, dynamic application scenarios often require the ability to partially reprocess and reassess knowledge. Since *graph expressions* provide formal computational models they can be evaluated on any range of *graph components*. Therefore, in dynamic application scenarios when new data is added or existing data is updated we only have to reevaluate the *graph expressions* that are impacted. Note that this can be accomplished without having to modify the overall knowledge derivation process. The distributed aspect of our framework also lends itself to the modeling of heterogeneous multi-agent systems such as radiation (section 6.2) and intrusion detection (section 6.3).

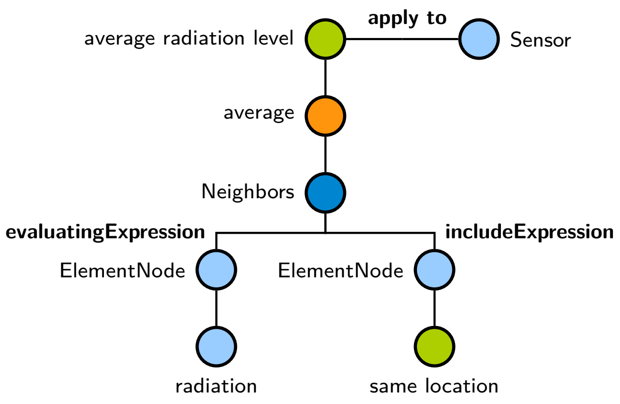
5.3.5 Knowledge Processing: Processing Lineage

As discussed in the *knowledge extraction* phase, we associate meta information with data elements. This includes lineage information on how it was extracted (i.e., data source, time, "local" context) and how it was transformed into *graph components*. In the *knowledge processing* component, we can attach additional lineage information that describes how data is processed. Since processing approaches as well as quality and trust assessment which provides "global" context are implemented as *graph expressions*, they are formally defined and allow tracing of lineage.

As an example, consider the calculation of the average radiation level in a region using several sensors with the following *metric*.

Expression 5.1 Average radiation level metric

In order to determine related *element nodes* we can use the *neighbors model expression*. Here, we filter sensors based on a *same location metric* (*include expression*) and retrieve the *radiation* values of the sensors (*evaluating expression*). Finally, we average the resulting list of radiation values using a *math expression*.



Let us assume we apply this *metric* to a list of 10 sensors in the same location where 5 sensors were provided radiation values from data source A, 3 from data source B, and 2 from data source C. We can then associate the resulting average radiation level with the appropriate lineage context in absolute (i.e., number of values incorporated from particular data sources) as well as relative terms (i.e., percentage of impact of specific data sources). The lineage meta information incorporated during the *knowledge extraction* phase is automatically propagated to the result of the *metric*. The combination of this with the data source context allows us to weight meta information accordingly.

In addition, we are able to attach context about processing to the result of *graph expressions*. This includes the *metric* that was used to produce the result as well as the *graph components* on which the *metric* was evaluated. Note that, all processing approaches and *belief engines* implemented as *graph expressions* can be annotated with this type of lineage context. Therefore, our framework improves knowledge derivation by providing formal definitions of *knowledge processing* techniques which allow for extensive tracing of data lineage which results in better informed and more realistic decision making.

5.3.6 Knowledge Processing: Cost Assessments

Determining cost aspects of *knowledge processing* is based on assessing the cost of performing *graph expression* evaluations on *graph components*. Since these *graph expressions* represent both processing and assessment approaches they provide a unified cost modeling technique based on graph theory (see [15, 19, 127, 128, 157] graph metrics). As such, the cost of *knowledge processing* can be expressed as the sum of the processing and assessment costs.

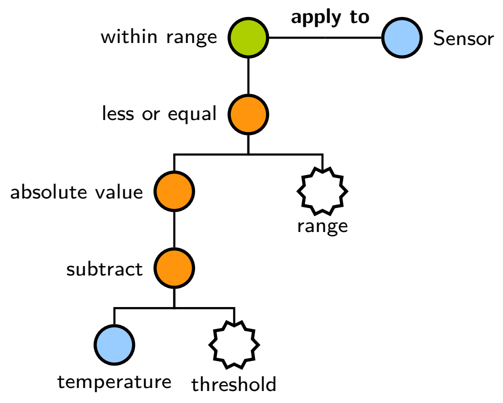
$$cost_{knowledgeprocessing} = cost_{processing} + cost^{global}_{assessment}$$

Our *TrustKnowOne* framework enables *graph components* to be associated with meta information and context. For the *abstract graph model* we can use a similar approach when determining the cost of *graph expressions*. In particular, every expression within the *expression tree* can be annotated with a cost factor. Note that this cost factor can be either static (i.e., fixed cost) or dynamic (i.e., based on number inputs and parameters) depending on the particular *graph expresssion* used. While static cost factors make overall cost estimation straightforward, dealing with dynamic cost factors, especially for *model expressions*, is more difficult. However, parameterizing those dynamic costs allows our framework to incorporate them into cost assessments.

The following example shows one approach that can be used by our *TrustKnowOne* framework to determine the cost of processing a particular *graph expression*.

Expression 5.2 Temperature within range metric

In order to determine whether a sensor's temperature is within a specified range, we can use a number of mathematical *graph expresssions* and two *system parameters*, the temperature to compare against (*threshold*) and the acceptable *range*.



We need to associate cost factors with types of *graph expressions*. Here, we choose 1 for mathematical computations and 0 for retrieving attribute values from *graph components* and using *system parameters*. In this specific example we apply the *metric* to a set of 10 sensor *element nodes*.

As shown in figure 5.4, the cost of evaluating a particular *expression* depends on the cost factors of its inputs and parameters (for *model expressions*) as well as its own cost factor. Note that while we only discussed a limited example of performing

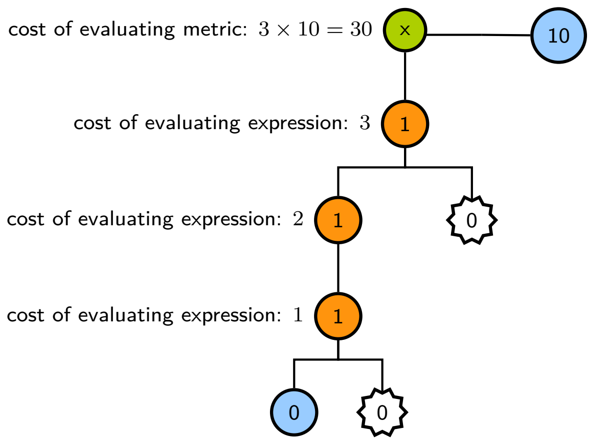

cost of evaluating metric: $3 \times 10 = 30$
×
10
cost of evaluating expression: 3
1
cost of evaluating expression: 2
1
0
cost of evaluating expression: 1
1
0
0

Figure 5.4: Cost assessment for *temperature within range metric* evaluated on 10 sensor *element nodes*

a cost assessment it works for both processing and assessment approaches because our framework models them both as *expression trees*.

Since the evaluation of *expression trees* is performed hierarchically, determining the cost of these *expression trees* can be achieved in a similar way as shown in figure 5.4. This formalization of *knowledge processing* cost assessments provides a number of advantages. First, it enables the analysis of different approaches found in literature. Second, because of the modeling as *graph expressions* to which cost factors can be attached, processing and assessment techniques can be compared to each other using a common methodology. Third, one of the problems found in a variety of other approaches is the lack of formalization which inhibits their modification, improvement, and reuse in other application scenarios. Our *TrustKnowOne* framework provides a formal method for cost assessment of *knowledge processing* approaches that is flexible and extensible.

The output of the *knowledge processing* phase is the graph model augmented with the "processed knowledge" as well as additional quality and trust assessments based on "global" meta information, context, and relationships. This becomes the input to the *knowledge evaluation* phase.

5.4 Knowledge Evaluation

The third and final layer of our *TrustKnowOne* framework is *knowledge evaluation*. Making decisions is a difficult process. Furthermore, many decision processes do not even consider quality and trust aspects because they are complicated and their integration is problematic. However, our *TrustKnowOne* framework provides the ability to model various decision processes while incorporating trust and quality of data.

These decision processes are represented by *graph expressions* similar to the ones used in the *knowledge processing* component. This allows one to operate directly on the *graph components* and the assessment provided as input by the *knowledge processing* phase to derive decisions. As a result of the *knowledge evaluation* phase, we have a set of decision options derived by evaluating a *decision process* on the data provided by the *knowledge extraction* and *processing* phases. These options include confidence assessments that can be traced all the way back to individual data elements and data sources as well as cost metrics of the entire decision process. Furthermore, the *knowledge evaluation* component provides the ability to increase the confidence in a decision by either requesting additional data or reevaluating existing data.

In order to provide the functionality described above, we discuss the following aspects of the *knowledge evaluation* component.

- Providing a formal description of decision making techniques using *decision processes*
- Incorporating quality and trust assessment aspects into decision making
- Requesting additional and challenging existing data if configurable confidence thresholds are not reached
- Providing interfaces to other systems (e.g., notification, propagation)
- Including evaluation lineage through the formal definition of *decision processes* as

graph expressions

- Determining cost for decision making approaches thereby enabling evaluation and comparison

5.4.1 Knowledge Evaluation: Architecture

In our *TrustKnowOne* framework, we model *decision processes* as *graph expressions.* The *knowledge evaluation* phase provides decision options based on the knowledge and assessments derived from the previous phases. This includes knowledge in the form of *graph components* and "local" assessments from the *knowledge extraction* component. In addition, we are able incorporate additional processed knowledge (i.e., knowledge as the result of processing of knowledge) and "global" assessments derived by the *knowledge processing* component. *Knowledge evaluation* is able to utilize a variety of *decisions processes* to evaluate this knowledge and determine the possible decision options.

Note that our evaluation approach follows the decision principles outlined by Pearl [126] which we incorporated into our framework as follows.

Rational Criteria We provide measurable factors in terms of quality and trust assessment *metrics* based on which a particular decision can be chosen over another.

Flexible Specification Our *abstract graph model* provides a unified and formal approach to modeling data, knowledge derived from data, and uncertainty (i.e., quality, trust) assessments.

Efficient Algorithms The *knowledge evaluation* part is able to base decisions on the rational criteria, data model, and assessments through the use of *graph expressions.*

Since *decision processes* are implemented using *graph expressions* we are able to model various simple (e.g., voting, ranking, weighting) as well as complex decision making (e.g., Markov decision processes [137], ensemble classifiers [153]). Note that this approach represents a natural extension of the processing techniques used in the *knowledge processing* component.

It is important to point out that our framework does not require a decision to be based on only a single quality or trust assessment approach. Instead, the *TrustKnowOne* framework incorporates ideas from ensemble learning [119, 134, 165] and consensus learning [12, 55, 116]. The former combines a set of topic specific techniques into a larger decision engine while the latter deals with changes in time and topology as found in dynamic scenarios. The flexibility of our *abstract graph model* allows decision processes of any kind to be incorporated. Since they are graph-based, approaches such as generic aspects [81], valuations [150], evidential networks [14] and expected outcome [13] which are frequently used in business applications are a natural fit for our *abstract graph model*.

Furthermore, *knowledge evaluation* incorporates trust and quality of data into the decision options. In particular, our framework provides a flexible *knowledge evaluation* approach that combines knowledge and assessments in ways ranging from simple voting or quorum schemes to complicated formulas that require a large number of parameters. Note that the *knowledge evaluation* component has access to all data managed in the *abstract graph model* as well as meta information and assessment metrics provided by the *belief engines* of the *knowledge processing* component. This allows us to enhance *decision processes* significantly by enabling context and various quality and trust assessments to affect decision making. For instance, *decision processes* have the ability to focus on high quality data or ignore data with low trustworthiness. While the specifics may depend on the particular *decision engine* used, our framework makes additional meta information available to be incorporated.

By extension, the decision of whether a data source is providing high quality and trustworthy data can be based on the assessment of data quality and trustworthiness of the individual data elements used in the decision. Furthermore, the *knowledge evaluation* component is able to maintain a history of previous decision options thereby allowing iterative improvements of decision making approaches.

5.4.2 Knowledge Evaluation: Requesting Additional or Challenging Existing Data

An important aspect of our framework is its ability to deal with changes in dynamic environments. Hence, the *knowledge evaluation* component needs to be flexible enough to address these changes (see [54, 57]). In particular, we provide the ability to ask for more data and perform challenges on existing data. As discussed earlier, this incorporates reinforcement learning [83, 163] into the *TrustKnowOne* framework. For instance, requesting additional data can be performed selectively by evaluating cost metrics of data sources and determining the one with the best estimated return. Furthermore, there are often multiple approaches to combining available information and deriving knowledge from it. Therefore, *knowledge evaluation* could also adjust which quality and trust assessments it incorporates as part of the decision process. This means the application of different or improved *belief engines* such as choosing more complex ones to increase decision confidence.

Since we provide confidence assessments with every decision option, thresholds can be used to determine which action to take. For example, within intrusion detection systems one could apply our framework in a manner where additional data is requested from the monitored resources until the decision of whether they are trustworthy or have been compromised can be made with a certain level of confidence.

5.4.3 Knowledge Evaluation: Interfaces to Other Systems

Knowledge derivation processes are often part of a larger system designed to solve a scenario specific problem. For instance, determining which resources have been compromised in an intrusion detection system should trigger notifications to relevant parties and countermeasures from the system administrators. Because of the flexibility of our *abstract graph model* where *graph expressions* are used to perform a variety of functions (e.g., knowledge processing, *belief engines*, *decision processes*), one could associate certain administrative *graph expressions* with appropriate actions to be executed. Since *knowledge evaluation* provides a set of decision options with confidence assessments, these *graph expressions* can be used to trigger various notifications based on configurable confidence thresholds (e.g., anomaly detection, outlier notification). In addition, this approach allows queries from other processes (i.e., control systems, business logic) concerning decision confidence, processed knowledge, and trustworthiness of data including its sources which enhances the usefulness of the *TrustKnowOne* framework.

5.4.4 Knowledge Evaluation: Evaluation Lineage

As discussed in the *knowledge extraction* and *knowledge processing* components, it is important to keep track of meta information about where data originated and how it was processed. Since *decision processes* are implemented as *graph expressions* similar lineage information (i.e., *metric*, *graph components* used) can be attached to the resulting decisions.

Lineage allows for a key aspect of *knowledge evaluation* which is the ability to determine the impact and relevance of specific data elements. In particular, we will base our approach on the idea of *minimum redundancy* where data should be reasonably separated in terms of their contribution and *maximum relevance* which means that only measurements with the highest relevance should be included [131].

This approach has several advantages. First, it allows the framework to perform knowledge and assessment model reduction which reduces complexity and increases performance by including only a subset of the original graph components in the *knowledge evaluation* component. Note that similar approaches include principal component analysis [111]. However, one of the problems with reducing the amount of knowledge and assessments is that it can potentially decrease the overall quality of the decision. Second, *knowledge evaluation* is able to optimize *decision processes*. By providing the formal descriptions of *decision processes* our framework enables evaluation and comparison of decision approaches. In particular, techniques such as maximum likelihood estimation [3] can be used on sample knowledge and assessment distributions to determine suitable parameters for *decision processes* as well as overcoming problems with partial evidence during the decision making. Third, given the extensive lineage information provided by the *knowledge extraction* and *knowledge processing* components we can determine the usefulness of certain data sources which in combination with cost assessments allows them to be evaluated and compared. This aspect can prove extremely valuable in resource constrained (i.e., cost, time, performance) as well as dynamic scenarios (i.e., determining from which data source to request additional data).

5.4.5 Knowledge Evaluation: Cost Assessments

Since *decision processes* are modeled as *graph expressions* we use a similar cost assessment approach to the one performed by the *knowledge processing* component. In particular, parts of the *decision processes* can be annotated to include cost factors representing aspects such as resources and time required to arrive at a decision. As discussed in the architecture, *knowledge evaluation* is able to incorporate requests for additional and challenge existing data. Because both of these costs are usually dynamic the resulting cost factors need to include parameters to reflect this. Hence, the cost of the *knowledge evaluation* is the sum of the *decision process* as well as any addition data requests or challenges that need to be performed as part of the decision making.

$$cost_{knowledgeevaluation} = cost_{decision} + cost_{data}^{additional} + cost_{data}^{challenge}$$

The formal representation of *decision processes* using *graph expressions* provides advantages similar to the ones discusses in the *knowledge processing*. First, we are able to perform a detailed cost analysis of individual decision techniques. Second, the formalization as *graph expressions* allows for comparison based on cost factors. Third, *decision processes* can be reused in different application scenarios and in order to improve decision approaches, adjustments can be made by simply extending *graph expressions* in order to achieve better cost factors.

The overall cost of performing knowledge derivation is the sum of the costs of the individual framework components.

$$\begin{aligned} cost_{knowledgederivation} = \\ cost_{knowledgeextraction} \quad &+ cost_{knowledgeprocessing} + cost_{knowledgeevaluation} \end{aligned}$$

Note that our *TrustKnowOne* provides a complete cost assessment approach that includes all aspects of knowledge derivation. Specifically, our formalization allows for individual approaches (i.e., extraction, transformation, “local” and “global” assessments, processing, decision making) to be evaluated, compared, and improved. Since our *TrustKnowOne* framework provides a clear separation between individual phases, the combinations of different approaches, especially in terms of *belief engines* with *decision processes*, can be evaluated in detail in order to find the best possible for a particular scenario. Furthermore, using parameters in cost factors enables our framework to perform cost assessment in both static and dynamic environments.

5.5 Evaluation of Model Vulnerabilities

Our *TrustKnowOne* framework provides a formal approach for knowledge derivation that incorporates trust and quality of data. Given the fact that there is an overabundance of different decision and assessment techniques all with their specific strengths and weaknesses, it becomes crucial to assess the impact of attack models (see [162, 180]) in order to choose a combination of approaches that is *fair, reliable, and secure.* Therefore, we need to distinguish between *operational system impairments* that are random and the assumption that *correlated changes* (possibly across time and space) in data are often part of an attack.

In general, attack scenarios can affect two parts of our framework, data and context. In particular, within the *knowledge extraction* phase, we face the problem of malicious or compromised data sources and incorrect "local" meta information. As the *knowledge processing* phase depends on this data from *knowledge extraction* in terms of *graph components*, it is indirectly affected by any attack scenario. Furthermore, malicious "global" information could be incorporated at this phase. The *knowledge evaluation* phase is also indirectly affected since its *decision processes* are based on the data as well as assessments provided by the previous phases.

One of the advantages of our framework is that it enables the evaluation of different attack models (see [162, 180]) on an individual approach (i.e., *belief engine*, *decision process*) as well as the entire knowledge derivation process. In particular, we determine the robustness of approaches based on their ability to perform their respective function (i.e., processing, assessment) with and without an attack present. In case of an individual approach we take the result of no attack as a baseline and compare it against the results achieved during various attack scenarios. As for the evaluation of the entire knowledge derivation the process is similar while the baseline is represented by the decision options available when no attack is present.

Using the robustness metrics in combination with cost assessments we can compare the relative value and robustness of different schemes in various scenarios (e.g., [54, 57, 99, 122]). Note that, since all the decisions are ultimately based on some data, we can evaluate the impact of missing, inaccurate or purposefully modified information using the approach discussed above.

5.6 Chapter Summary

The value and novelty of the *TrustKnowOne* framework lies in its formal description of knowledge derivation and assessment. We utilize the flexibility of our *abstract graph model Berlin* to manage heterogeneous data. Furthermore, processing as well as performing assessments are formalized as *graph expressions* that are evaluated on the *abstract graph model.*

Our framework is divided into three phases, each with their clear responsibilities and boundaries. *Knowledge extraction* provides an *adapter* approach to data acquisition that transforms raw data into *graph component* equivalents. Furthermore, it assesses data on a "local" level that does not require additional context. *Knowledge processing* enables the implementation of processing approaches as *graph expressions* in order to derive additional derived knowledge. It also incorporates *belief engines* which purpose is to model "global" assessments that includes relationships. *Knowledge evaluation* provides the ability to arrive at decisions using *decision processes* with various data, processed data, "local" and "global" assessments available as basis.

For each of the components there are a number of aspects that make the framework stand out from regular processing approaches found in literature. *TrustKnowOne* is capable of dealing with dynamic environments by allowing various data acquisition models, requesting additional data from data source, and challenging existing data. In addition, the framework provides formal means to integrate data from different sources with varying levels of trust.

The use of *graph expressions* to model approaches enables the formal description of *belief engines* for assessments and *decision processes* for decisions thus enabling evaluation and comparison. Furthermore, *graph expressions* can easily be computed in parallel thereby ensuring scalability. One of the main benefits of our approach is the way the framework provides lineage information and cost assessments by annotating *graph components.* As such, our *TrustKnowOne* framework is able to improve knowledge derivation through the use of an *abstract graph model* on which *graph expressions* representing knowledge and assessments are evaluated.

6

Scenario Analysis

The proposed framework will be of value to many applications such as sensor networks, participatory sensing, smart grids, cloud computing, and health care. In each of these cases data is obtained from geographically distributed heterogeneous sources, data is then processed and decisions need to be made. However, managing and integrating data from distributed heterogeneous sources as needed in these types of scenarios presents a variety of problems.

In the previous chapters we presented the *TrustKnowOne* framework which allows trust and quality of data aspects to be incorporated into knowledge derivation processes. Here we present its application to three distinct scenarios. As part of the discussion we will highlight how our approach exhibits both the formalization and flexibility necessary to model each of the realistic scenarios. These scenarios are used to confirm the advantages of the *TrustKnowOne* framework over current approaches.

We focus our analysis on the following representative and realistic scenarios. The selected scenarios and their implementations are realistic in terms of being geographically distributed, exhibiting time dynamics, and consisting of large and diverse data sets. First, we discuss how we can evaluate the trustworthiness of Smartphone Apps by incorporating a variety of relationship and context assessments. We show that this approach yields a significant improvement over current methods that are based on basic App attributes [95]. Our data set for this scenario contains a total 11326 Apps, 790940 reviews (651801 with text, 139139 without) as well as 134 different kinds of permissions captured in July 2012. For this purpose we developed a web crawler to pull the real and rich App attributes out of Google Play (Android Market). As such, our data is a diverse representation of realistic data with complex attributes and relationships.

Second, we apply our framework to distributed collaborative sensing in the domain of radiation detection. Here, we deal with changes in sensor values over time as well as complex relationships between them. In particular, we combine data from three data sources amounting to ≈ 2.5 million time stamped data points over the course of nine months which are geographically distributed across Japan. Two of the data sets were provided by the International Atomic Energy Agency [75] whereas the third data set from Safecast [144] represents measurements taken from thousands of people in a collaborative sensing effort. As such, the Safecast [144] data represents a challenging data set in terms of correlating related measurements, a common challenge in collaborative sensing environments. Thus, the measurements captured in the three data sets provide a realistic basis for evaluating our framework.

Third, intrusion detection provides a dynamic and challenging environment for knowledge derivation because there exist a wide variety of approaches to determine trustworthiness of system nodes. We discuss how our framework is able to formalize one approach [54] in order to be able to compare and evaluate it against a number of attacks. Our evaluation involves simulation of several dynamic systems with up to 60 nodes generating ≈ 9000 time stamped test messages over 75 days. The scope of this scenario is realistic for demonstrating the effects of a variety of attacks and evaluating trust assessment approaches on intrusion detection systems.

6.1 Trusting Smartphone Apps

Smartphones are becoming the mobile hubs of information for many people and companies. What started as a way to provide users with the flexibility of installing small software components called Apps to enhance the usability of their phone has grown into a global market with hundreds of thousands of applications built by thousands of developers. However, while there are plenty of well established companies developing useful applications or entertaining games there is no easy way to differentiate them from companies that put users at risk or worse are directly distributing malware or spyware.

One attribute that is often used in distinguishing "good" Apps from "bad" ones are their ratings. Nevertheless, research has shown that this can prove to be an unreliable metric, especially in cases with low rating counts. Reviews are also supposed to provide the user with an assessment of an App's trustworthiness by *real people*. However, fake reviews written by collaborators of the developer or the developer himself are common to boost an App's ranking. How is the average user able to distinguish between real and fake reviews? Finally, Apps run inside a security sandbox and need permissions to interact with the smartphone and the data stored on it. The problem is that users are usually not aware of what specific permissions mean or why they need to be granted.

In this scenario[1] we present a trustworthiness assessment model for Apps that takes into consideration these factors as well as others to provide the user with an indication of whether an App can be trusted and if so why. Furthermore, the model incorporates various relations between Apps and we discuss whether or not they should have an impact on the individual App's assessment. The research demonstrates that in order to make a decision to install an App one has to consider more than just App information and look into its associated meta data as well. The *TrustKnowOne* framework presented in chapter 5 enables the modeling of the smartphone App trustworthiness scenario

[1]A version of this scenario was published in Martin Kuehnhausen and Victor S. Frost. Trusting Smartphone Apps? To install or not to install, that is the question. In *2013 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support*, 2013

discussed here.

6.1.1 Trusting Apps: Overview

Until recently personal information was stored on users' home computers and business information was stored on company servers. Each had developed certain mechanisms to secure their information. Users set up passwords, firewalls and antivirus scanners on their machines and companies employed virtual private networks, sophisticated access control and intrusion detection systems. While this has not changed much over the past years, what has changed is that information has moved from these "protected" areas to mobile phones. Phones have made the transition into smart devices that are powerful enough to perform various functions that used to be limited to personal computers, laptops or servers. Furthermore, what used to be separated, personal and business information is now mostly merged on a single device which causes security issues. While some solutions have been developed in order to protect personal and business information, most notably virtualization of multiple systems on a single smartphone [7], other areas such as the protection [45] and control over cloud and mobile data remain problematic [56]. Furthermore, there is a recent initiative to use smartphones as payment methods replacing credit cards such as Google Wallet [61].

One of the major threats for information stored on a smartphone are Apps that the user installs. While many of them are used to extend features of the phone and make it more usable or efficient, others may be malicious and only interested in harvesting information [48]. The problem is that there is often no clear distinction between the two, e.g., some Apps provide useful features while also collecting a lot of information.

The domain of mobile phone applications is inherently dynamic with changing App attributes, relations and trust assessments as well as external context in the form of meta about the Apps from other sources. Thus the App domain is well matched to our trust framework which was initially outlined in [96] and discussed in detail as part of this research that allows for knowledge extraction, processing, and evaluation while incorporating quality of data and trust.

We use three common approaches to evaluating perceived trust in Apps – ratings, reviews and permissions – that when taken into consideration on their own they are flawed. Therefore, as part of this dissertation we propose several trust assessments for the approaches and show that the evaluation of basic App properties in combination with these assessments can be useful. In particular, we discuss why each of the assessments is necessary and evaluate their impact on the trustworthiness of a Apps as perceived by the user.

The goal here is to make the user aware of any trust issues related to an App by providing confidence metrics for its attributes because it improves the overall decision process of whether or not to install an App. It is out of the scope of this discussion to determine if an App is malicious or spyware as discussed in [132]. This determination is hard since, as stated earlier, many Apps provide useful functionality while also exposing private information (requesting read access to contact lists, calendars and social network accounts, etc.). However, we develop metrics that can be used to alert users to take a closer look at questionable Apps.

6.1.2 Trusting Apps: Framework Modeling Approach

Determining if Apps are trustworthy or not is a large scale data mining problem since the number of Apps available is large (>500,000) and relationships between them complex (similar set of permissions, one person reviewing multiple Apps, etc.). However, we propose to utilize a graph modeling approach where Apps and other related information such as reviews and permissions are represented as nodes which allows us to describe the various relationships as edges in a graph. This approach makes it easy to traverse and correlate information by choosing a particular App and limiting the number of related items (hops in graph terms) to consider.

As stated above it is important to look at the information available directly such as average rating and the number of reviews. However, also evaluating meta data like whether a review is positive or negative or whether the App's permissions requested are reasonable may yield a better overall trustworthiness assessment. The problem is that retrieving and utilizing this information is more complicated. For example, we could assess the perceived sentiment of a review by the appearance of keywords (e.g., good, great, bad) and the risks of a set of permissions by comparing them against other App's within the same category. However, in order to do this we need a flexible framework that allows us to incorporate relationships between Apps, reviews and permissions as well as meta information such as sentiment and rankings (position in Top Free, Top Paid, etc.) into trustworthiness assessments.

Here, we have applied the *TrustKnowOne* framework which is able to extract, process and evaluate knowledge and complex relationships from data that incorporates trust and data quality assessments. First, we describe all relevant elements and relations to be included in a graph model. Second, one or more belief engines modeling trustworthiness are defined. They are able to utilize data from the graph model as well as meta information to provide confidence assessments. Third, one or more decision processes can use data from the graph model as well as incorporate confidence assessments from the belief engines. Here, we show its application to determine the trustworthiness of Apps.

A key element of the framework is the definition of metrics which can be thought of as "computable" items derived from the graph model and meta information. Belief engines and decision processes can be described using such metrics, which allows us to abstract processing and evaluating knowledge in a formal way that avoids having to deal with domain specific models. However, we need to be aware that sometimes data is incomplete. Hence, we can only compute metrics for data values that exist. In the case that values used to compute a metric do not exist, the metric itself does not exist for these values. It is important for the decision engine to factor in those missing values and the existence of metrics as they could potentially skew the overall assessment.

We developed a web crawler to retrieve information about Apps from Google Play (Android Market). In particular, we extracted the top 100 Apps in each of the 30 categories and their respective collections (Top Free, Top Paid, Top Grossing, etc.) in July 2012. Because of some overlap (the same App could appear in multiple collections) our data set contains a total 11326 Apps, 790940 reviews (651801 with text, 139139 without). There are 10444 Apps with permissions and we discovered 134 different kinds of permissions overall.

Using this data set combined with a goal of trust assessment we will demonstrate the following aspects of the *TrustKnowOne* framework for this scenario:

- Modeling heterogeneous data (Apps, categories, reviews, permissions) and relationships in our *abstract graph model* (section 4.1)
- Formalizing confidence assessments for App attributes using context and expected behavior as *belief engines* (section 5.3.1)
- Representing decision making as formal *decision processes* with the option of whether or not to incorporate confidence assessments (section 5.4.1)

An overview of how the scenario relates to individual components of our framework is shown in figure 6.1.1. Here, we introduce these components which we will discuss in detail throughout this section.

Knowledge Extraction In this scenario we model relevant smartphone entities such as Apps, categories, reviews, and permissions. Here, we utilize the web crawler described above to create a data set at a particular time instance. While it is possible to run the crawler at different times thus creating time series information for Apps we focus our discussion on one particular instance. As such the *element instance graph* is a static representation of Google Play (Android Market).

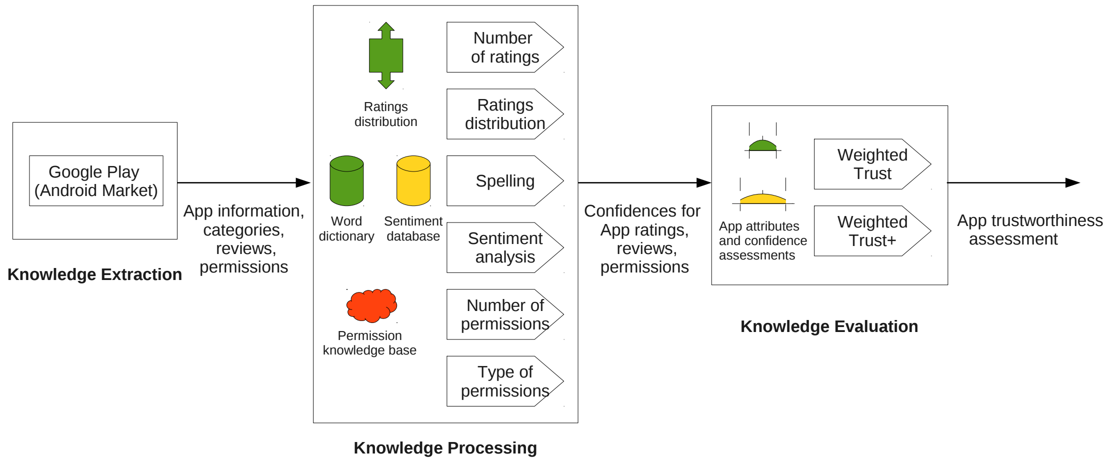


Figure 6.1.1: Framework overview for the trusting smartphone Apps scenario

Knowledge Processing We take into consideration context and expected behavior (e.g., ratings distributions, word dictionary, sentiment database, permission knowledge base) in order to derive confidence assessments for the App's rating value, reviews and permissions. In particular, belief engines showcase deterministic and probabilistic approaches in determining trustworthiness and quality of data. As a result we provide a variety of assessment that can be combined with basic App information to make decisions about trusting smartphone Apps.

Knowledge Evaluation Using the confidence assessments derived by the *knowledge processing* phase we have various options to incorporate them into the decision making process. Here, we compare two approaches. The first one, does not utilize confidence assessments and yields a trustworthiness assessment based only on the App attributes. However, the second approach relates them to the respective attributes thus enabling decision engines to form better decisions. We compare these approaches and discuss why the latter provides a better representation of the state of trusting smartphone Apps.

Next, we present a detailed analysis in which aspects of the scenario are related to the *TrustKnowOne* framework.

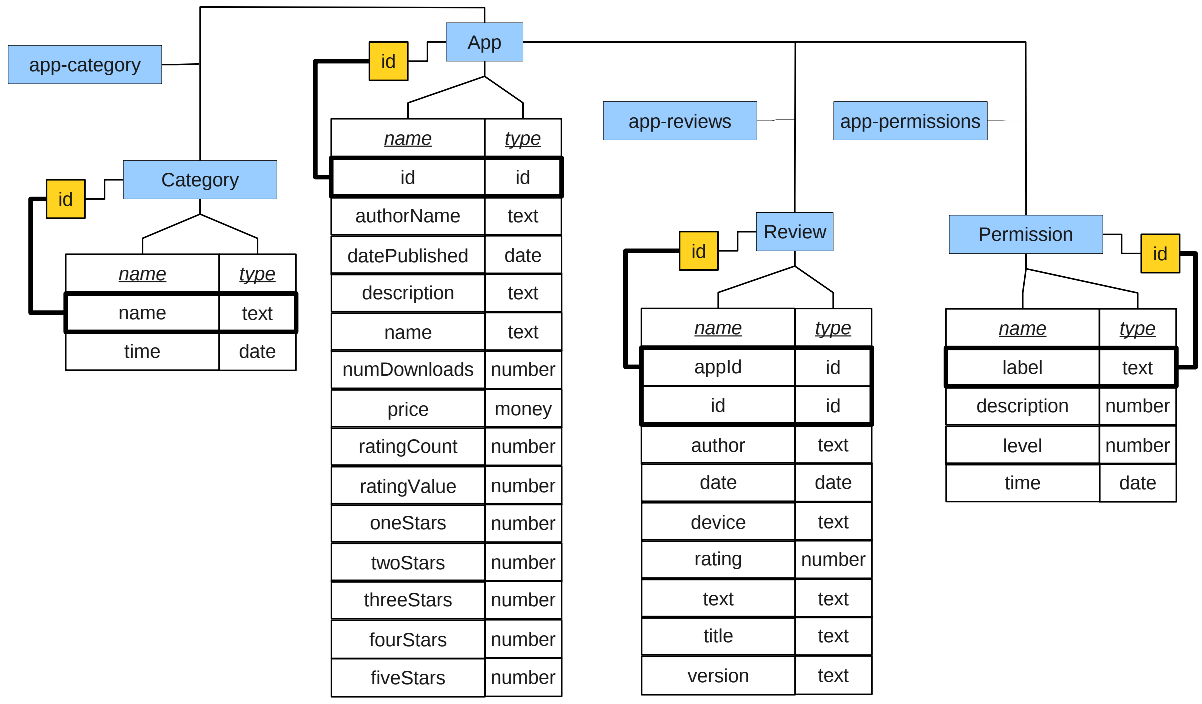


Figure 6.1.2: App *element description graph*

6.1.3 Trusting Apps: Knowledge Extraction

Smartphone Apps have a variety of properties and while most users are aware of basic ones such as the average rating or number of downloads they pay less attention to others (e.g., number of ratings, number of one star ratings, etc.). In addition Apps can form complex relationships with other Apps as well as categories, permissions, and reviews. This needs to be modeled accordingly if one is to derive trustworthiness assessments for Apps.

The abstract graph model we propose is shown in figure 6.1.2 where the key component is the individual *App element.* The App domain there are a variety of *attributes* which we can broadly classify into

informative author name, description, name

less relevant date published, number of downloads, price

relevant rating count, average rating value, number of one to five star ratings

Specifically, we will focus our trustworthiness assessment on the relevant *attributes* and will incorporate only those into our discussion of this scenario. Package names (e.g., com.google.android.apps.maps for Google Maps) are used to uniquely identify *Apps*. *Categories* are used to group *Apps* and we model them as *elements* that are identified by a *name*. The relationship between an *App* and its *Category* is expressed by the *app-category relation*.

Every *App* has some extended information associated with it. In particular, the *Permissions* it requires and the *Reviews* that were made of it. *Permissions* contain a *label* used to identify it, a *description* providing additional information, and a *level* which can either be *safe* or *dangerous*. The user installing an App needs to specifically request seeing safe permissions whereas dangerous permissions are automatically prominently displayed. *Reviews* describe user feedback for an *App*. It is actually a combination of rating and textual review. Thus, every *Review* has a *rating* but not every *Review* contains *text*. While there are a number of other *attributes* such as *author, date, device, title*, and *version* our focus is on the *rating* and *text attributes*. Since a *Review* is always associated with an *App* its unique identifier is a composite of a review id and the *App's* id. We model relationships accordingly by introducing *app-review* and *app-permissions relations*.

Note that here we only discuss *elements* and *relations* relevant to our trustworthiness approach (figure 6.1.2). Other components of the abstract graph model such as other *elements* (e.g., *Badge*, *PermissionGroup*, *Collection*), additional *attributes* (e.g., file size, software version), and more complex relationships (e.g., also installed, also viewed, same developer) are present but not considered in the initial trustworthiness approach we present here. However, the flexibility of the proposed framework facilitates their inclusion in the future.

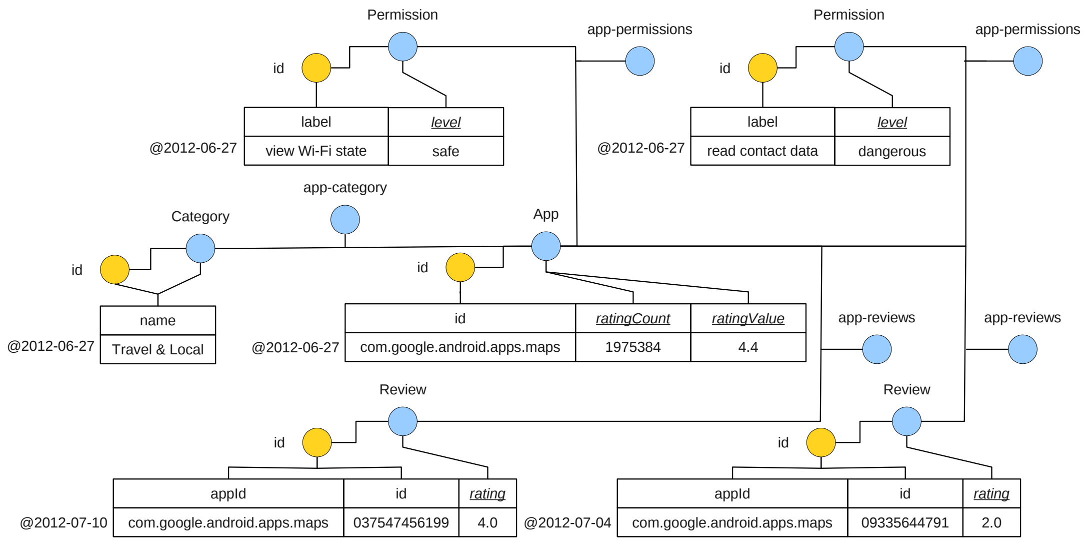


Figure 6.1.3: App *element instance graph* example for Google Maps

Figure 6.1.3 shows a subset of the data that is being modeled. For clarity's sake we have chosen only a few relevant *attributes* to be displayed. In particular, we have a Google Maps Android *App element node* with the rating count (number of ratings) and rating value (average rating) *attributes*. We show two *Review element nodes* with their rating *attributes*. Note that these *element nodes* make use of a composite id as specified above. They are connected to the *App* using the *app-reviews relation edge* accordingly. Furthermore, two example *Permission element nodes* with varying level states are shown. We express their relationship with the *App* as *app-permissions relation edges*. The *Travel & Local category element node* is interesting as it only has the *name attribute*. It is connected via the *app-category relation edge*.

6.1.4 Trusting Apps: Knowledge Processing

As part of this dissertation we propose a number of trustworthiness assessments ranging from 0% to 100% for App attributes. In particular, we consider assessments of ratings, reviews and permissions as well as relationships between Apps. We will discuss how they can be utilized and why they are necessary in determining an App's trust.

6.1.4.1 App Ratings

Of the several attributes that users look at when considering installing an App is its rating (on a 1 to 5 star scale). However, the often used average rating score is not necessarily a good indicator [29] because it is usually large, in our data set the average rating is 4.2 with a standard deviation of 0.64. Here, we discuss two ways of measuring the confidence a user should have in the average rating.

Number of Ratings As the number of ratings grows so should our confidence in the meaning of the average value. We have two options to derive at such a measure. On the one hand, we can compute the sample standard deviation s of the 1 through 5 star ratings and use this estimate to determine the standard error $SE_{\bar{x}}$. On the other hand, we propose to use the Student's t-distribution which does not require a known standard deviation and approximates a normal distribution as the degrees of freedom approach infinity. This allows us to propose a confidence metric using only the number of ratings n as

$$c_{\#rating}(n) = \begin{cases} 0 & \text{if } n \leq 6 \\ 1 - \dfrac{ST(n)}{\sqrt{n-1}} & \text{if } n > 6 \end{cases} \tag{6.1.1}$$

where

$$ST(n) = CDF^{-1}_{T\sim(n-1)}(0.975) \tag{6.1.2}$$

is the value of the inverse cumulative distribution function of the Student's t-distribution with $n - 1$ degrees of freedom at a two-sided 95% confidence interval with

$$\lim_{n\to\infty} CDF^{-1}_{ST\sim(n-1)}(0.975) = 1.96 \tag{6.1.3}$$

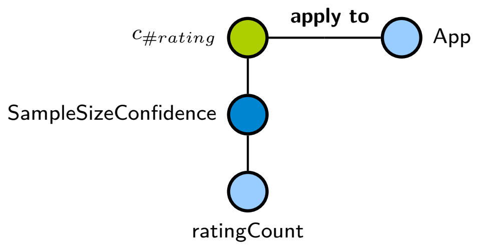
representing a sample estimation of the standard deviation approaching normal distribution. Hence, for very large n the proposed metric can be related to the standard error as

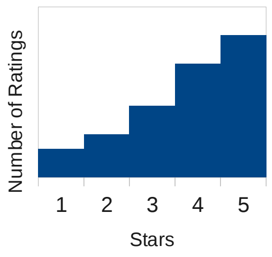
$$\lim_{n\to\infty} \frac{ST(n)}{\sqrt{n-1}} \approx \frac{s}{\sqrt{n}} = SE_{\bar{x}} \tag{6.1.4}$$

One can easily see that using the Student's t-distribution is preferable because the standard deviation for a low number of biased ratings would yield an undesired high confidence. For example, 10 five star ratings give $s = 0$ and $ST(10) = 2.262$ which results in a confidence indicator of $1 - \frac{s}{\sqrt{10}} = 100\%$ for s and a confidence indicator of $1 - \frac{ST(10)}{\sqrt{10-1}} = 24.6\%$ for the proposed approach. Note that the Bayesian rating in [29] could also be used to adjust the value of the rating. In terms of our abstract graph model we have:

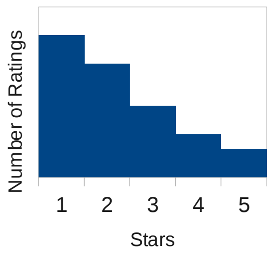
Expression 6.1.1 $c_{\#rating}$

The confidence in the number of ratings $c_{\#rating}$ is expressed using the *sample size confidence model expression.* It is based on equation 6.1.1 and here we apply it to the *rating count attribute* of an *App*.

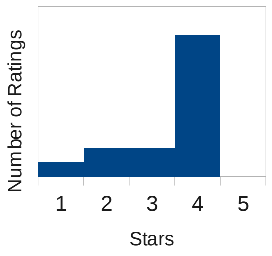




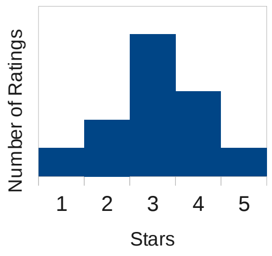
(a) Skew left: trend is towards high ratings



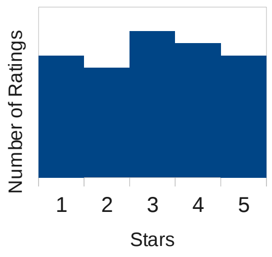
(b) Skew right: trend is towards low ratings



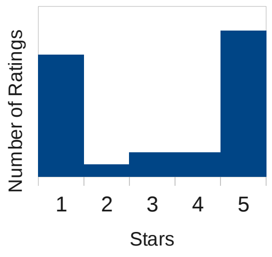
(c) Unimodal constant: ratings are leaning heavily towards a constant



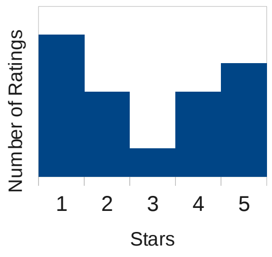
(d) Normal: ratings are distributed around one mean



(e) Uniform: ratings do not have a meaningful separation



(f) Bimodal constant: ratings are leaning heavily towards to the extremes



(g) Bimodal normal: ratings are distributed around two means

Figure 6.1.4: Examples of rating distribution types

Ratings Distribution Type Figure 6.1.4 shows a variety of distribution types possible for App ratings. They are important in determining the *meaning* of the average rating for an App. In particular, we need to consider the following cases:

Unimodal The average rating is a reasonable reflection of the App's quality where the majority of ratings fall within the range of the average rating (Figures 6.1.4a-6.1.4d).

Uniform The ratings do not have a meaningful separation (trending towards good or bad) and hence, the average is not an accurate interpretation of the overall ratings (Figure 6.1.4e).

Bimodal The ratings fall into two extreme categories (usually really good and really bad). This is troublesome since the average rating is a deceiving reflection of the App's quality (Figures 6.1.4f and 6.1.4g).

We propose the following *weighted means difference* algorithm to discover bimodal trends in these distributions. The result is a metric of how close the distribution is to either one or two constants. Note that, since we are only interested in discovering a trend we do not need to separate uniform and symmetrical bimodal distributions. However, in order to make this distinction one could use Shannon's information entropy [149].

Here, we develop a *weighted means difference* algorithm to ratings distributions but it can easily be generalized to other discrete or continuous distributions. Some resulting measures are shown in table 6.1.1.

First, we separate the n ratings into the following sets

Table 6.1.1: Result of the weighted means difference algorithm for ratings distributions (number of 1-5 stars)

Difference	Distribution
$\rightarrow 0.0$	constant
≈ 0.25	unimodal
1.0	uniform or bimodal
≈ 1.5	bimodal
$\rightarrow 2.0$	two constants

$$R = \{r_0, \ldots, r_n\} \tag{6.1.5}$$

$$R_s = \{r_0, \ldots, r_n | r = s\} \tag{6.1.6}$$

$$R_{low} = \{r_0, \ldots, r_n | r \in \{1, 2, 3\}\} \tag{6.1.7}$$

$$R_{high} = \{r_0, \ldots, r_n | r \in \{3, 4, 5\}\} \tag{6.1.8}$$

where r_i is the i^{th} rating in stars and $|R| = n$. The average rating is calculated as

$$\overline{R} = \frac{1}{n} \sum_{r_i \in R} r_i \tag{6.1.9}$$

Second, we consider the following special cases:

$|R_{low}| = 0$ There are only high ratings (four and five stars) and the distribution is either skewed left normal or a constant. Hence the *weighted means difference* only depends on the high ratings

$$\begin{aligned} wmd(R) &= \frac{|R_4| \times abs\left(\overline{R} - 4\right)}{n} \\ &+ \frac{|R_5| \times abs\left(\overline{R} - 5\right)}{n} \end{aligned} \tag{6.1.10}$$

$|R_{high}| = 0$ There are only low ratings (one and two stars) and the distribution is either skewed right normal or a constant. Hence the *weighted means difference* only depends on the low ratings

$$\begin{aligned} wmd(R) &= \frac{|R_1| \times abs\left(\overline{R} - 1\right)}{n} \\ &+ \frac{|R_2| \times abs\left(\overline{R} - 2\right)}{n} \end{aligned} \tag{6.1.11}$$

Third, we calculate the average of the lower and upper ratings sets

$$\overline{R}_{low} = \frac{1}{|R_{low}|} \sum_{r_i \in R_{low}} r_i \tag{6.1.12}$$

$$\overline{R}_{high} = \frac{1}{|R_{high}|} \sum_{r_i \in R_{high}} r_i \tag{6.1.13}$$

and factor in the number of ratings in each of them

$$w_{low} = \frac{|R_{low}|}{|R_{low}| + |R_{high}|} \tag{6.1.14}$$

$$w_{high} = \frac{|R_{high}|}{|R_{low}| + |R_{high}|} \tag{6.1.15}$$

to derive a *weighted means difference* with respect to the overall average rating

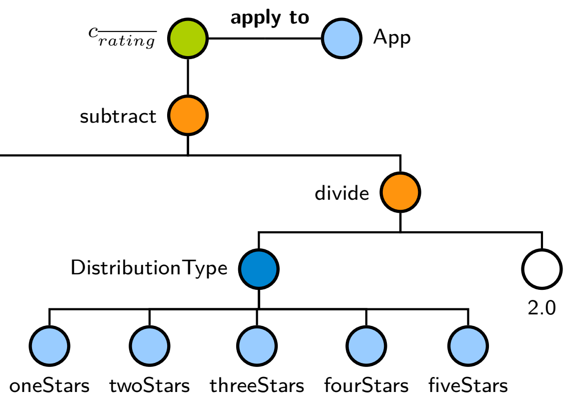
$$
\begin{aligned}
wmd(R) = \quad & w_{low} \times abs\left(\overline{R} - \overline{R}_{low}\right) \\
+ \quad & w_{high} \times abs\left(\overline{R} - \overline{R}_{high}\right)
\end{aligned}
\tag{6.1.16}
$$

Therefore, another confidence metric in the average rating is

$$
c_{\overline{rating}}(R) = 1 - \frac{wmd(R)}{2} \tag{6.1.17}
$$

Expression 6.1.2 $c_{\overline{rating}}$

The confidence in the ratings distribution $c_{\overline{rating}}$ is based on the described *weigted means difference* algorithm which is implemented in the *distribution type model expression*. As input we use the distribution of rating stars. Furthermore, we normalize the result to yield a confidence *metric* between 0% and 100% accordingly.



6.1.4.2 Reviews

Since users associate a review with a “recommendation” by real people, it is a valuable resource for evaluating an App. However, there are a number of problems associated with evaluating reviews. Influential fake and bad reviews can dominate over interesting and useful ones [112, 120]. Furthermore, users review Apps differently when they can keep their identity anonymous [41]. Identifying fake reviews is beyond the scope of this discussion. Here, we focus on two other metrics that typically influence a user's perception of a review. Let us consider the text of a review as a collection of words

$$T = \{w_0, \ldots, w_n\} \tag{6.1.18}$$

where w_i is the i^{th} word of text in the review.

Spelling Correct spelling can be an indicator of professionalism. Therefore, when looking at reviews we need to factor in the number of misspelled words. The proposed confidence metric in terms of spelling is defined as

$$c_{spelling}(T) = 1 - \frac{|T_{ms}|}{|T|} \quad \text{if } |T| \geq 1 \tag{6.1.19}$$

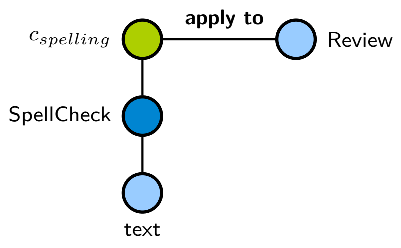
where

$$T_{ms} = \{w_0, \ldots, w_n | w \in T, w \text{ misspelled}\} \tag{6.1.20}$$

is the set of misspelled words. Note that for our spell checking purposes we use [1]. As part of our framework we can define this confidence as follows.

Expression 6.1.3 $c_{spelling}$

The confidence in spelling $c_{spelling}$ can be defined using the *spellcheck model expression* which implements equation 6.1.19. In this case we apply it to the *text attribute* of a *Review*.



Sentiment Analysis Whether a review is positive or negative can be evaluated using sentiment analysis. One approach is to describe the overall sentiment of a review is by finding words in the review that represent a positive or negative sentiment for the set

$$T_s = \{w_0, \ldots, w_n | w \in T, \exists \text{ sentiment for } w\} \tag{6.1.21}$$

with the overall sentiment defined as

$$sent(T) = \sum_{w_i \in T_s} sent(w_i) \tag{6.1.22}$$

where $sent(w_i)$ is the sentiment of a word and $sent(T)$ of the entire review. Because more text does not necessarily imply more words with sentiments we propose to make the sentiment proportional to the number of words in the review and bound it by the number of words with sentiments.

$$sent_p(T) = \frac{sent(T)}{max(\sqrt{|T|}, |T_s|)} \quad \text{if } |T| \geq 1 \tag{6.1.23}$$

This means that the same number of words with sentiments have more impact the shorter the review is. Furthermore, sentiment analysis is based on positive or negative values associated with specific words. Here, we use a word list by Nielsen [115]. Therefore, we need to normalize them to a common scale between 0 representing negative, 0.5 neutral and 1 positive overall sentiments which yields

$$sent_{scaled}(T) = \frac{sent_p(T) + abs\left(0 - sent_{min}\right)}{sent_{max} - sent_{min}} \tag{6.1.24}$$

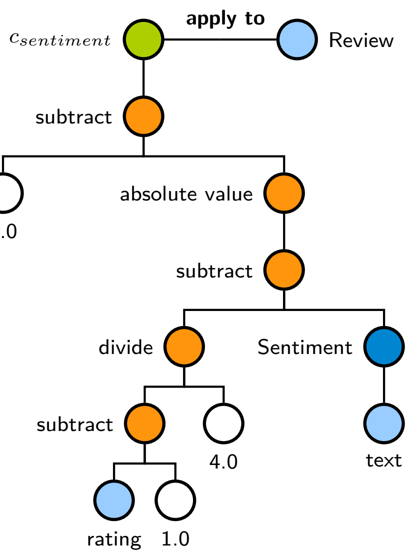
where $sent_{min}$ and $sent_{max}$ represent the minimum and maximum possible sentiment values used. This sentiment is a good additional indirect indicator of the App's quality. However, here we are interested in how closely the sentiment of the review text reflects the review's star rating. Discrepancies lower the confidence in a review which we leads us to define a confidence metric as

$$c_{sentiment}(T) = 1 - abs\left(\frac{r-1}{4} - sent_{scaled}(T)\right) \tag{6.1.25}$$

where $\frac{r-1}{4}$ is the rating given in connection with the review adjusted to range from 1 star (0% confidence) to 5 stars (100% confidence) and the overall confidence the difference between this rating and the review's sentiment. This approach is reflected in the following graph model representation.

Expression 6.1.4 $c_{sentiment}$

The confidence in the sentiment $c_{sentiment}$ is modeled as an *expression* tree that uses a series of mathematical operations compute the difference between the sentiment and its rating. In particular, the *sentiment model expression* reflects equation 6.1.24. We then combine the sentiment analysis result with several math *expressions* as discussed in equation 6.1.25.



6.1.4.3 Permissions

Apps require permissions to utilize smartphone system functionality as well as to read and write user data. Users are often overwhelmed by the complexity of permissions and even developers generally lack a thorough understanding of which ones are necessary and which ones are too invasive [47, 49]. However, permissions are like keys to information stored on the smartphone. We present several approaches to determine the trustworthiness of Apps based on the sets of permissions they require but focus on the dangerous ones.

Number of Permissions The number of permissions used within a particular category can be a good indication of how many permissions are adequate for an App in the specific category. Given the sets

$$C = \{c_0, \ldots, c_n\} \tag{6.1.26}$$

$$A_c = \{a_0, \ldots, a_n\} \tag{6.1.27}$$

$$W = \{A_0 \cup \cdots \cup A_n | \forall c \in C\} \tag{6.1.28}$$

where c_i is the i^{th} category, a_i is the i^{th} App in category c and A_c the set of Apps for a particular category c consider the following sets of permissions

$$\begin{aligned} P &= \{p_0, \ldots, p_n\} && (6.1.29) \\ P_a &= \{p_0, \ldots, p_n | \text{App } a \text{ has } p\} && (6.1.30) \\ P^c &= \{P_0 \cup \cdots \cup P_n | a \in A_c\} && (6.1.31) \end{aligned}$$

where p_i is the i^{th} permission, P_a the set of permissions for the App a, and P^c the permissions for the category c. We can define the average number of permissions for a category

$$\overline{P^c} = \frac{\sum_{a \in A_c} |P_a|}{|A_c|} \tag{6.1.32}$$

and overall

$$\overline{P} = \frac{\sum_{c \in C} \sum_{a \in A_c} |P_a|}{|W|} \tag{6.1.33}$$

This allows us to propose a model for the permission confidence using the following ratios for categories

$$c_{\#permissions}^{category}(n) = \frac{\overline{P^c}!}{n!} \times \overline{P^c}^{n - \overline{P^c}} \tag{6.1.34}$$

and

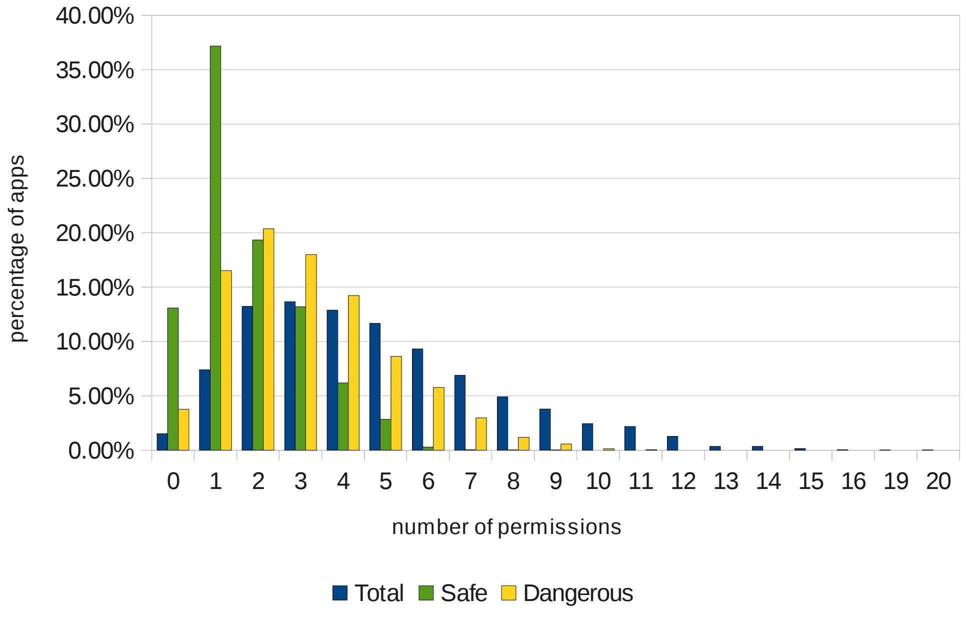


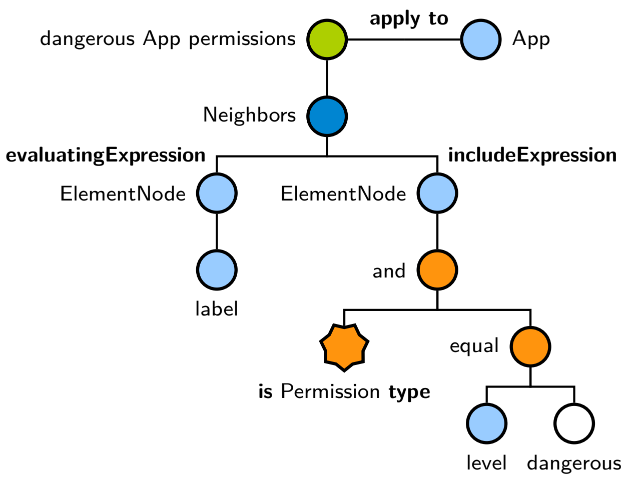
Figure 6.1.5: Distribution of App permissions

$$c_{\#permissions}(n) \quad = \quad \frac{\overline{P}!}{n!} \times \overline{P}^{n-\overline{P}} \tag{6.1.35}$$

overall where n is the App's number of permissions as well as $\overline{P^c}$ and $\overline{P}$ the rounded averages described above. Note that this confidence is based on the ratio of Poisson probabilities. We use the Poisson distribution here as it has the advantage of being displaying tail characteristics that are more suitable to describing the distribution of permissions as shown in Figure 6.1.5 where most Apps have only few permissions. As for our evaluation we focus on dangerous App permissions.

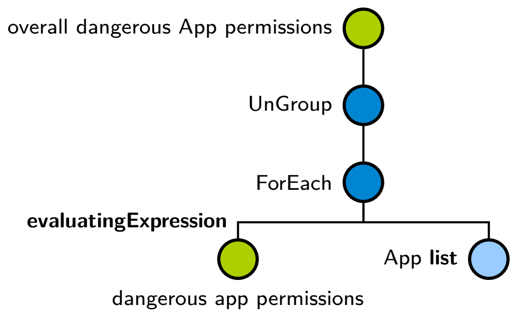
Expression 6.1.5 dangerous App permissions

Since permissions are always associated with an App we retrieve all its neighbors that match the Permission type and are dangerous.



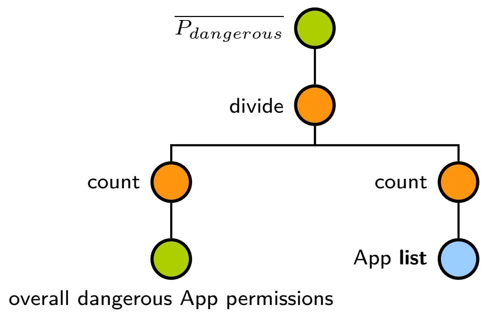
Expression 6.1.6 overall dangerous App permissions

The overall dangerous App permission *metric* retrieves dangerous permissions from all *element nodes* that are part of the *App* list. Two list *model expression* are used here. *For each* repeats the *evaluating expression* for all *Apps* while *ungroup* converts the lists of *App* permissions into one combined list of permissions.



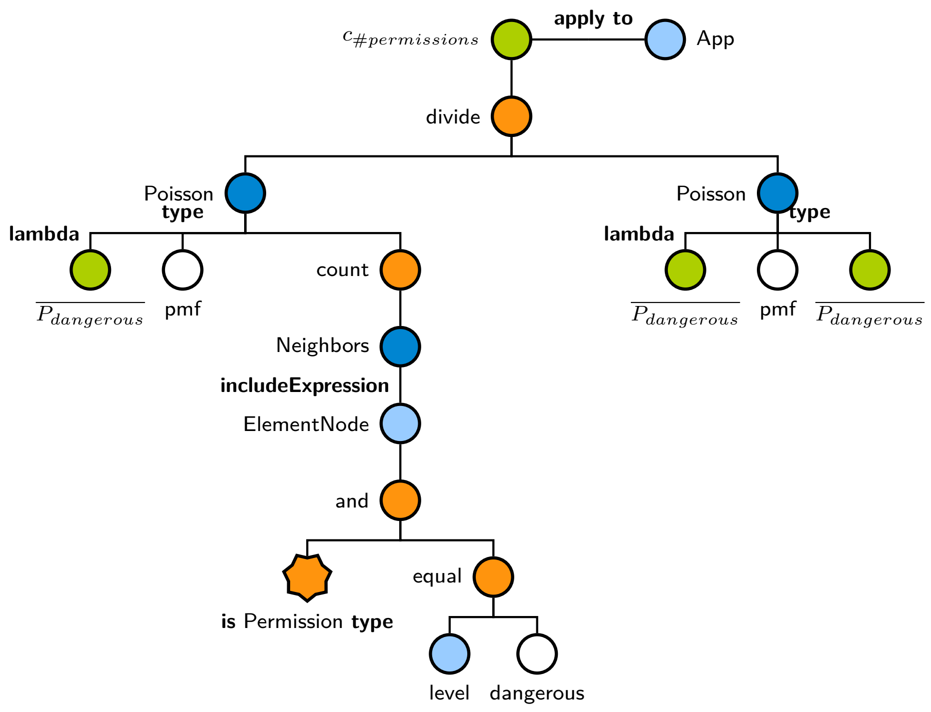
Expression 6.1.7 $\overline{P_{dangerous}}$

The average number of dangerous App permission $\overline{P_{dangerous}}$ then becomes a straightforward ratio of the number of all dangerous App permissions and the number of *Apps*.



Expression 6.1.8 $c_{\#permissions}$

The confidence in the number of dangerous permissions $c_{\#permissions}$ can be expressed using a combination of basic *expressions* and the *Poisson model expression.* Note that we utilize the average number of dangerous permissions $\overline{P_{dangerous}}$ both, as mean for the Poisson distribution and as best case for the resulting probability density.



Type of Permissions Because some permissions are more common than others (see table 6.1.2) we also consider the different types of permissions used by a particular

Table 6.1.2: Most used permissions across 30 categories from our data set of 11326 Apps with 134 different kinds of permissions as of July 2012

Permission	Apps with permission	Average rank
full Internet access	80.83%	1.03
view network state	54.88%	2.57
modify/delete USB storage contents		
modify/delete SD card contents	53.88%	2.67
read phone state and identity	39.63%	4.13
control vibrator	27.62%	5.43
prevent tablet from sleeping		
prevent phone from sleeping	22.38%	7.30
coarse (network-based) location	16.54%	8.10
automatically start at boot	14.13%	9.53
fine (GPS) location	11.38%	12.60
Market billing service	10.44%	13.40
discover known accounts	9.96%	12.30
take pictures and videos	8.29%	16.47
read contact data	7.97%	14.13
view Wi-Fi state	7.79%	13.13
Market license check	7.58%	13.33

App. Most used permissions are also reported in [47, 132]. We adapt the Jaccard set similarity [77] and propose the following for bags of permissions to compare an App's set of permissions with the weighted set of *average* permissions required by other Apps in the same category and overall. Note that a bag is a set of items where each particular item can occur multiple times. Hence, we normalize the confidence in a set of permissions by treating 75% similarity and above as 100% confidence for categories as

$$c^{category}_{permissions}(P_a) = min\left(\frac{4}{3} \times \frac{|\hat{P}_a|}{|\hat{P}|}, 1\right) \tag{6.1.36}$$

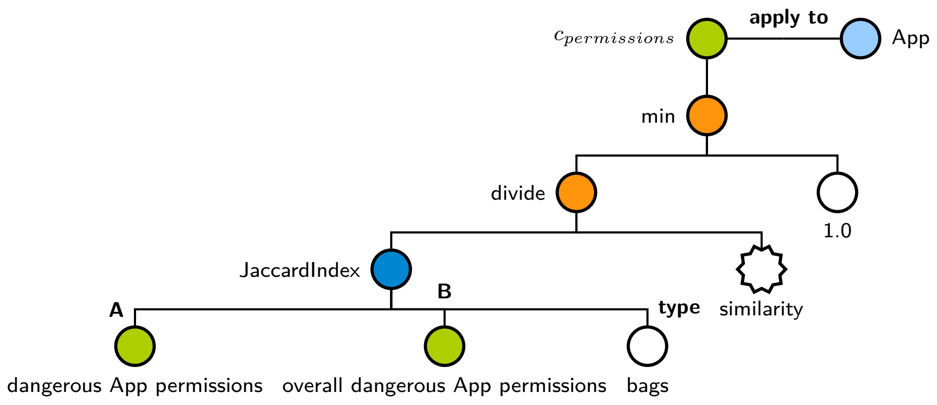
where $\hat{P}^c$ is the bag of all permissions $\{P_0 \uplus \cdots \uplus P_n | a \in A_c\}$ for a particular category and $\hat{P}_a$ is the bag of permissions $\{p_0, \ldots, p_n | p \in P_a\} \subseteq \hat{P}^c$ of an App. Similarly, for all Apps we define the confidence metric as

$$c_{permissions}(P_a) = min\left(\frac{4}{3} \times \frac{|\hat{P}_a|}{|\hat{P}|}, 1\right) \tag{6.1.37}$$

where $\hat{P}$ is the bag of all permissions $\{P_0 \uplus \cdots \uplus P_n | a \in W\}$ overall and $\hat{P}_a$ is the bag of permissions $\{p_0, \ldots, p_n | p \in P_a\} \subseteq \hat{P}$ of an App.

Expression 6.1.9 $c_{permissions}$

The confidence in the type of dangerous permissions $c_{permissions}$ is modeled using the *Jaccard index model expression* applied to the set of dangerous permissions of the particular App and the bag of dangerous permissions of all Apps. The similarity level above which the confidence results in 100% confidence can be adjusted using the system parameter *similarity*.



6.1.5 Trusting Apps: Knowledge Evaluation

In order to assess the trustworthiness of an App we consider the three basic attributes – average rating, average review score, and number of permissions – and their trustworthiness assessments as described above. We compare a decision process based only on these attributes with one that includes confidences for the attributes. We propose a basic decision process (more sophisticated decision engines are a topic for future research) as a weighted sum of the scaled attributes

$$trust(App) = \sum w_i \times m_i \tag{6.1.38}$$

where w_i is the assigned weight with $\sum w_i = 1$ and m_i one of the following metrics:

$m_1 = \frac{r-1}{4}$ the scaled average rating with r being the App's rating

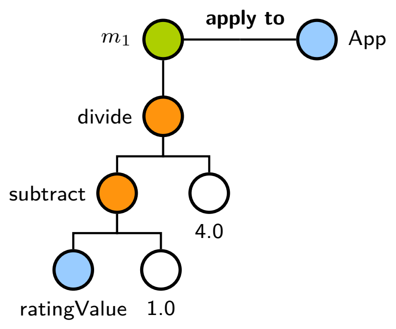
$m_2 = \frac{1}{|reviews|} \sum_{r_i \in reviews} \frac{r_i - 1}{4}$ the scaled average review rating with r_i as the review's rating

$m_3 = max(\frac{\varphi - p}{\varphi}, 0)$ the scaled number of dangerous permissions with p as the number of dangerous permissions and φ a scaling parameter (10 by default)

We can model these *metrics* as the following *expressions*.

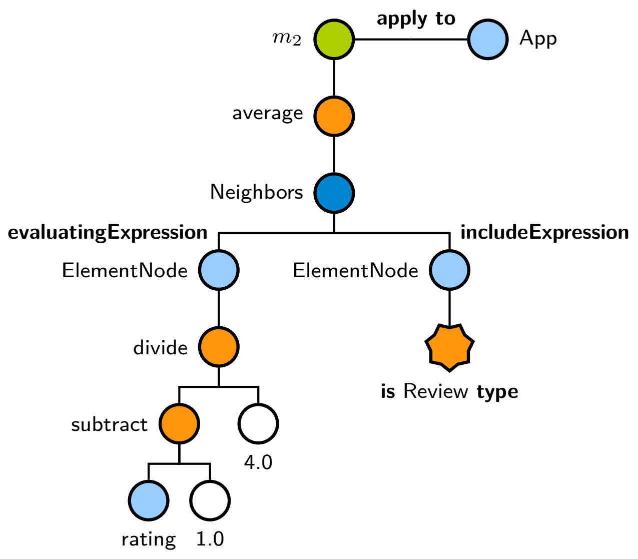
Expression 6.1.10 m_1

The scaled average rating m_1 can be modeled using basic math *expressions* which are performed on an *App's rating value.*



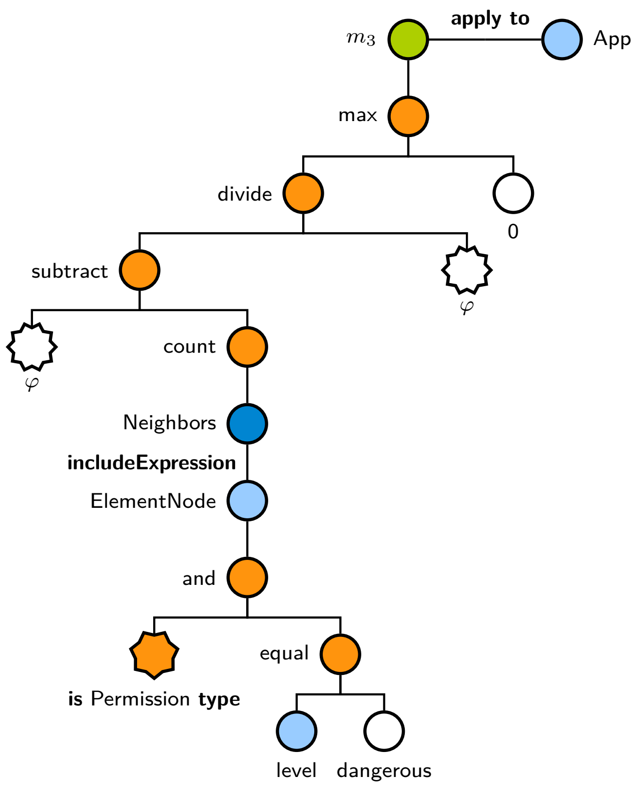
Expression 6.1.11 m_2

The scaled average review rating m_2 can be expressed using the *neighbors model expression* where the *rating* of each *Review* is scaled using math *expressions.*



Expression 6.1.12 m_3

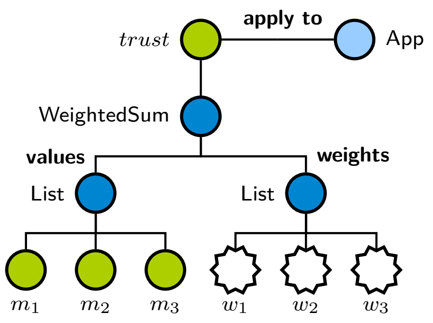
The scaled number of dangerous permissions m_3 is modeled by using the *neighbors model expression* to determine the set of dangerous permissions and math *expressions.* The scaling factor is expressed as an adjustable system parameter φ.



Given the *metrics* discussed above we can then define equation 6.1.38.

Expression 6.1.13 *trust*

The trustworthiness of an App considering only the basic attribute *metrics* scaled average rating, scaled average review rating, and scaled number of dangerous permissions is modeled using a *weighted sum model expression* on the previously defined *metrics* as specified by equation 6.1.38. We provide the ability to change the weighting scheme by adjusting the system parameters w_1, w_2 and w_3.



As shown in figure 6.1.6, each of the trustworthiness assessments ranges from 0% to 100% confidence and allows for a reasonable separation between "good" and "bad" Apps. Factoring these into our decision process we adjust each metric by the confidence in it. This means that if a metric is trusted 100% it does not change but that overall the lower the confidence is the lower the metric score will be. Note that, we associate the fact that a metric does not exist with 0% confidence. The proposed decision process becomes

$$trust^{+}(App) = \sum w_i \times m_i \times \left(\Delta m_i^1 \times \Delta m_i^2\right) \tag{6.1.39}$$

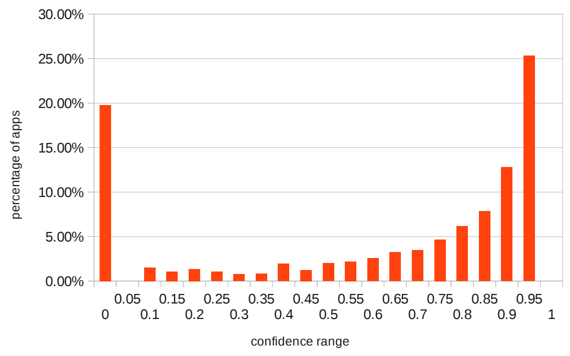
where w_i and m_i are the weights and metrics defined above which we adjust using the respective assessments Δm_i^1 and Δm_i^2 for each of the metrics. In particular:

$\Delta m_1^1 = c_{\#rating}(n)$ the confidence in the number of ratings

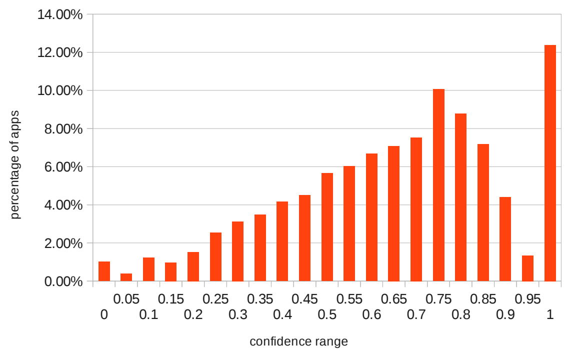
$\Delta m_1^2 = c_{\overline{rating}}(R)$ the confidence in average rating considering the distribution of ratings

$\Delta m_2^1 = \frac{1}{|reviews|} \sum_{T \in reviews} c_{spelling}(T)$ the average confidence in the reviews considering their spelling

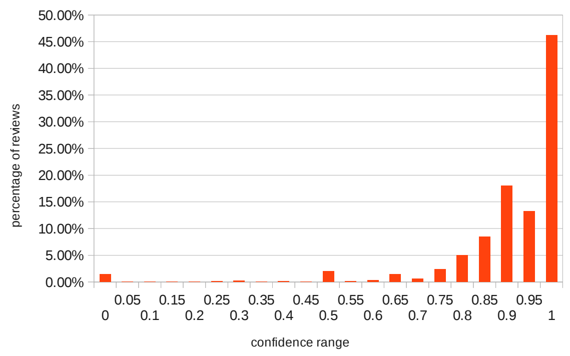
$\Delta m_2^2 = \frac{1}{|reviews|} \sum_{T \in reviews} c_{sentiment}(T)$ the average confidence in the reviews considering the difference between their rating and sentiment



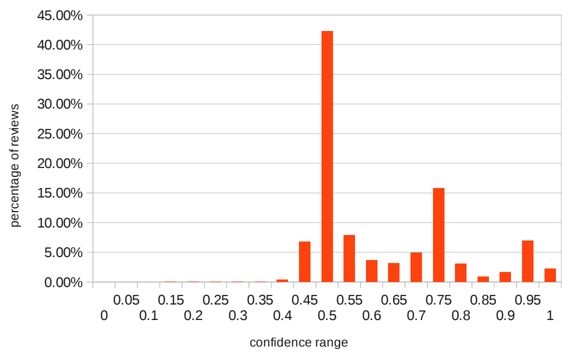
(a) Confidence in number of ratings, $\mu = 63.88\%$ and $\sigma = 37.54\%$



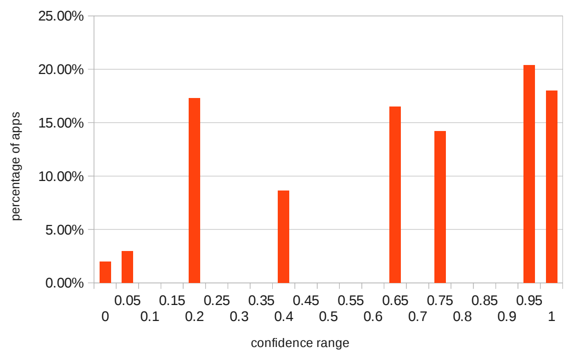
(b) Confidence in average rating considering distribution of ratings, $\mu = 67.46\%$ and $\sigma = 23.59\%$



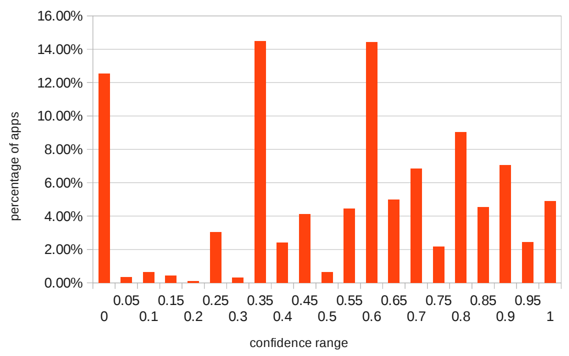
(c) Confidence in a review considering spelling, $\mu = 91.84\%$ and $\sigma = 15.86\%$



(d) Confidence in a review considering sentiment rating difference, $\mu = 63.54\%$ and $\sigma = 16.04\%$



(e) Confidence in number of dangerous permissions, $\mu = 68.13\%$ and $\sigma = 31.52\%$



(f) Confidence in set of dangerous permissions using 75% set similarity as 100% confidence, $\mu = 56.97\%$ and $\sigma = 29.81\%$

Figure 6.1.6: Trustworthiness assessments overview

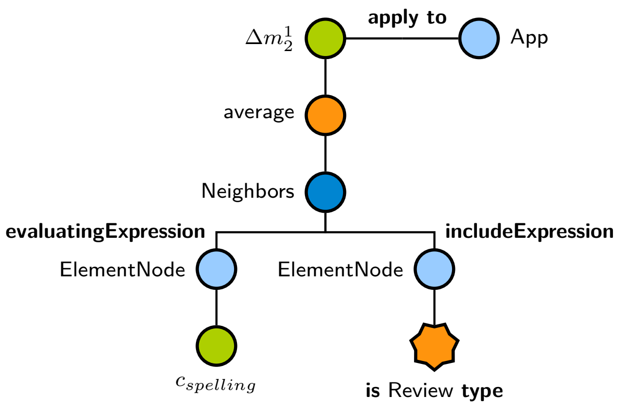
$\Delta m_3^1 = c_{\#permissions}(n)$ the confidence in the number of dangerous permissions

$\Delta m_3^2 = c_{permissions}(P_a)$ the confidence in the set of dangerous permissions

While some of the *metrics* have been discussed as part of the *knowledge processing* component the other ones can be defined using the following *expressions*.

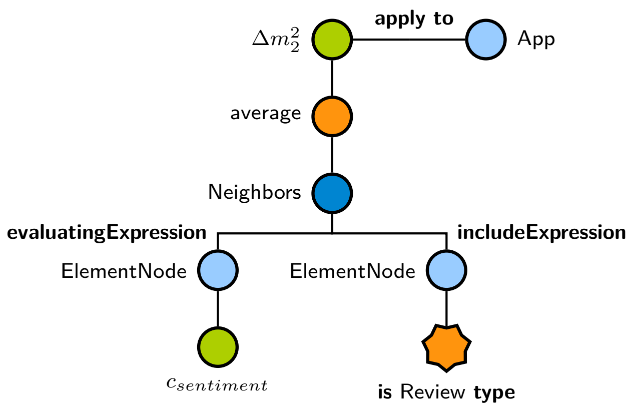
Expression 6.1.14 Δm_2^1

The average confidence in the spelling of reviews Δm_2^1 can be expressed using the *neighbors model expression* where we evaluate the $c_{spelling}$ *metric* on each *Review*.



Expression 6.1.15 Δm_2^2

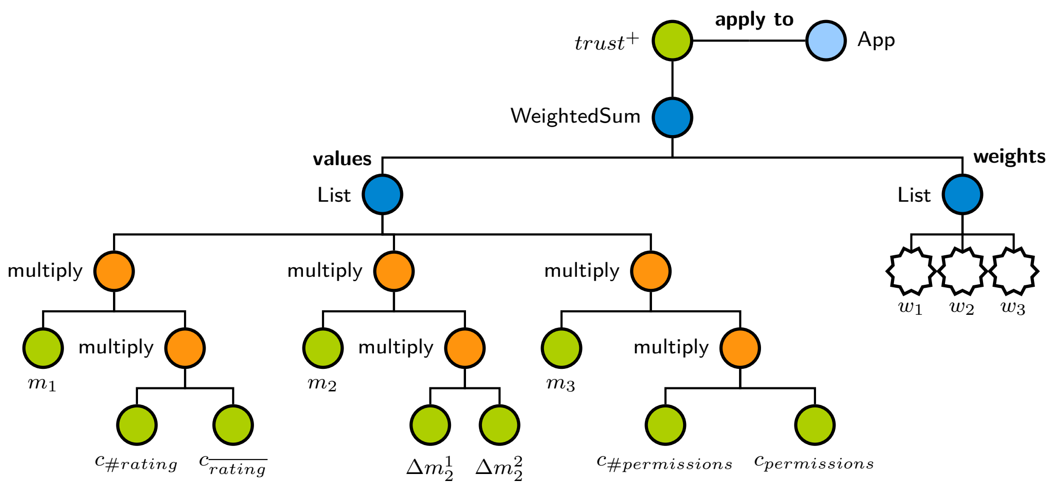
The average confidence in the sentiment of reviews Δm_2^2 is modeled in a similar fashion. Hence, the *neighbors model expression* is used to evaluate the $c_{sentiment}$ *metric* for each *Review* accordingly and the results subsequently averaged.

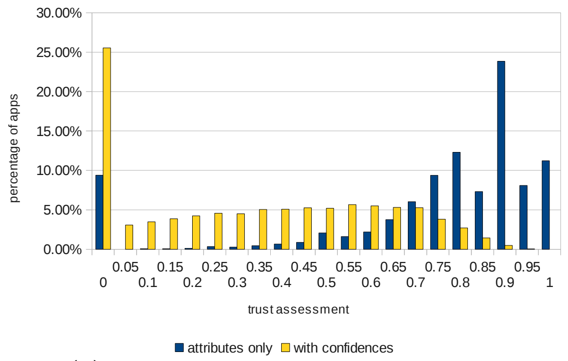


As a result the enhanced version of determining App trustworthiness which incorporates confidence assessments is defined as:

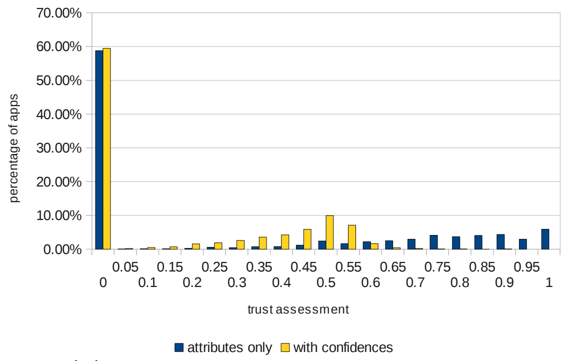
Expression 6.1.16 *trust*$^+$

The trustworthiness of an App considering factoring in confidence assessments for the basic attribute *metrics* scaled average rating, scaled average review rating, and scaled number of dangerous permissions is modeled using a *weighted sum model expression* on the previously defined *metrics* and the confidences as specified by equation 6.1.39. Here, we provide the ability to change the weighting scheme by adjusting the system parameters w_1, w_2 and w_3 as well.

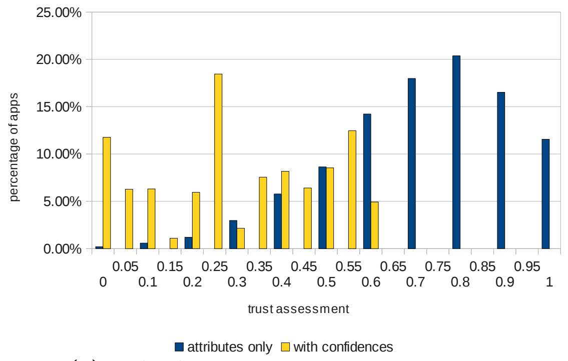




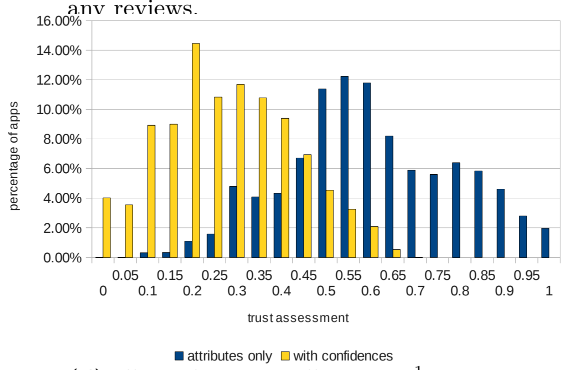
(a) only average app rating: $w_1 = 1, w_2 = 0, w_3 = 0$



(b) only average review ratings: $w_1 = 0, w_2 = 1, w_3 = 0$. Note that 58.52% of the Apps in the data set do not have any reviews.



(c) only dangerous permissions: $w_1 = 0, w_2 = 0, w_3 = 1$



(d) all attributes equally: $w_1 = \frac{1}{3}, w_2 = \frac{1}{3}, w_3 = \frac{1}{3}$

Figure 6.1.7: Comparison of App trustworthiness assessments between using only metrics (equation 6.1.38) and including confidence assessments (equation 6.1.39) for a variety of different weights.

The results for the Apps in the data set using different weights are shown in figure 6.1.7. It is notable that factoring in confidences results in a significant difference in the trust assessments. Whenever we include the confidences we generally lower the overall trust in an App. However, the overall trustworthiness distribution of the Apps with confidences seems far more reasonable than without. Especially for often artificially inflated ratings we notice a more evenly spaced distribution with a lot fewer "good" Apps (Figure 6.1.7a).

Furthermore, most of the distributions noticeably change which impacts the number of Apps above certain trust thresholds. For example, considering all attributes with equal weights (Figure 6.1.7d) without the confidence metrics about 5% of the Apps have a trust level above 90% but including the confidence metrics leads to the top 5% of Apps having trustworthiness assessments between 55% and 65%. This shows that there is no clear threshold available that could be used by a user to determine trustworthiness of Apps. However, the results describe overall trends and relative rankings which in combination with the individual confidence metrics allow the user to perform a more detailed analysis of an App.

The trustworthiness assessments shown in Figure 6.1.6 make it clear that individual attributes of Apps have various levels of trust and that this should be considered when using these attributes to determine trust. Hence, incorporating these assessments into the overall decision process using a weighted approach such as $trust^{+}$ (Equation 6.1.39) or more sophisticated methods is a necessary step towards improving the overall trust assessment of Apps as shown in Figure 6.1.7.

It is important to note that some App attributes as well as trustworthiness assessments may be better suited to the needs of one decision process than another. One also needs to consider the fact that some confidence metrics yield contrasting results for the same App attribute. This means that the ultimate decision on how to utilize these attributes and their confidences is up to the user. However, as part of this dissertation we proposed potential trustworthiness assessments and showed that it is necessary to incorporate them into the overall decision process because even though two Apps may have similar attribute values such as average ratings their "true" value may be far different.

6.1.6 Trusting Apps: Summary

Trust assessment of Apps is necessary and important since smartphones are becoming the new information hubs for people and companies but their security is generally lacking (rooting is common, malware and spyware widely circulated) such that there is no guarantee that information is safe. In addition, even App stores (Google Play, Apple App store) often contain unsafe Apps.

In this scenario we discussed the application of our formal but flexible framework to the domain of smartphone Apps. In particular, we modeled a heterogeneous system using the *abstract graph model* described in chapter 4. Furthermore, we proposed several metrics that can be utilized to determine confidence in App attributes such as average rating, average reviews rating and number of dangerous permissions and provided formal representations of them using *graph expressions* (section 4.4). Most importantly we showed that incorporating these confidence metrics using our approach described in chapter 5 is helpful in determining trustworthiness and ultimately whether to install an App or not. As such, we developed two *decision processes* and described them in detail using *graph expressions*. Furthermore, the entire scenario shows how flexible our framework since every computation and assessment is simply based on evaluating formal *graph expressions* on formal *graph components*.

Future research will focus on evaluating the quality of the recommendation based on the proposed techniques and refining the decision engines. For instance, one could easily extend the current decision process to only consider Apps as trustworthy where no confidence metric falls below a certain threshold.

Furthermore, even though we proposed trustworthiness assessments that take into consideration relationships between Apps we need to investigate assessments of Apps at various levels:

local looking only at the attributes of a single App without considering its relation to other Apps (rating but not compared to average rating of all Apps, etc.)

similar comparing Apps with “similar” attributes (similar rating, similar number of reviews, etc.)

related comparing Apps that share relationships among each other (same developer, also installed, also viewed, etc.)

category comparing Apps in the same category as they should have similar attributes but may be unrelated and hence potentially leads to Apps forming clusters

global comparing a single App to all other Apps

external meta data using information from outside the primary source (i.e., Google Play) and correlating it with the existing Apps attributes and relationships.

By using the *TrustKnowOne* framework described here we provide an approach based on *graph expressions* that allows existing as well as future *belief engines* and *decision processes* to be implemented. In particular, these *graph expressions* can be associated with metrics and cost assessments thereby enabling approaches to be evaluated and compared formally.

6.2 Radiation Detection in Heterogeneous Sensor Networks

Here a radiation detection scenario will be used to illustrate the kind of data and processes our *TrustKnowOne* framework will be able to consider. Imagine the following; a set of sensors are available to measure levels of radiation. These sensors could be privately owned (connected to or part of a smart phone) or part of a government sensor network. The goal is to use geographically distributed radiation readings from a set of heterogeneous sensors to decide if the environment is safe. In general, the groups of radiation detectors shown in table 6.2.1 may be present.

Note that cost could be an indicator of the accuracy and capabilities of the radiation sensors; more expensive sensors could not only detect the presence but the type of

Table 6.2.1: Radiation sensor groups

Type	Trustworthiness	Security	Cost
public	not verified	not signed	low
non-government/private	varying	varying	varying
government	verified	signed	high or medium

radiation, e.g., Alpha, Beta and Gamma radiation. These groups possess sensors of varying cost, accuracy and trust which means the resulting observations need to have a differentiated influence on the decision making process of telling whether or not there is radiation present.

Furthermore, the entire set of heterogeneous sensors forms various physical (geographically close) and logical (e.g., same owner, same class of sensor, or same age of sensor) relationships that we are able to utilize in our framework to determine the accuracy of the radiation detection and the trust we have in specific sensors. In the decision making process we may want to trust readings from government sources more than from public or private sensors because they are signed and verified. In addition, mobile sensors move and their readings from previous locations are not necessarily accurate anymore but we could still use the data to a certain degree when determining radiation, thus making the measurements also time varying.

The use of a network of geographically distributed heterogeneous sensors combined with the our framework could have proved useful to detect the radiation levels at Japan's earthquake-stricken Fukushima nuclear power plant and contributed to decisions concerning the safety of the surrounding environment. We chose this scenario to showcase how our framework is able to model the heterogeneous system of sensors and its complex relationships necessary to improve the knowledge derivation process for radiation detection.

6.2.1 Radiation Detection: Overview

Radiation detection is a natural application of sensor networks. In this section, we discuss how our framework can be applied to what is referred to as *collaborative sensing*. Sensor networks and in particular radiation detection networks used to be expensive to set up, difficult to maintain and often under the authority of a government agency. However, the cost of sensors has decreased to the point where consumers are able to purchase them. Furthermore, the capabilities of sensors have improved dramatically. This means that both application-specific sensors (e.g., for radiation detection) as well as multi-purpose sensors (e.g., GPS, temperature, and humidity combined) can now be utilized more efficiently (i.e., using one multi-purpose sensor instead of multiple specific ones) and effectively (i.e., higher accuracy with lower cost).

Nevertheless, several problems remain to be solved. For instance, the availability of a various sensor types is prone to create heterogeneous environments in which data integration and fusion become necessary preprocessing steps before data can be analyzed. Specifically for the radiation detection there are several different types of detectors for individual radiation sources (e.g., cosmic, terrestrial, nuclear), elements (e.g., Caesium, Plutonium, Uranium), and emissions (e.g., alpha, beta, gamma rays). Furthermore, detecting radiation is sensitive to several factors such as location (e.g., inside buildings, open field) as well as measurement inaccuracies (i.e., ground level, at 1 meter height, sensor calibration).

These physical aspects then need to be considered with respect to logical context such as ownership and trustworthiness, all of which makes radiation detection one of the most complex sensor network application scenarios. In particular, during the Fukushima nuclear incident in 2011 Safecast [144] reported that there were problems with communicating radiation level measurements in a timely manner because of bureaucracy and political pressure. When data was released it was often incomplete or inaccurate. While most of the data was later adjusted after public protest this led to a decrease in trust of official government authorities. Hence, activists turned to collaborative sensing approaches in order to alleviate these issues. In this scenario we will discuss how our framework can be applied to model knowledge derivation that incorporates trust and quality of data in this kind of heterogeneous sensor network environment.

6.2.2 Radiation Detection: Framework Modeling Approach

Our *TrustKnowOne* framework is able to model heterogeneous systems as well as deal with the dynamic environment often found in sensor networks. Furthermore, many of the problems described can be solved through the various aspects of our framework. In particular, the *abstract graph model* allows us to model different types of sensors and define relationships between them. Additionally, *dimension models* can be used to express notions of the accuracy and constraints of sensors (see section 4.3). As trust becomes an even more important component of being able to derive knowledge from the vast amount of data provided by sensor networks, our framework can describe trust and quality assessments of data in a uniform and flexible manner using *graph expressions*.

As a case study we will focus on *collaborative sensing* in the context of Japan's Fukushima nuclear incident in 2011. The incident caused the establishment of Safecast [144] where users can submit radiation measurements they have taken. Furthermore, there exists an official government database of radiation measurements by the International Atomic Energy Agency [75]. Here, we analyze the combined data sets from both sources and discuss approaches to assess the confidence in individual measurements as well as the trustworthiness of sensors and their sources.

In the following sections we will discuss how the *TrustKnowOne* framework is able to model data processing from multiple source with varying trust aspects and their analysis. In particular, we will address:

- Modeling heterogeneous data (Safecast measurements, IAEA sensors), relationships (location clusters, id clusters), and data sources (collaborative, government)

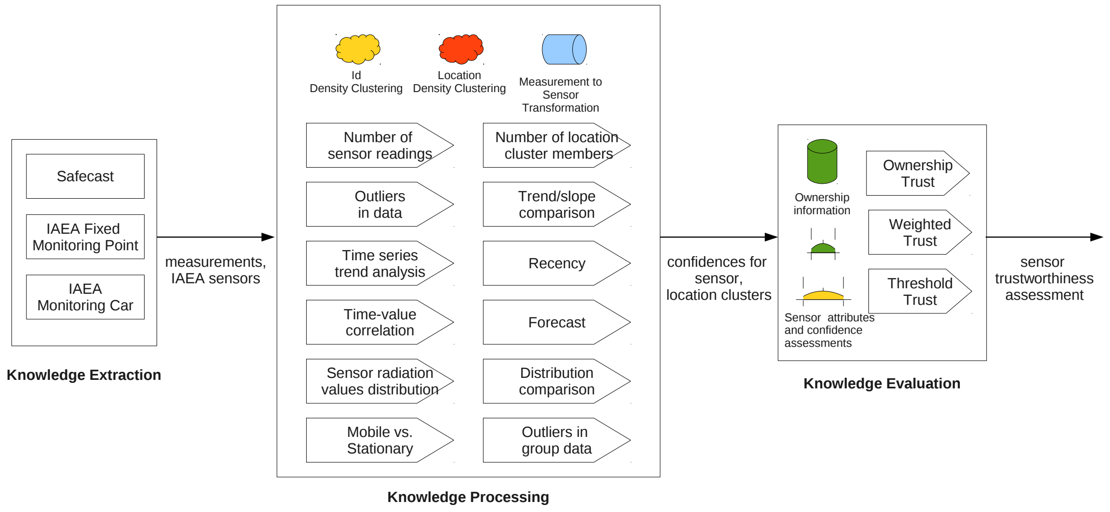

Safecast
IAEA Fixed Monitoring Point
IAEA Monitoring Car
Knowledge Extraction
measurements, IAEA sensors
Id Density Clustering
Location Density Clustering
Measurement to Sensor Transformation
Number of sensor readings
Number of location cluster members
Outliers in data
Trend/slope comparison
Time series trend analysis
Recency
Time-value correlation
Forecast
Sensor radiation values distribution
Distribution comparison
Mobile vs. Stationary
Outliers in group data
Knowledge Processing
confidences for sensor, location clusters
Ownership information
Sensor attributes and confidence assessments
Ownership Trust
Weighted Trust
Threshold Trust
Knowledge Evaluation
sensor trustworthiness assessment

Figure 6.2.1: Framework overview for the radiation detection scenario

in our *abstract graph model* (section 4.1)

- Incorporating location dynamics (mobile vs. stationary sensors) and time variant information (sensor readings are time series data) throughout *knowledge processing* and *evaluation* (section 5.3.1, 5.4.1)
- Performing complex data transformations (conversion of attribute values, combination of elements, creation of elements) using *graph expressions* (section 4.4)
- Formalizing confidence assessments for sensors using time series information, context, and group relationships as *belief engines* (section 5.3.1)
- Representing decision making as formal *decision processes* with the option of whether or not to incorporate confidence assessments (section 5.4.1)

Figure 6.2.1 shows an overview of how the individual components discussed in this section relate to our *TrustKnowOne* framework.

Knowledge Extraction We model two types of sensors, one representing coming from the Safecast [144] the other from the IAEA [75] data set. Note that Safecast [144] only provides single measurements which makes it difficult to perform extended trust assessments that incorporate relationships between data. We retrieved the data using the their web application interface for the entire year of 2011. The data from IAEA [75] is based on several sensors which makes correlation is possible. However, while the number of radiation measurements per sensor is high there are only a few sensors available. As a result of the framework is able to utilize three static data sources that provide sensor and radiation level information provided with time and location stamps.

Knowledge Processing In order to perform proper processing and trust assessment we need to transform the individual measurements from Safecast [144] into sensors. Our approach incorporates two density-based clustering procedures where we exploit location and id sequence properties of measurements. Confidence assessments are provided in two ways. First, each sensor is evaluated based on its attributes individually. Second, we put the sensor into context by examining its relationship with sensors in the same location (cluster). Hence, a variety of trust and quality assessments is provided that allows the detailed evaluation of sensors.

Knowledge Evaluation Based on the individual as well as the location cluster assessments of a sensor we are able to evaluate its trustworthiness. We discuss three approaches that do not depend on choosing particular confidence assessments over others but rather combine them in a variety of ways. First, all assessments are incorporated which allows absolute (e.g., 0%-100% scale) as well as relative (e.g., top 10% percentile) ranking. Second, we allow the user to pick thresholds to determine trustworthiness levels. Third, based on ownership we incorporate confidences differently into the decisions.

The next sections will provide a detailed analysis of the scenario aspects and their relationship to our framework.

6.2.3 Radiation Detection: Knowledge Extraction

We propose the following approach to modeling the collaborative radiation detection scenario described. The key components of the *abstract graph model* are various types of sensors. Hence, we model each type as an independent *element*. The two data sources that we consider are user submitted measurements from Safecast [144] and official measurements from the International Atomic Energy Agency (IAEA) [75]. As such we define *elements* with *attributes* that match their properties.

In addition, we provide a *Location element* that is relevant because it is used to relate individual *Sensor elements* to each other. Specifically, *Sensor elements* represent derivations from multiple *Measurement elements* through means of clustering. Since *IAEASensor elements* already provide time series information they can be modeled as is. During this process the *Sensor elements* will be automatically assigned to a particular *Location* whereas *IAEASensor elements* are assigned to a single *Location*. We model relationships accordingly for each of the sensor types and a particular *Location*. The *abstract graph model* for this scenario is shown in figure 6.2.2.

An example of how the *element instance graph* might look like is shown in figure 6.2.3. In particular, it displays how three *Measurements* are transformed into a *Sensor* that incorporates their individual values. Note that we left out a number of attributes for clarity in the figure.

After the density-based clustering process is performed during the *knowledge processing* stage, only *Sensor* and *IAEASensor element nodes* are utilized for determining trustworthiness aspects. In figure 6.2.4 we show two *Sensor element nodes* as well as two *IAEASensor element nodes*. Each is related to a single *Location element node* using the *Location-Sensor* and the *Location-IAEASensor relation edges* respectively.

An overview of the data sets we use as part of this scenario is given in table 6.2.2. We chose the data sets based on their diversity. Whereas Safecast [144] contains thousands of collaboratively collected measurements which are loosely connected, the IAEA [75]

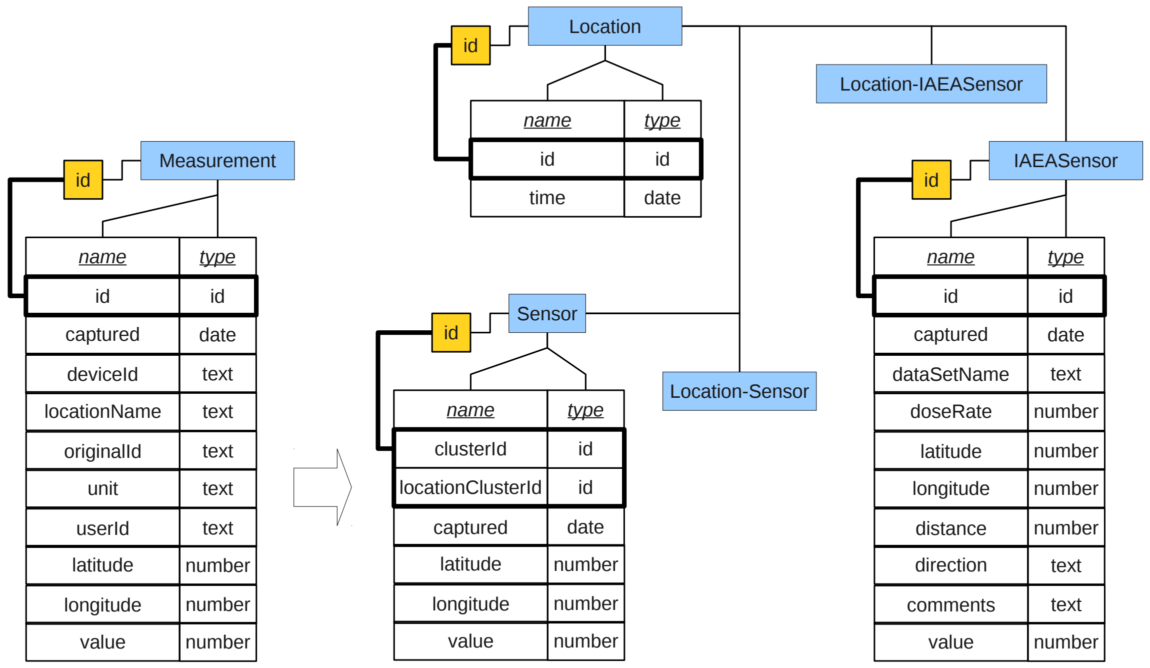


Figure 6.2.2: Radiation detection *element description graph*

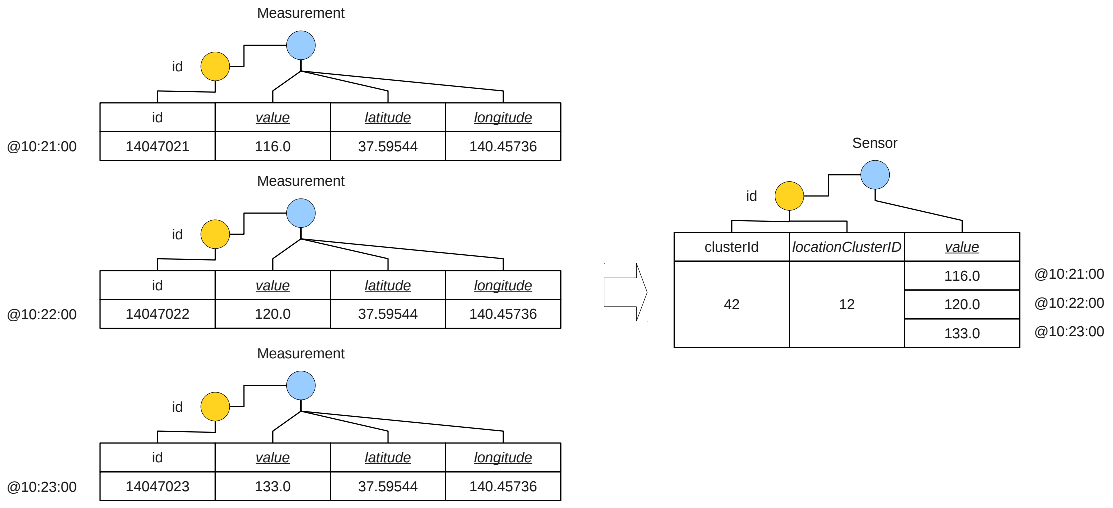


Figure 6.2.3: Radiation detection clustering *element instance graph*

data sets comprise of official government sensors and their radiation levels. As such we are able to apply our framework in a heterogeneous environment of sensors with radiation levels captured as time series and context such as location and ownership

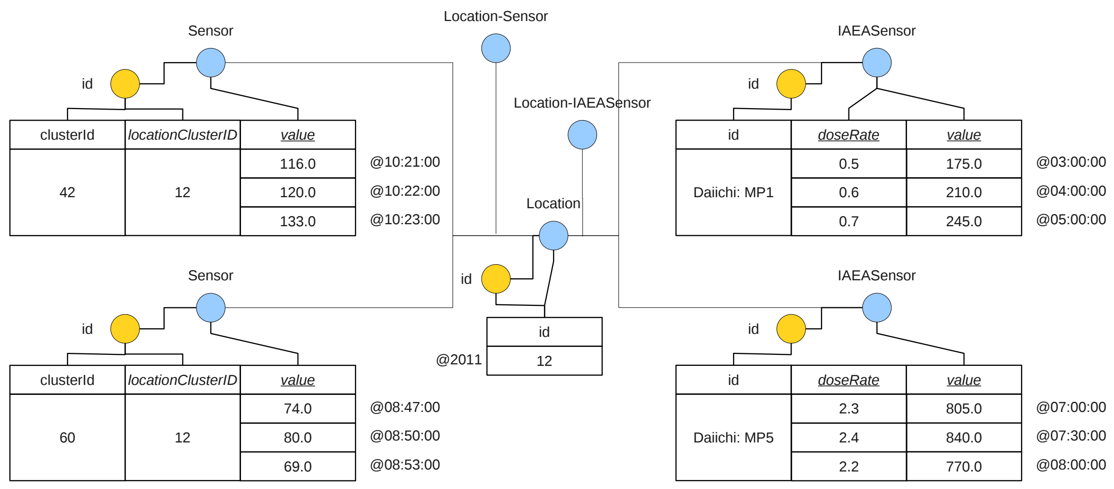


Figure 6.2.4: Radiation detection *element instance graph*

Table 6.2.2: Radiation detection data set overview

Data Set	Data Points	Sensors	Time Frame	Type
Safecast [144]	2,120,078	≈ 40,000	2011-04-23–2011-12-31	mobile and stationary
Fukushima Daiichi: Fixed Post gamma dose rates [75]	311,720	8	2011-04-05–2011-12-31	stationary
Fukushima Daiichi: Monitoring Car gamma dose rates [75]	41,989	12	2011-03-14–2011-12-31	stationary

can be incorporated into the decision process. Note that while the IAEA monitoring car data set name implies mobile sensors the measurements actually reflect those of transportable sensors. That is the sensor was transported by car to a specific location, set up, and measurements taken. Hence, we will refer to the data set transportable sensors.

6.2.4 Radiation Detection: Knowledge Processing

In order to assess the trustworthiness of individual measurements, sensors, and data sources we define *metrics* that reflect various aspects of confidence in *element node* properties. In general, there are two categories of approaches that we will discuss. First, sensor *attributes* and *metrics* computed from these *attributes* can be used to evaluate sensors on an individual basis. Second, sensors form various relationships (e.g., sensors of the same location cluster) that can be exploited to find outliers and anomalies.

However, before we can apply these *metrics* we need to discuss the how the data provided by the Safecast [144] can be incorporated into our framework.

6.2.4.1 Deriving Sensor Identities Through Density-based Clustering

We retrieved the *collaborative sensing* data set from Safecast [144] using their web application programming interface. The data consists of individual measurements that were taken and submitted by a variety of users. Specifically, we have information about time and location of the measurement as well as its radiation value. However, there are several problems with the data initially provided by Safecast [144]. Some attributes such as “device id”, “location name”, and “original id” are not set in about 99% of the cases. Furthermore, the usefulness of other attributes is limited. In particular, all data from 2011 has the “cpm” which stands for counts per minute and the “user id" is 1 for about 99% of the measurements. This makes it hard to identify measurements that were taken by the same sensor or user and model relations between measurements beyond the basic time (measurements at a similar time) and location (measurements in a similar location) domains. As such, one of the problems is to transform the “raw” measurements from the Safecast [144] data set into measurements we can infer came from specific sensors.

In order to address these problems we employed the following approach. First, we performed density-based location clustering on the measurements to group measure-

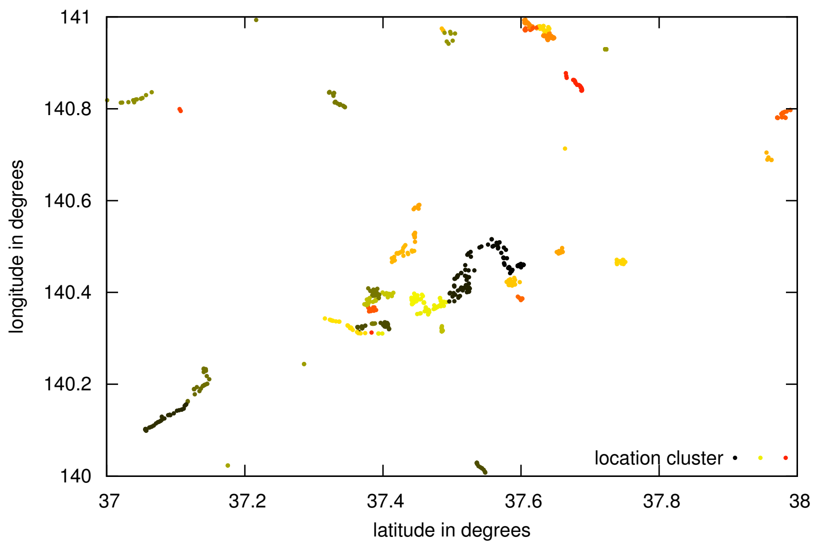


Figure 6.2.5: Various location clusters within a specified latitude and longitude identified by color

ments in the same vicinity of each other to infer the identity of specific sensors. The reason we used a density-based approach here over distance-based clustering lies in the fact that we want to model mobile as well as stationary sensors. Distance-based clustering would have worked well on stationary sensors but would have likely split measurements from the same sensor if it was moving along a path. As shown in figure 6.2.5 stationary sensors still appear as individual clusters forming the expected circle like coloring but mobile sensors are identified as well forming paths of the same color.

Second, within individual density-based location clusters we inferred the identity of individual sensors by utilizing a property of the Safecast [144] data set. Most measurements were uploaded in sequence and thus have consecutive ids. This fact was discovered as many sequential measurements had similar location attributes while being specific time intervals (e.g., 5 seconds, 10 seconds) apart. Therefore, we performed a second clustering based on series of consecutive ids. figure 6.2.6 shows one of the density-based

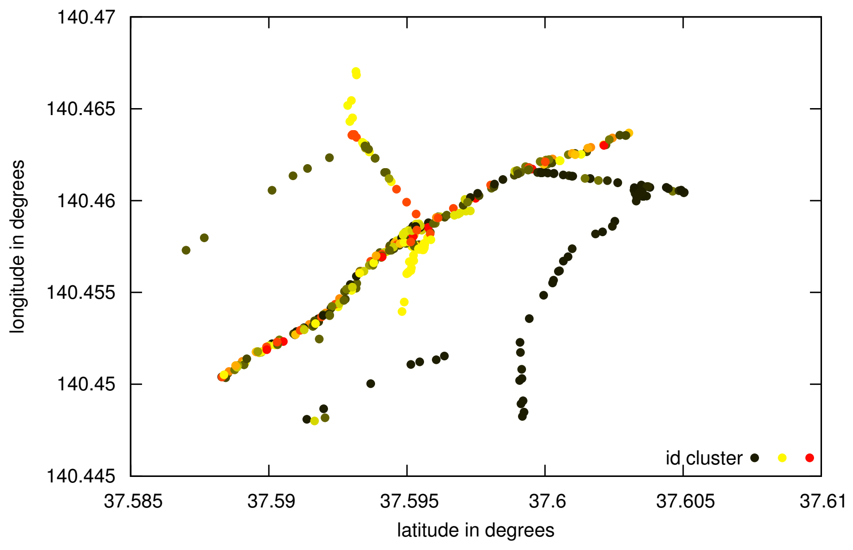


Figure 6.2.6: Several id clusters within an individual location cluster identified by color

location clusters where consecutive id clustering has been used to identify clusters. We can clearly see that many sensors move along paths meaning they are mobile. This validates our density-based approach to infer sensor ids to a certain degree.

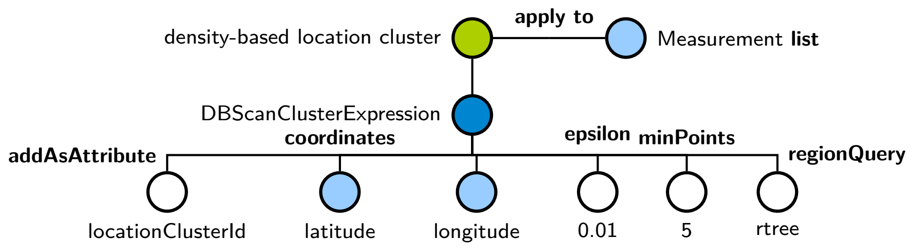
Note that we used this two step approach to avoid the case were measurements were identified as belonging to the same sensor simply because of sequential ids. However, we need to point out that we cannot identify cases in which a sensor was used to gather measurements at one location or path and than later elsewhere. In our approach we would treat this as two different sensors. Next, we will discuss the clustering approaches in detail and describe how we integrated them into our framework using flexible graph *expressions*.

The first step is to cluster based on location. As described above we make use of a density-based approach. In particular, we chose DBSCAN by Ester et al. [46]. The basic idea behind the algorithm is as follows. We iterate over the set of data points where we determine neighbors using a *region query*. This *region query* performs a neighborhood search and returns all neighbors within a specified distance ϵ. If the number of neighbors exceeds a minimum number of points it is considered a new cluster. All neighbors that do not already belong to a cluster become members of the new cluster. Furthermore, these neighbors are used as starting points for expanding the cluster. This means that we determine their neighbors within distance ϵ and continue recursively.

Using this approach we are able to deal with measurements that were captured by mobile sensors since the density-based technique will be able to identify path clusters whereas a distance-based approach will not. The key component of the algorithm is the *region query*. It represents the distance calculation function and needs to be implement in a scalable manner in order to avoid the $O(n^2)$ complexity of pair-wise data point distance computations. There are several spatial indices available. Here, we employ *R-trees* developed by Guttman [66] since they give good performance ($O(\log n)$) for nearest neighbor searches (see Brinkhoff et al. [21]).

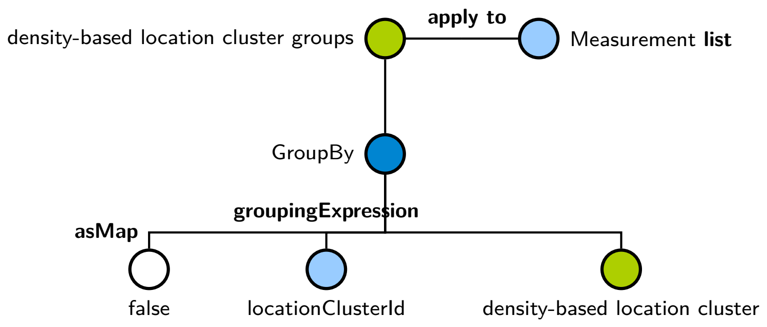
Expression 6.2.1 density-based location cluster

We model the density-based location clustering approach as a *DBScan cluster model expression*. In particular, we choose the minimum number of measurements to form a cluster to be 5 and the neighborhood search radius (epsilon) to be 0.01 which since the calculation is based on the Euclidean distance of GPS coordinates is about 1100 meters. As a spatial index we use an *R-tree* for the coordinates that are stored in the *latitude* and *longitude attributes*. We also add the cluster id as a new *location cluster id attribute*.



Expression 6.2.2 density-based location cluster groups

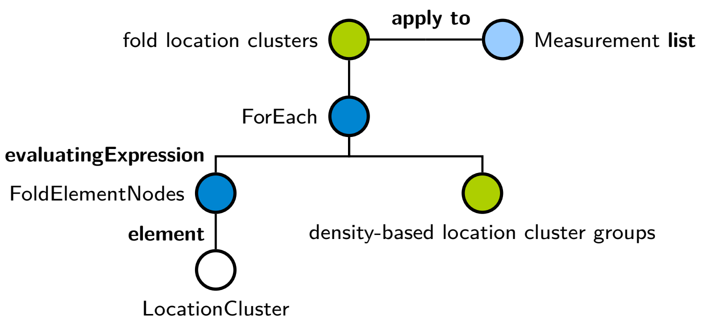
In order to group the Safecast measurements by their *location cluster id* we use a *model expression* called *group by.* Note that we have to options for the result of this expression. First, as a list of groups, such as $\{\{m_1, m_2\}, \{m_3, m_6\}, ...\}$ where m_i is a particular measurement *element node* or as a map $\{l_1 \rightarrow m_1, m_2\}, l_2 \rightarrow \{m_3, m_6\}, ...\}$ where l_j is a particular location cluster id.



The result of the *density-based location cluster groups metric* is a list of clusters that follow the density-based location clustering approach where each element in the group is a measurement with the additional *location cluster id attribute.* One option is to stop here and treat all measurements in the same group as coming from a single sensor.

Expression 6.2.3 fold location clusters

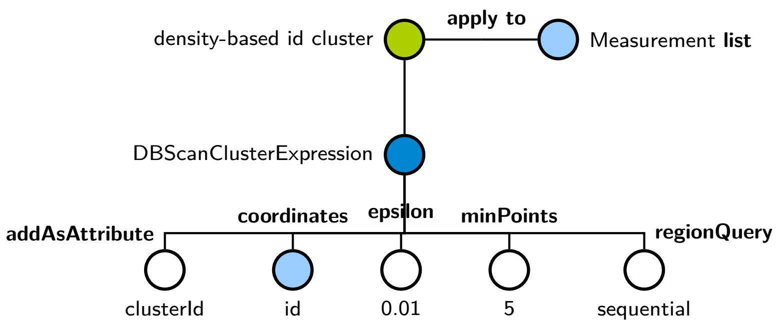
We can transform individual *element nodes* into new *element nodes* specified by the *element* parameter using the *fold model expression.* Here, we apply this transformation to each of the location cluster groups using the *for each model expression.* Thus the result of the *fold location clusters metric* would be *location cluster element nodes* containing the combined time series information of all measurements in a particular location cluster.



In our case, as explained above, we proceed and perform additional clustering based on the fact that measurements with consecutive ids are likely to come from the same sensor. We can utilize the same density-based clustering approach DBSCAN. The one thing we need to modify is the *region query*. Here, we describe our *sequential* neighborhood search technique. For each data point we return the neighbors by searching the list of measurements for ids before and after the one of the specified measurement until there is a break. For example, given the set of measurements $\{m_1, m_2, m_3, m_7, m_8, m_9, m_{10}, m_{11}, m_{15}, m_{16}, ...\}$ where i of m_i is the id, the *sequential region query* would return the neighbors $\{m_7, m_9, m_{10}, m_{11}\}$ for m_8.

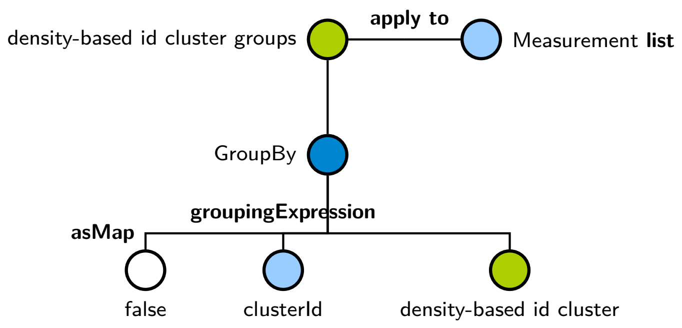
Expression 6.2.4 density-based id cluster

We model the *density-based id cluster metric* by using the *DBScan model expression* again. Here, we specify to perform the neighborhood search based on the *sequential* id cluster described above. We set the minimum number of point for a cluster to 5. Note that the epsilon that is part of the DBSCAN algorithm is ignored here because of the way our *sequential region query* works. The clustering is performed on the *measurement id attribute* and we add the resulting id cluster as the *cluster id attribute.*



Expression 6.2.5 density-based id cluster groups

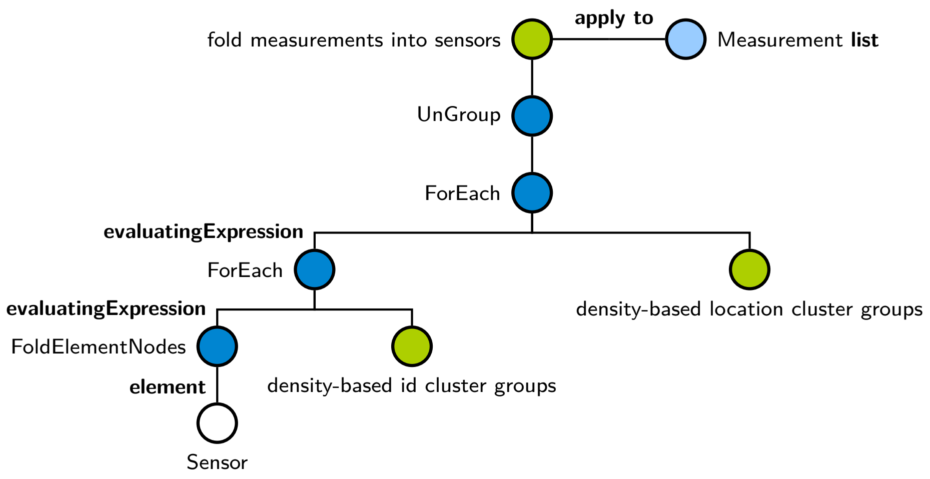
We can model the cluster of sensors with consecutive ids by using the *group by model expression* on the *cluster id attribute*. As a result we specify a list of cluster id groups.



Now that we discussed both, the density-based location and the density-based id clustering we apply them in order to derive *sensor element nodes*. Thus we can transform the “raw” measurements from the Safecast [144] data set into sensors as follows.

Expression 6.2.6 fold measurements into sensors

The transformation of measurements into sensors consists of several steps. First, we cluster by location using the *density-based location cluster group metric*. Second, using the *for each model expression* we cluster within each of the location cluster groups by id employing the *density-based id cluster groups metric*. Third in order to derive *sensor element nodes* we apply the *fold nodes model expression* to each of the id cluster groups with the location cluster groups. This results in a list of location cluster groups containing individual *sensor element nodes*. Finally, we merge all the groups into one list of clustered *sensors* using the *ungroup model expression*.



In order to allow quick and straightforward management of sensors within a location cluster we can utilize the fact that each sensor maintains the *location cluster id attribute*. Hence, after the clustering process we apply the following *metric* to all *sensor element nodes* to create *location element nodes* and *relation edges* to them accordingly.

Expression 6.2.7 create location group nodes

The *location element nodes* can be derived by applying the *create group node mode expression* to the groups of sensors provided by the *group by model expression*. Here, the group node being created is of type *location* and uses the *location cluster id* of the group as its *id attribute*. Note that, we also implicitly create the appropriate *relation edges* of the specified *location-sensor* type.

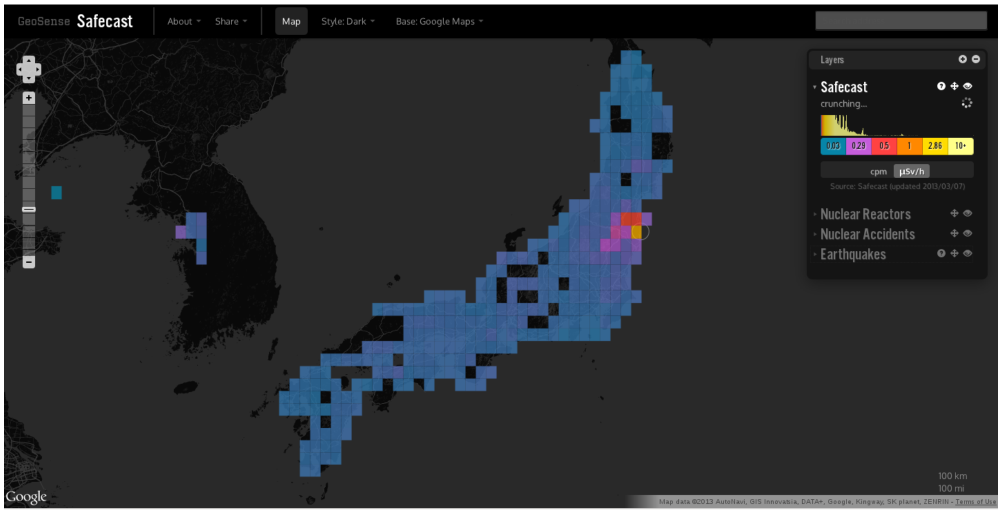
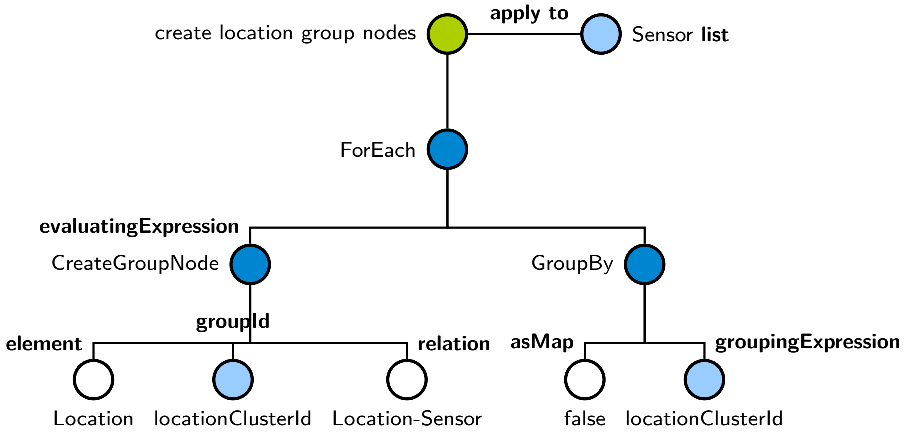


Figure 6.2.7: Safecast [144] heatmap of user submitted radiation measurements for Japan



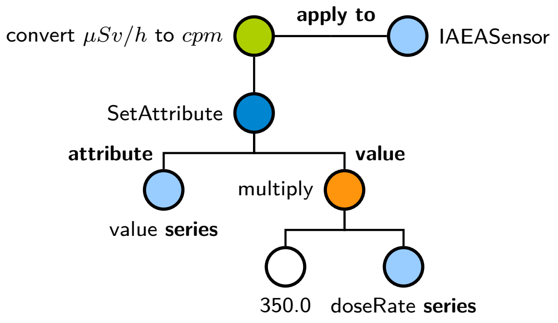
Since we have various data sources one thing we need to keep in mind is the unit of the detected radiation values. While Safecast [144] uses counts per minute (*cpm*) the International Atomic Energy Agency [75] uses $\mu Sv/h$. We employ the following conversion scheme as provided by Safecast [144] where $1\mu Sievert/hour = 350cpm$. Note that Safecast [144] also provides a color coded chart of radiation severity which is shown in table 6.2.3 and utilized to display the heatmap in figure 6.2.7.

Expression 6.2.8 convert $\mu Sv/h$ to *cpm*

Table 6.2.3: Radiation severity according to Safecast [144]

Severity	Color	*cpm*	$\mu Sv/h$
very low	blue	10	0.03
low	purple	100	0.29
medium	red	175	0.50
severe	orange	350	1.00
very severe	yellow	1000	2.86
most severe	bright yellow	>3500	>10

We can change *attribute* values using the *set attribute model expression*. Here we convert the *dose rate* time series given in $\mu Sv/h$ into *cpm* and set it as the *value* time series.



6.2.4.2 Individual Sensors

On the individual sensor level, we propose *metrics* which are based on the time series of detected radiation measurements. Hence, some of the *metrics* we discuss here affect each other. For example, the number of time series values has a definitive correlation with trend analysis as more values increase our ability to model trends with higher confidence.

Number of Sensor Readings It is important to evaluate the number of sensor measurements. This is particular true for time series analysis.

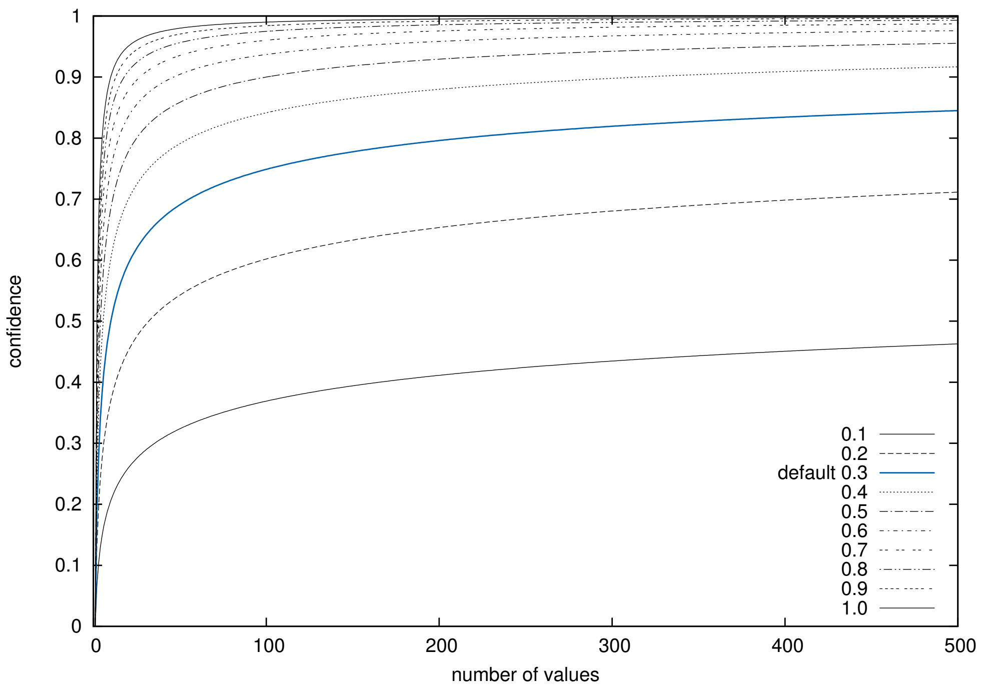
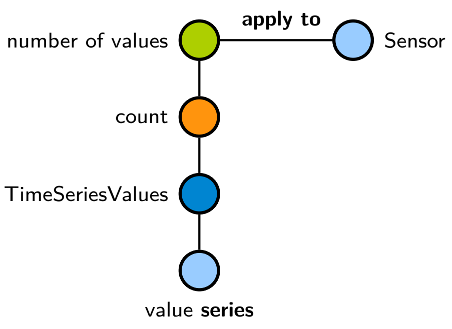


Figure 6.2.8: The confidence based on the number of ratings $c_{\#values}$ modeled using the Pareto cumulative distribution function with $x_{min} = 1$ and α ranging from 0.1 to 1.0

Expression 6.2.9 number of values

The number of detected radiation values of a sensor can be determined by a basic count of the number of time series values.[2]



Here we utilize the form of the cumulative distribution function of the Pareto dis-

[2]Note that in general all individual and location cluster sensor *metrics* can be applied to *Sensor* and *IAEASensor element nodes.*

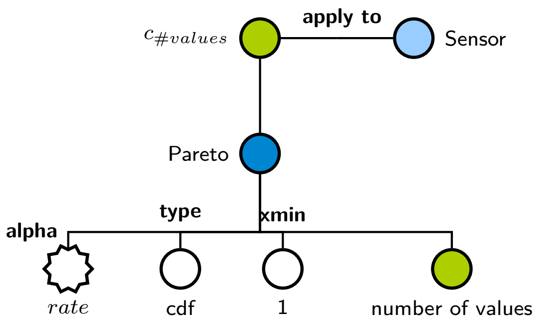
tribution to express the confidence in the number of values.

$$Pareto_{cdf}(x) = \begin{cases} 1 - \left(\frac{x_{min}}{x}\right)^{\alpha} & \text{if } x \geq x_{min} \\ 0 & \text{if } x < x_{min} \end{cases} \tag{6.2.1}$$

where x is the number of values, x_{min} is the initial value for the function before which the probability is 0 and α is the rate parameter. We choose this function because it follows a power law and after an initial phase of low number of values it quickly approaches 1 (see figure 6.2.8).

Expression 6.2.10 $c_{\#values}$

The $c_{\#values}$ *metric* is expressed using a *Pareto model expression* with an initial value x_{min} and a rate parameter *alpha*. As the result type we specify the cumulative distribution function (*cdf*).



Outliers in Data Variance in data and specifically data points that do not align with a general trend should make us question whether an overall time series of data is accurate. There exist several techniques for detecting outliers in multidimensional data. However, here we focus on time series data. As such, we utilize Cook's distance [35] for finding "influential" data points. The basic idea of the approach is that we perform a regression on the available data and compare the sum of the squared residuals of the regression including all data points with one that leaves one data point out. Hence, the distance for an individual data point i can be defined as

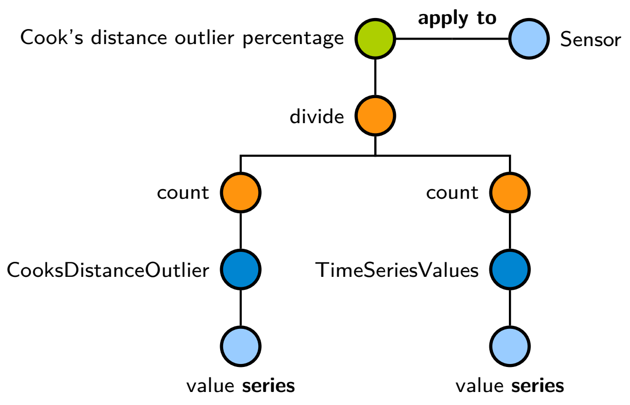
$$distance_i = \frac{\sum_{j;j\neq i}^{n} \hat{Y}_{all}(j) - \hat{Y}_{without\ i}(j)}{p \times MSE(\hat{Y_{all}})} \tag{6.2.2}$$

where $\hat{Y}_{all}(j)$ is the predicted value for j using the regression model for all data points, $\hat{Y}_{without\ i}(j)$ is the predicted value for j using the regression model without i, p the number of regression parameters, and $MSE(\hat{Y_{all}})$ the mean squared error of the original regression model. Note that, for example, the number of parameters p is two for a linear regression (i.e., slope and intercept) and $n+1$ for an nth order polynomial regression (i.e., the factors).

In order to determine the "influential" data points or outliers Cook originally suggested $distance_i > 1$. However, we choose $distance_i > \frac{4}{n}$ where n is the number of data points because this criterion has been identified by Bollen and Jackman [18] as being better suited.

Expression 6.2.11 Cook's distance outlier percentage

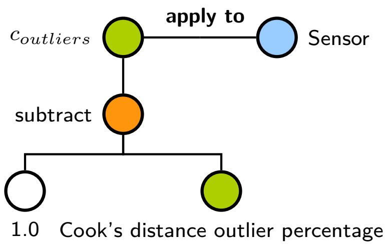
We express outliers in data as the percentage of outliers according to Cook's distance for a given time series of detected radiation values. The *Cook's distance outlier model expression* yields a list of all data points that have a $distance_i > \frac{4}{n}$. The requested percentage thus becomes the ratio of number of Cook's outliers over the number of values in the time series.



Our confidence in a time series of values considering outliers follows the intuition that the more outliers there are the less confidence we should have in the time series being an accurate representation of its true values.

Expression 6.2.12 $c_{outliers}$

The $c_{outliers}$ *metric* is modeled as being anti-proportional to the *Cook's distance outlier percentage metric.* This reflects the fact that more outliers mean less confidence and less outliers mean higher confidence.



Time Series Trend Analysis Trend analysis is key in determining whether a time series follows a specific pattern or not. In the case of the radiation detection we would like to see either constant or slowly declining radiation values, both can be detected by analyzing the slope of a linear regression.

However, there are several issues that need to be addressed. First, the slope is directly related to the time scale begin used. This means that we need to be careful to use the same time scale (e.g., milliseconds, seconds, minutes) across comparisons. Second, if the time scale becomes to small (i.e., milliseconds) and the distance between data points of the time series large (i.e., weeks or months) there are computational challenges in terms of dealing with floating points. Third, there exist a variety of approaches to estimating the slope such as

- treating the first time series as the starting point with time $t_0 = 0$ and estimating the slope relative to that time
- ignoring time intervals between data points completely and thus treating time as

a discrete sequence $t_0, t_1...t_n$

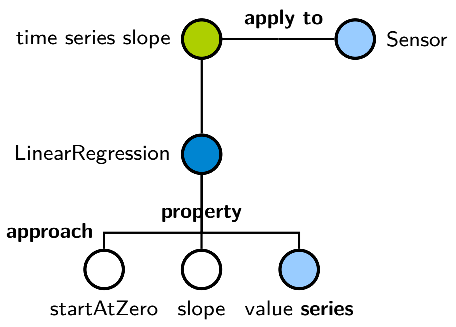
- defining a generic start time accross all time series that are being compared and calculating the slope relative to that time

Finally, one needs to consider the fact that outliers and anomalies often affect statistical trend analysis. This gives rise to robust estimators such as the Theil-Sen estimator (see Fernandes and G. Leblanc [50]).

Our approach addresses some of these problems while deferring others. For instance, we perform a linear regression on the time series data using the first data point as time $t_0 = 0$. Furthermore, by default we use milliseconds as a time unit but also provide the means to include a time scaling factor. The question of robustness against outliers is avoided by having a separate confidence *metric* such as $c_{outliers}$ focus on that aspect.

Expression 6.2.13 time series slope

The slope of a time series is determined using a *linear regression model expression.* Note that we treat the first data point of the time series as having time $t_0 = 0$ and determine the slope relative to that time.



As described above, for the radiation detection the slope of radiation values should be almost constant or slightly negative. In order to determine the confidence in the time series trend we utilize a ratio of Normal distribution probability densities such that

$$c_{slope} = \frac{N(s|0, \sigma^2)}{N(0|0, \sigma^2)} \tag{6.2.3}$$

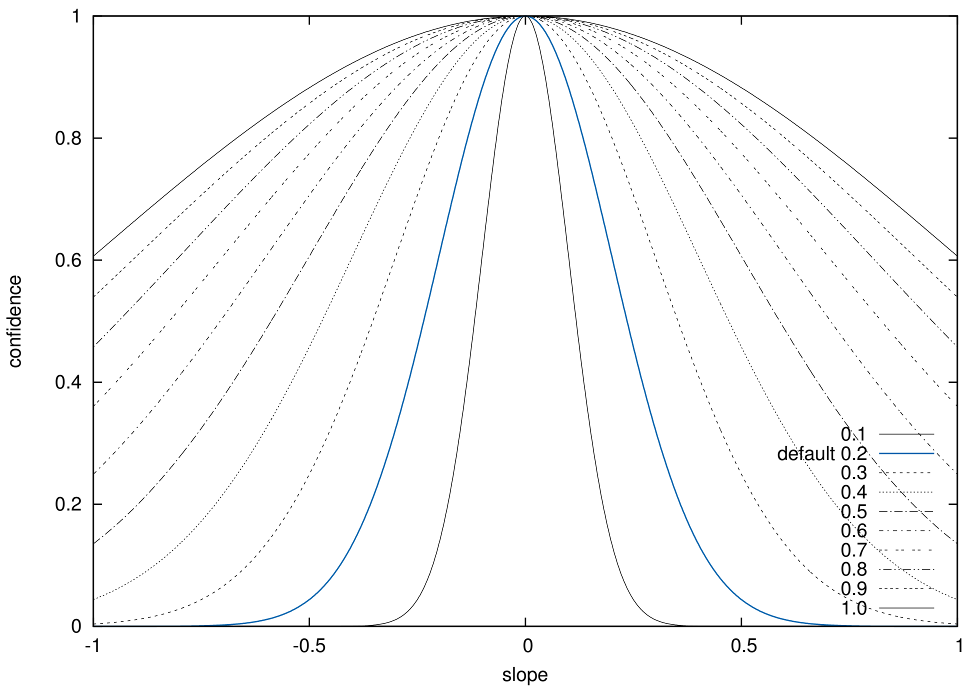


Figure 6.2.9: The confidence based on the time series slope c_{slope} modeled using the ratio of Normal distributions probability densities with $\mu = 0$ and σ ranging from 0.1 to 1.0

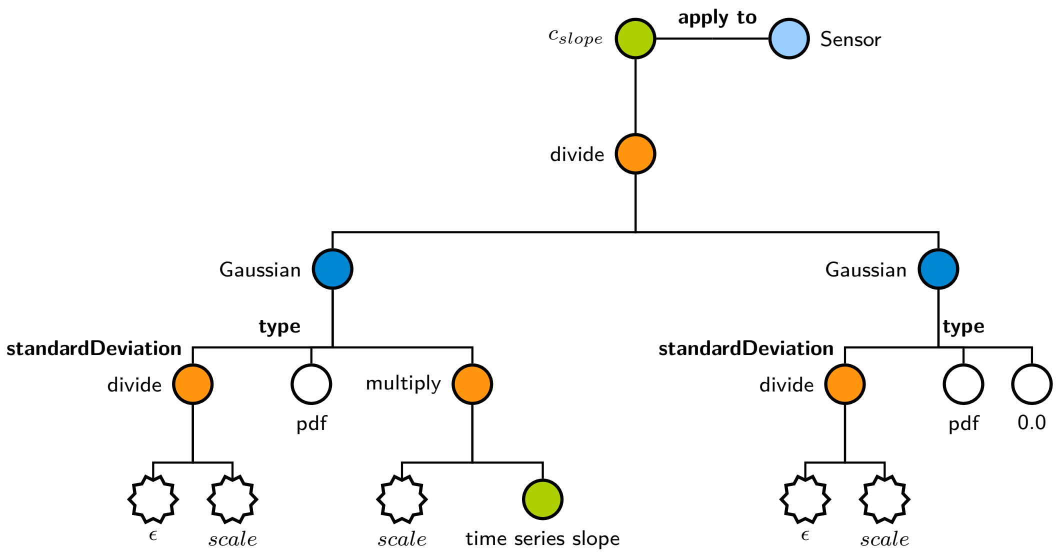
where s is the slope of the linear regression and

$$N(x|\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{6.2.4}$$

Note that the denominator part $N(0|0,\sigma^2)$ reflects the highest possible probability density (see figure 6.2.9).

Expression 6.2.14 c_{slope}

The c_{slope} *metric* is expressed as the ratio of Normal distribution probability densities using *Gaussian model expressions.* Specifically, we determine density for the time series slope over the maximum possible density. Here, we factor in the system parameter *scale* for time scale purposes as well as ϵ to allows adjustments to the confidence degradation rate as slopes deviate more and more from constant.



Time-Value Correlation The relationship between detected radiation values and time is important when determining trustworthiness. Linear dependence is one of the most noteworthy correlation measures as it can be used to predict the behavior of one variable based on another. We employ a basic approach, the Pearson product-moment correlation coefficient for set of data X and Y defined as

$$R(X,Y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_x s_y} \tag{6.2.5}$$

where $\bar{x}$ and $\bar{y}$ are the sample mean and s_x and s_y are the sample standard deviations. We want to point out that the sample correlation is used because the radiation time series values are only complete up to a certain point in time.

Expression 6.2.15 sample correlation coefficient
The sample correlation coefficient according to Pearson can be modeled as a byproduct of the *linear regression model expression*

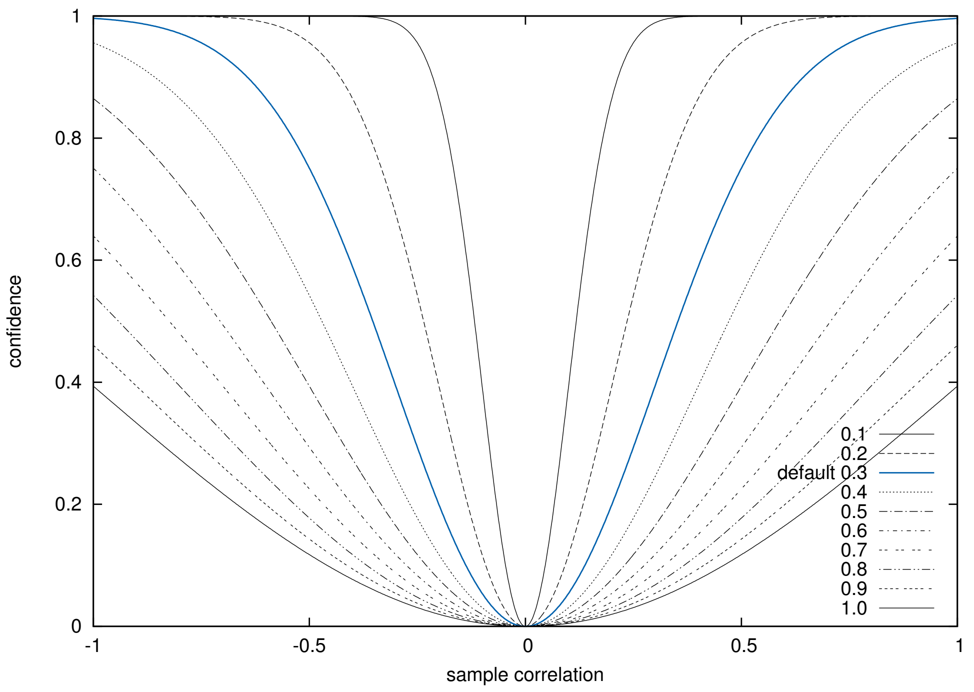
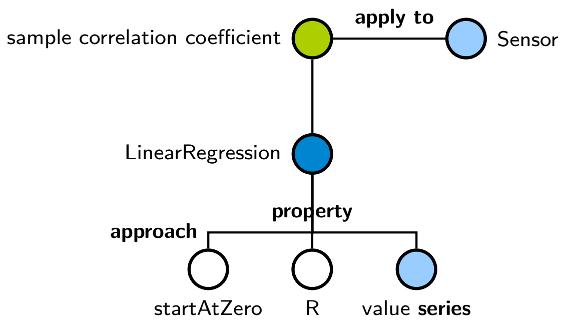


Figure 6.2.10: The confidence based on the sample correlation $c_{correlation}$ modeled using ratios of Normal distributions probability densities with $\mu = 0$ and σ ranging from 0.1 to 1.0



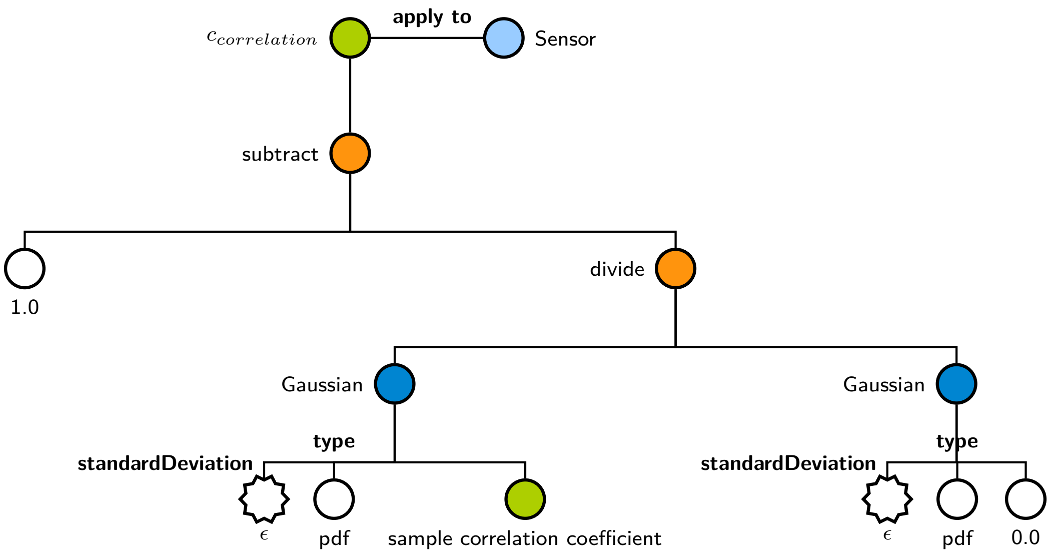
Linear correlation ranges from -1 to 1 such that the closer the value is to either the stronger the correlation. We utilize a similar approach as the for the slope confidence. However, we are interested in the complement probability. Hence

$$c_{correlation} = 1 - \frac{N(r|0, \sigma^2)}{N(0|0, \sigma^2)} \tag{6.2.6}$$

where r is the correlation coefficient and $N(x|\mu, \sigma^2)$ the Normal distribution as defined in equation 6.2.4.

Expression 6.2.16 $c_{correlation}$

The $c_{correlation}$ *metric* is expressed as the ratio of Normal distribution probability densities using *Gaussian model expressions*. We determine density for the time series correlation over the maximum possible density. In addition, we factor in the system parameter ϵ to allow adjustments to the confidence degradation rate as correlations are not as strongs.



Sensor Radiation Values Distribution The distribution of detected radiation values can be used to determine calibration accuracy of sensors and as such trustworthiness in their values. The assumption is that after initial calibration the sensors reading errors should follow a Normal distribution. Here, we base our confidence assessment on the Anderson-Darling test [5], specifically in the context of testing whether a given distribution is a Normal distribution. It defines the following test statistic

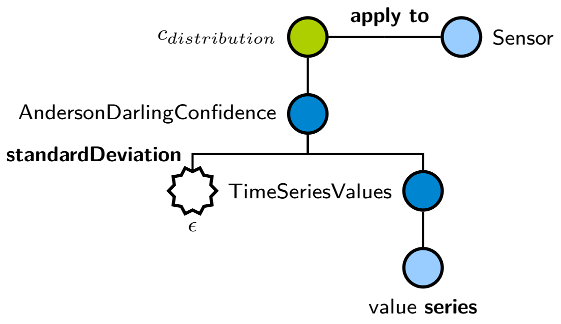
$$A^2 = -n - \frac{1}{n} \sum_{i=1}^{n} (2i-1)(\ln \Phi(Y_i) + \ln(1 - \Phi(Y_{n+1-i}))) \tag{6.2.7}$$

where n is the number of values, Φ the cumulative distribution function of the Normal distribution, and $Y_i = \frac{X_i - \hat{\mu}}{\hat{\sigma}}$ the standardized radiation values.

However, because of the limited availability of critical values we perform an approximation of the critical values provided by Stephens [161] using a Normal distribution. Furthermore, the Anderson-Darling test is originally designed as a hypothesis test yielding two outcomes. In order to model confidence we modify the approach and report the cumulative distribution probability of the test statistic A^2 given the Normal distribution approximating the critical values best.

Expression 6.2.17 $c_{distribution}$

The $c_{distribution}$ *metric* uses a modified Anderson-Darling test as described above to determine whether time series values come from a normal distribution with $\mu = 0$ and standard deviation specified by system parameter ϵ. This confidence is calculated with the *Anderson-Darling model expression.*

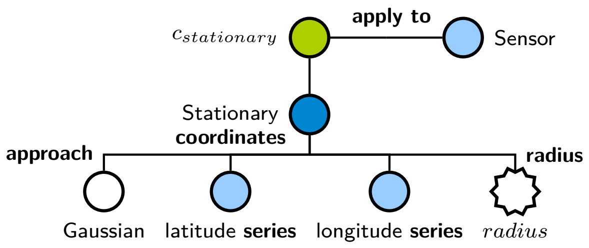


Mobile vs. Stationary We need to model two categories of sensors, stationary and mobile. Earlier, we discussed how our density-based clustering has the ability to identify both. Another aspect that we need to consider is that detected radiation measurements from stationary sensor should be trusted more than mobile ones because environmental changes have a lesser effect on the readings.

In order to factor this aspect into our trustworthiness assessment of sensors, we need methods to identify sensors whether a sensor is mobile or stationary. We utilize the following technique. We designate the first coordinates of a time series of radiation values as the center. Then we assess how many of the other coordinates in the time series fall with a certain radius using one of two approaches

- determine the percentage of time series values whose coordinates fall with in the specified radius, thus treating the radius as a cut-off
- calculate the average probability of the coordinates being part of the sensor based on ratios of Normal distributions and using the radius as standard deviation

Expression 6.2.18 $c_{stationary}$
The $c_{stationary}$ *metric* is modeled using a specific *stationary model expression*. It supports both of the approaches discussed above where radius is expressed as a system parameter. Here, we apply the *stationary* on the *latitude* and *longitude attributes*.



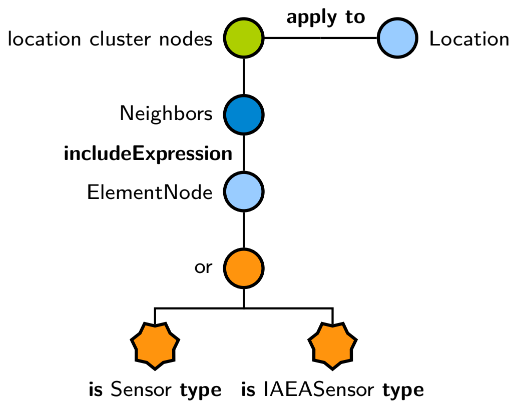
6.2.4.3 Location Clusters

Given the fact that sensors close to each other should detect similar radiation values leads us to model relationships between them. In particular, our density-based clustering results in location clusters. All sensors within the same location cluster are related. We can exploit these relationships and in this section discuss several approaches to assessing trustworthiness of individual sensors based on their relationship with others.

First, we need to discuss some of the basic relationship *metrics* we will use. For instance, each *location element node* is related to a collection of *sensor* and *IAEA sensor element nodes*.

Expression 6.2.19 location cluster nodes

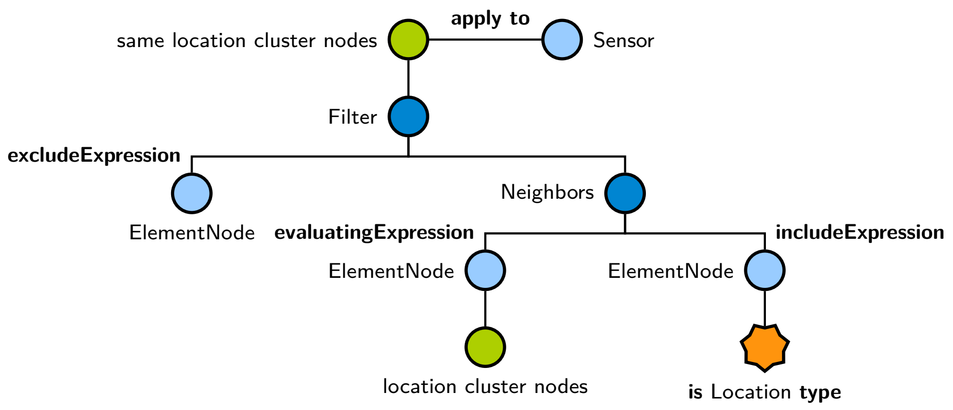
We can retrieve the related *sensor* and *IAEA sensor element nodes* using the *neighbors model expression* limited to these two types of *element nodes*.



Furthermore, from the perspective of an *sensor* or *IAEA sensor element node* we need to be able to fetch related *element nodes* as in *element nodes* belonging to the same location cluster.

Expression 6.2.20 same location cluster nodes

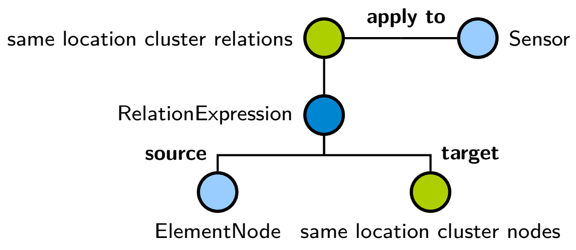
The *element nodes* related to a particular *sensor* or *IAEA sensor element node* can be retrieved by first getting the according *location element node* using the *neighbors model expression* and then getting all its related *element nodes* while excluding itself from the list using the *filter model expression*.



For most of the location cluster confidence *metrics* we perform comparisons between *sensor* or *IAEA sensor element nodes* within the cluster. Therefore we need to be able to express these relationships accordingly.

Expression 6.2.21 same location cluster relations

Relationships can be created temporarily, i.e., as part of the evaluation of an *expression*, using the *relation model expression* which simply specifies the *source* and *target* of a *relation edge*. In this case, we choose the *source* to be the particular *element node* the *metric* is applied to and the *target* as the list of *element nodes* from the same location cluster. This will automatically model a list of *relation edges* where the *source node* is fixed and the *target node* drawn from the provided list of *element nodes*.

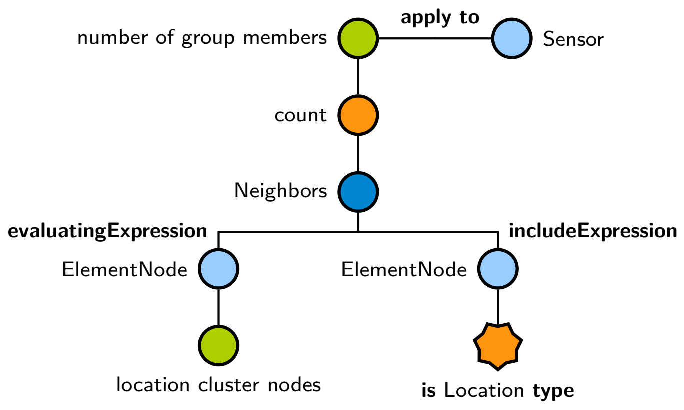


As such we can develop several trustworthiness assessments based on the relationships between sensors within the same location cluster.

Number of Location Cluster Members The number of correlated sensors within a location cluster directly impacts our trustworthiness. This is due to the fact that we could view a location cluster as a system of "checks and balances" where it becomes harder for a sensor to be dishonest given that it will not fare well in comparison to others. Thus, a low number reduces the number of comparisons possible and therefore should decrease our trustworthiness in the sensor that are part of the cluster as well.

Expression 6.2.22 number of group members

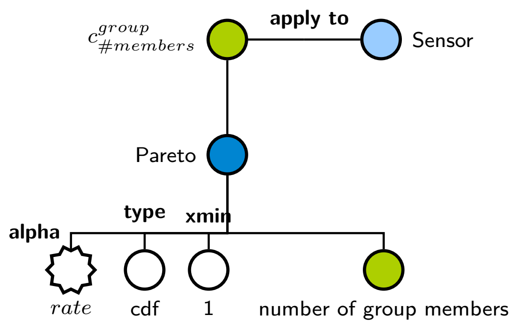
The number of group members is a straightforward count of the *element nodes* in a location cluster. We can reference the appropriate location cluster using the *neighbor model expression.* Note that this includes the *sensor* itself in the count.



We utilize the same approach as for the number of detected radiation values in a time series where the confidence is modeled using the Pareto distribution (see equation 6.2.1 and figure 6.2.8)

Expression 6.2.23 $c^{group}_{\#members}$

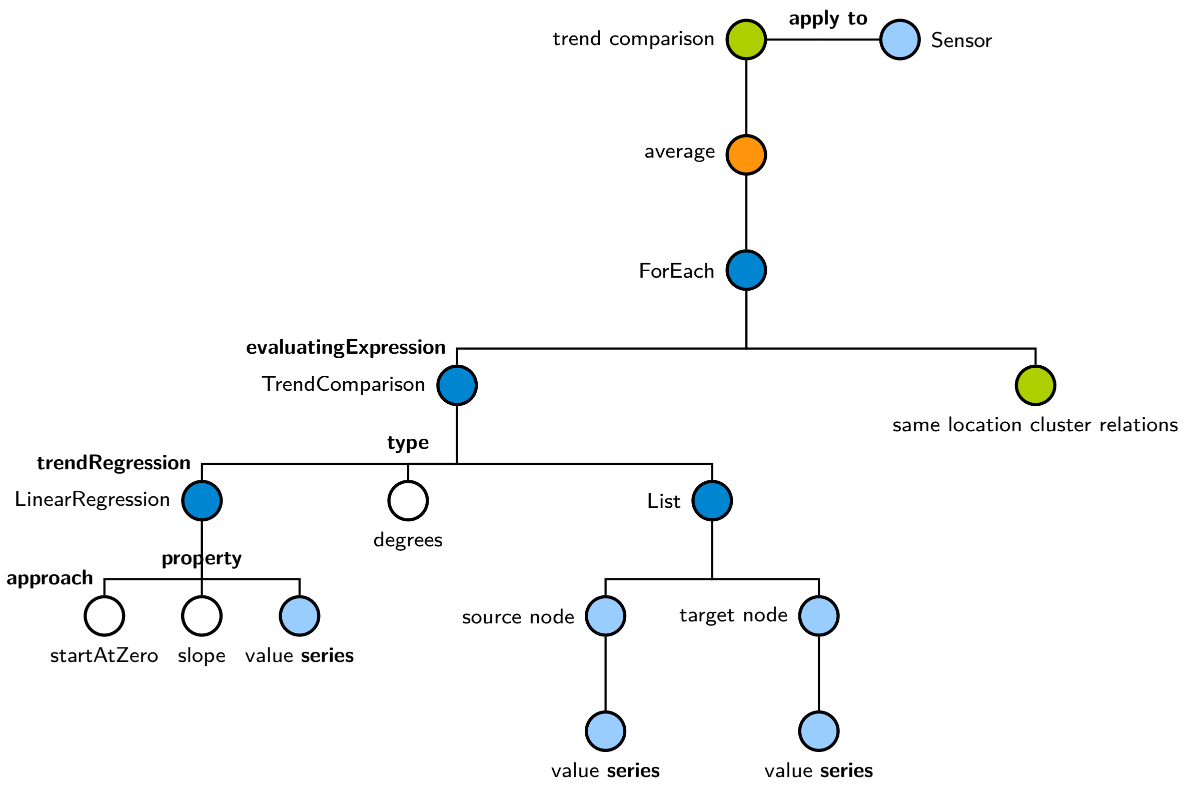
The $c^{group}_{\#members}$ *metric* is expressed using a *Pareto model expression* with an initial value x_{min} and a rate parameter *alpha*. As the result type we specify the cumulative distribution function (*cdf*).



Trend/Slope Comparison The trend of sensor values gives an indication of the overall radiation over time. Here, we are interested in determining whether the trend of a time series is vastly different from trends in the same location cluster. Note that each comparison is performed on a pair of time series values. As we have several options to express the trend of a time series we choose a linear regression to estimate the slopes. This approach allows us to compute the average degree difference of the slopes of one sensor compared to all others.

Expression 6.2.24 trend comparison metric

We express trend comparison as a *model expression* which is performed pairwise on the time series of the specified sensor and the other sensors in the same location cluster. We choose the result of each comparison to be in degree difference based on a linear regression of the time series.



The confidence in the average slope difference can then be modeled using the ratio of Normal distribution densities similar to the one used for the c_{slope} *metric* (see equation 6.2.3 and figure 6.2.11).

Expression 6.2.25 $c_{\frac{group}{trends}}$

The $c_{\frac{group}{trends}}$ *metric* is defined as the ratio of two Normal distribution densities represented by *Gaussian model expressions*. We use the system parameter ϵ to allow for confidence adjustments by adjusting the standard deviation of the distributions.

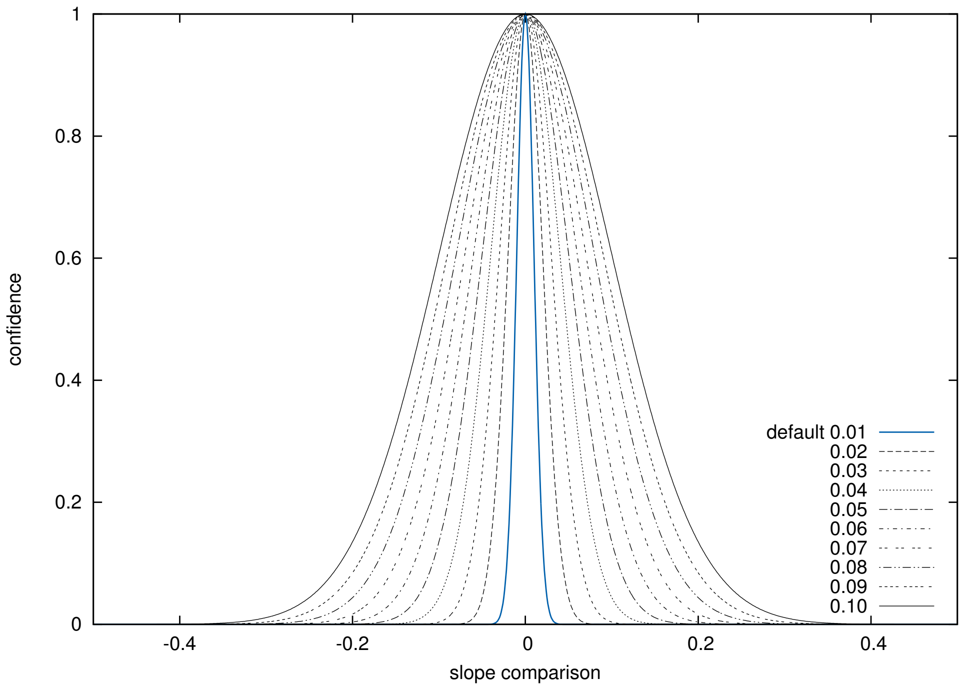
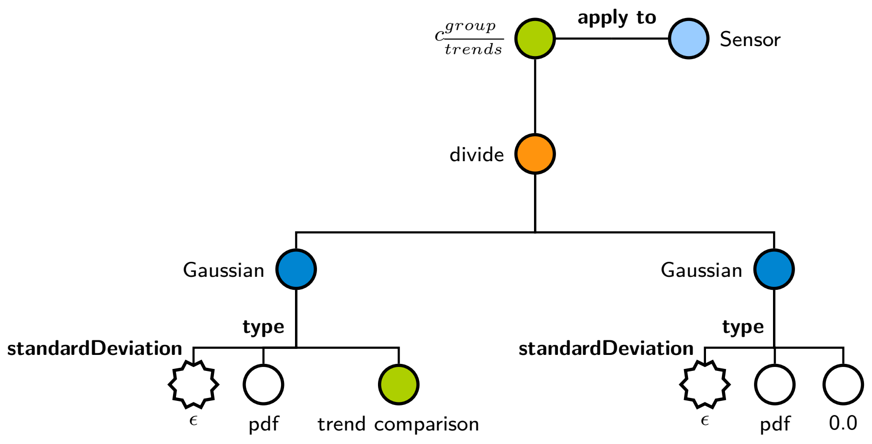


Figure 6.2.11: The confidence based on the comparison of time series slopes $c_{\frac{group}{trends}}$ modeled using the ratio of Normal distributions probability densities with $\mu = 0$ and σ ranging from 0.01 to 0.1

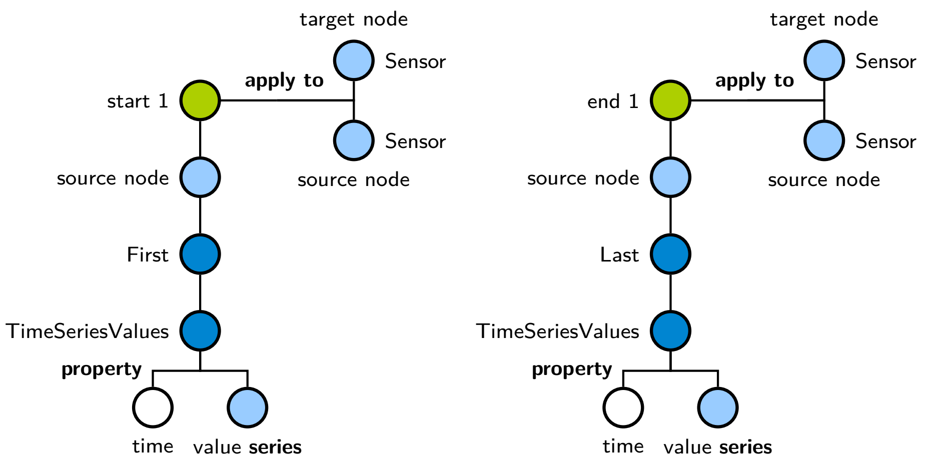


Recency When comparing measurements one important factor to consider is how old they are. For instance, recent sensor measurements should be considered more valuable and more trustworthy than older ones. Here, we incorporate this notion of recency to assess the confidence in a particular sensor's values given sensors in the same location cluster.

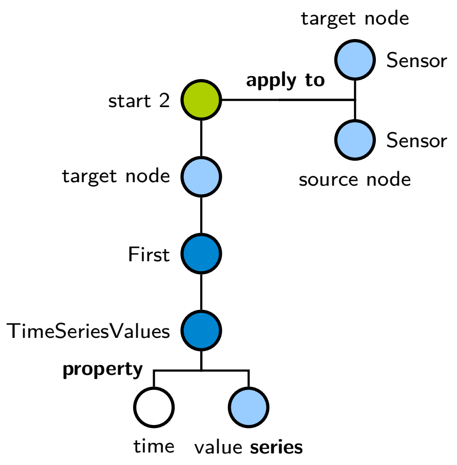
In order to evaluate the time relationship between two time series we need to be able to refer to the start and end of each. Since we choose to model the comparison as an *expression tree*, the two sensors with their respective time series are expressed as part of a *relation*. This enables us to define the start and end times of the time series as follows.

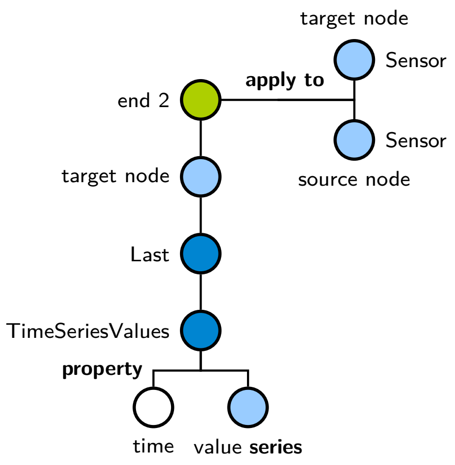
Expression 6.2.26 time series start and end time metrics

The *time series values model expressions* can be used to retrieve a list of values as well as a list of time instances reflecting the time series of a particular sensor. Our framework supports several *list expressions* of which *first* and *last* refer to the first and last elements of a particular list respectively. As such we are able to express the start and end of the first time series using the *source node reference*.



The second time series corresponds to the *target node* of the *relation* the *expression* is being applied to.





There exist three cases that we need to evaluate when comparing two time series A and B with respect to recency. First, time series A occurred strictly before B (i.e., no overlap with $end_1 < start_2$). Second, the time series are reversed and time series B occurred strictly before A (i.e., no overlap with $end_2 < start_1$). Third there is a partial or complete overlap of the two time series. Hence, we define recency as follows.

$$receny(A, B) = \begin{cases} start_2 - end_1 & \text{if } end_1 < start_2 \\ start_1 - end_2 & \text{if } end_2 < start_1 \\ 0 & \text{otherwise} \end{cases} \tag{6.2.8}$$

Using the start and end time metrics from above we can model this formula accordingly.

Expression 6.2.27 recency metric

The three cases are modeled using *case expressions* inside a *switch expression*. Here, the *replacement expression* is only applied to the case where the *pattern expression* evaluates to true. Note that this case testing is done in sequence such that the last case can be seen as the default case being applied when no pattern matches. By default the result of the recency calculation is in milliseconds. We provide a *system parameter scale*

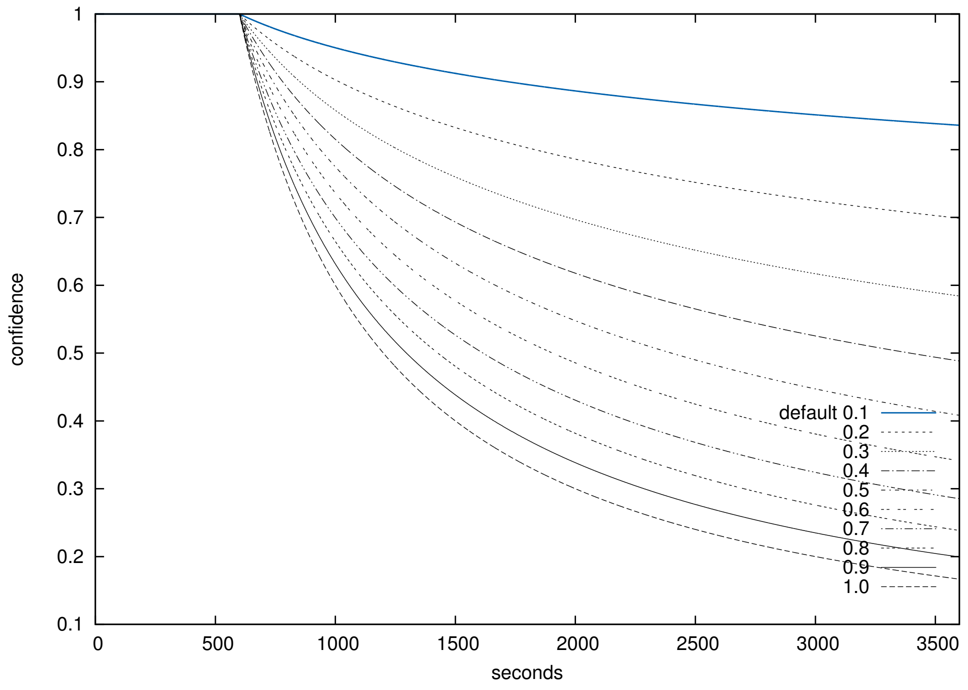
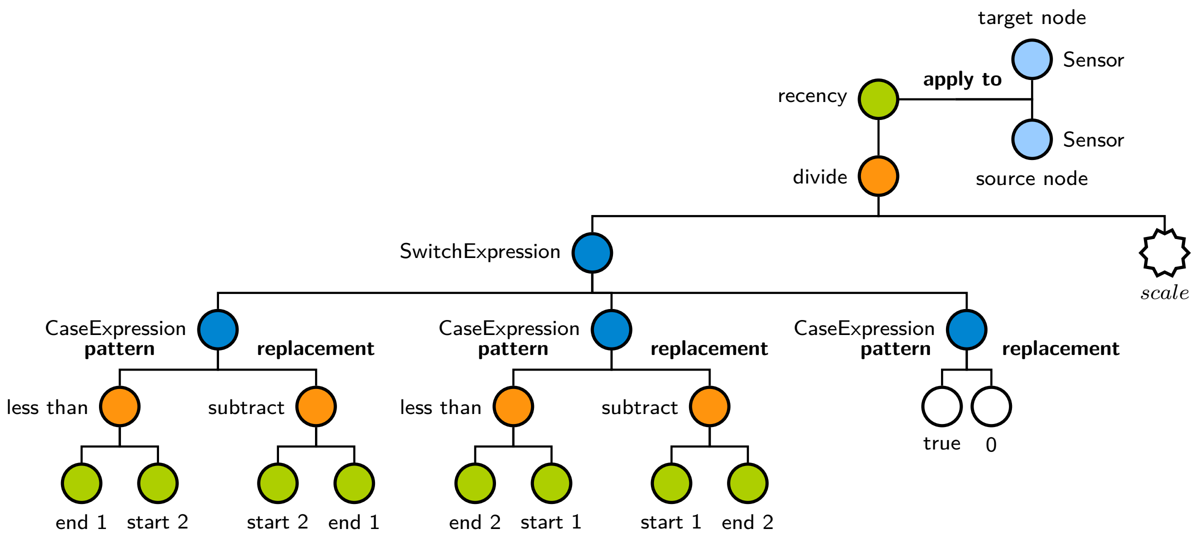


Figure 6.2.12: The confidence based on recency $c^{group}_{recency}$ modeled using the Pareto survival function with $x_{min} = 10minutes$ and α ranging from 0.1 to 1.0

that allows conversion into other units of time.



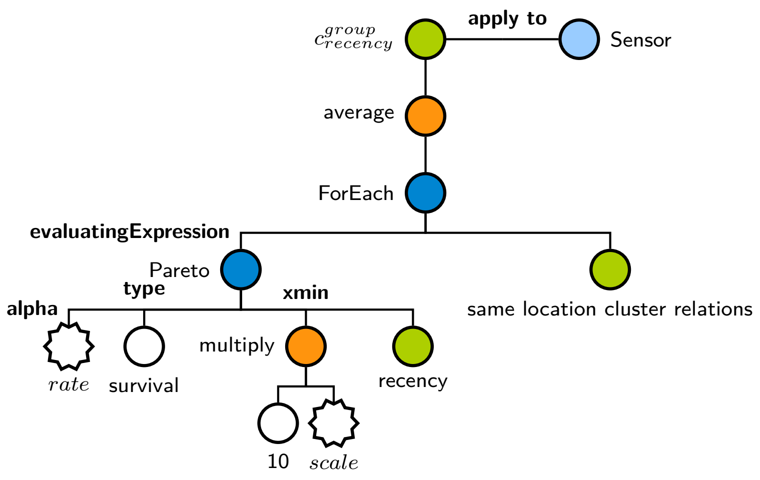
Here we utilize the survival function of the Pareto distribution to express the confidence in the recency of values.

$$Pareto_{survival}(x) = \begin{cases} \left(\frac{x_{min}}{x}\right)^{\alpha} & \text{if } x \geq x_{min} \\ 1 & \text{if } x < x_{min} \end{cases} \tag{6.2.9}$$

where x_{min} is the initial value after which the survival rate (i.e., confidence) decreases and α is the rate parameter (see figure 6.2.12). This approach is well suited for modeling a graceful decline in confidence after a certain time threshold has been exceeded.

Expression 6.2.28 $c^{group}_{recency}$

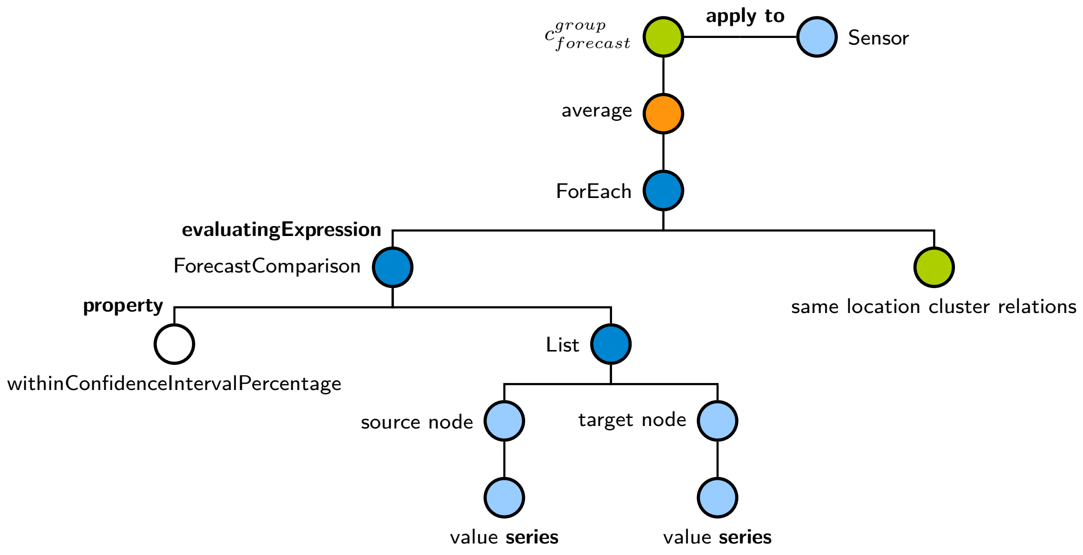
The $c^{group}_{recency}$ *metric* can be expressed using the *Pareto model expression* of type *survival* applied to the *recency* of a particular *relation*. Here we average the resulting confidences to provide an aggregate confidence assessment of a sensor with regards to others.



Forecast Apart from comparing the recency of two time series we can also evaluate how well one predicts the other. There exist several approaches to forecasting time series values. In line with other *metrics* discussed above we perform a linear regression on the earlier time series and evaluate the percentage of values of the other that fall within a certain confidence interval (e.g., 99%, 95%, etc.).

Expression 6.2.29 $c^{group}_{forecast}$

We model the $c^{group}_{forecast}$ *metric* using a *forecast comparison model expression* because of the complexity of having to perform a linear regression and determining time series values that have been correctly forecast. The overall confidence is then the average of the forecast confidences for all the *relations* of a *sensor* with others in the same location cluster.



Distribution Comparison In order to evaluate whether time series values of one sensor follow the same distribution as values of another sensor we can utilize Welch's t test Welch [177]. It represents a statistical test that allows us to determine if the means of two distributions are equal. The advantage here is that this test does not depend on the distributions having the same standard deviations. As such we can determine whether sensors generally yield similar results even if their accuracy varies. Welch's test statistic for two time series A and B can be defined as

$$Welch(A, B) = \frac{\overline{A} - \overline{B}}{\sqrt{\frac{s_A^2}{|A|} + \frac{s_B^2}{|B|}}} \tag{6.2.10}$$

where $\overline{X}$ is the sample mean, s^2 the sample variance, and $|X|$ the number of time series values.

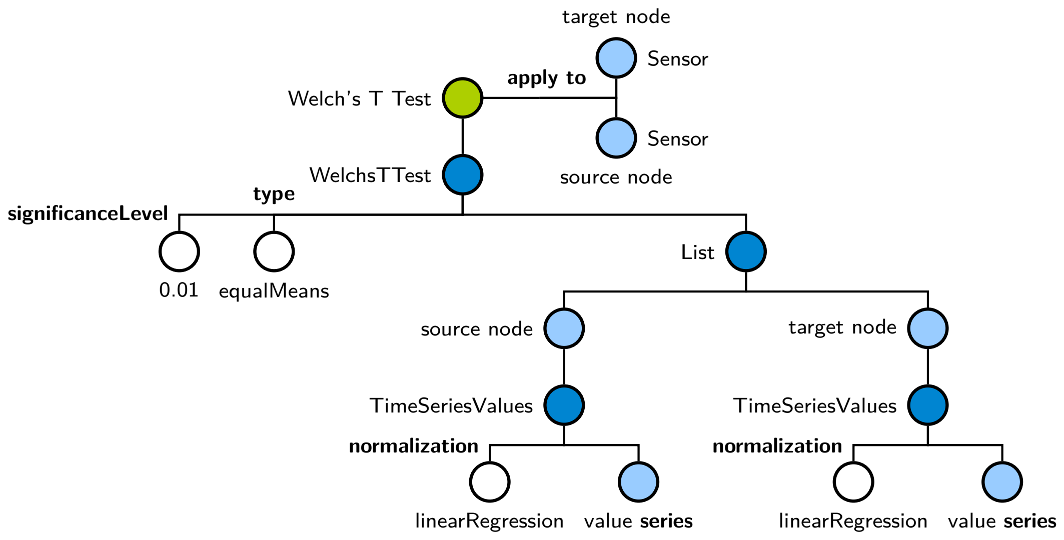
The null hypothesis of the two means being equal is evaluated using a two-sided significance test with the Student's t-distribution. The degrees of freedom for the Student's t-distribution in this case is specified as

$$\nu = \frac{\left(\frac{s_A^2}{|A|} + \frac{s_B^2}{|B|}\right)^2}{\frac{\left(\frac{s_A^2}{|A|}\right)^2}{|A|-1} + \frac{\left(\frac{s_B^2}{|B|}\right)^2}{|B|-1}} \tag{6.2.11}$$

Note that we normalize the time series values by calculating a linear regression on them and adjusting the values based on their difference to the trend.

Expression 6.2.30 Welch's T Test metric

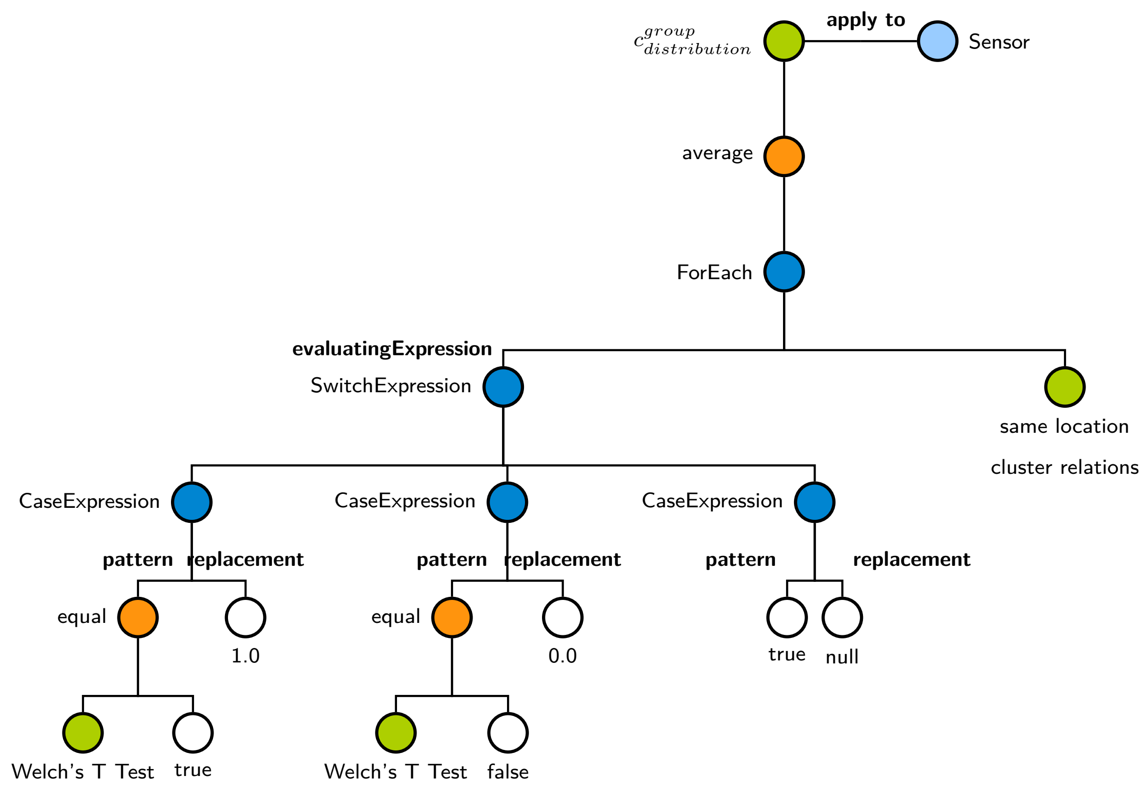
For both time series we apply *time series values model expressions* with the type of normalization set to linear regression. The *Welch's T Test model expression* then calculates the t and ν terms (i.e., according to equation 6.2.10 and 6.2.11) and performs a two-sided test with the specified *significance*.



Since the Welch's t test only provides a binary decision (i.e., equal means or not) due to the hypothesis testing approach we need to convert this into a proper confidence assessment. We chose to compute the average of the test results using 1 for *true* and 0 for $false$.

Expression 6.2.31 $c^{group}_{distribution}$

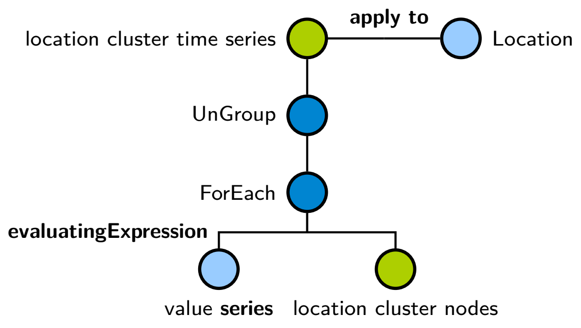
The $c^{group}_{distribution}$ makes use of several *model expressions*. For instance, the *for each model expression* takes care of evaluating the *Welch's t test* results for every *relation* that involves the specified *sensor element node*. The *switch* and *case model expression* then deal with the conversion of true or false into the appropriate numbers.



Outliers in Data As with the outliers in sensor readings of one sensor we can perform a similar analysis within a location cluster. Our approach here is to combine all time series data from the sensors in a particular location and detect possible outliers. We can utilize the same technique of Cook's distance [35] applied to individual sensors for finding "influential" data points.

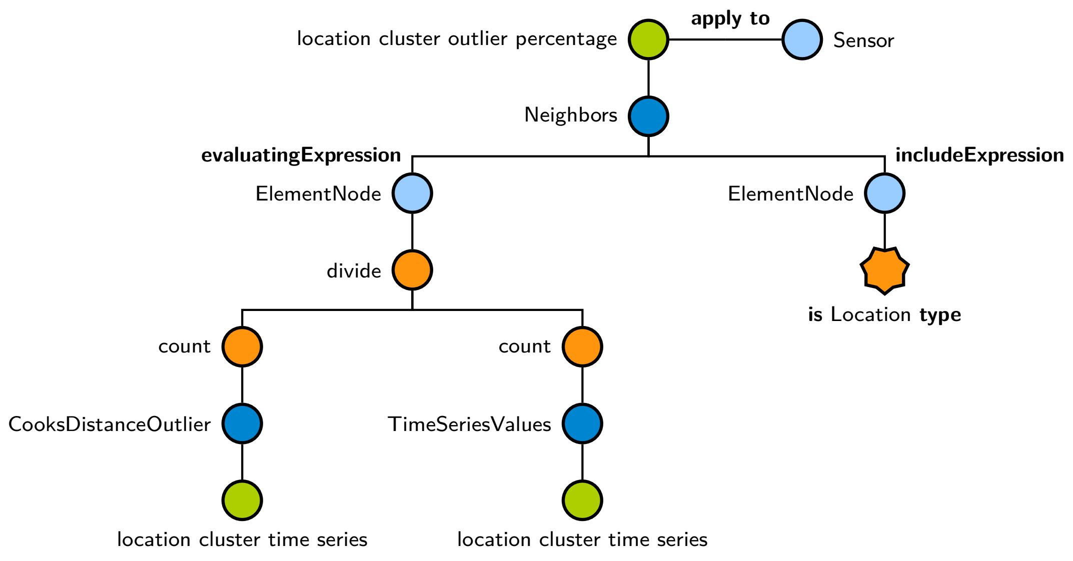
The following *metric* can be used to combine the time series' of all sensors within a location cluster.

> **Expression 6.2.32** location cluster time series metric
>
> First, we determine the time series for each of the *sensor element nodes* in the location. Second, we use the *ungroup model expression* to convert the individual time series into a single time series.



As such, we are able to calculate the percentage of outliers in a location cluster from the perspective of a particular sensor.

> **Expression 6.2.33** location cluster outlier percentage metric
>
> We model outliers within a location cluster as the percentage of outliers according to Cook's distance. Here, we utilize the *neighbor model expression* to determine the respective location cluster of a *sensor element node*. The location cluster outlier percentage is thus the ratio of the outliers to the overall number of values in the combined time series.



We use the same logic as for individual outliers where the more outliers exist the less confidence we have in a particular sensor.

Expression 6.2.34 $c^{group}_{outliers}$

The $c^{group}_{outliers}$ *metric* is expressed as the difference between 1 and the *location cluster outlier percentage*. This reflects the fact that more outliers mean less confidence and less outliers mean higher confidence.

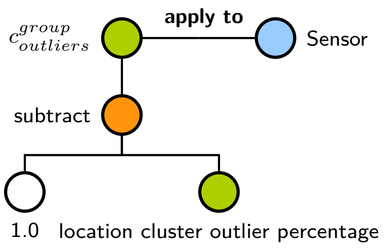


Table 6.2.4 shows an overview of the *metrics* we discussed as part of the *knowledge processing* phase of the framework. In the following section we will describe how these *metrics* can be incorporated into evaluating the trustworthiness of sensors.

Table 6.2.4: Radiation detection *metrics* overview

Type	Metric	Description
individual sensor	$c_{\#values}$	number of sensor measurements
individual sensor	$c_{outliers}$	outliers in data
individual sensor	c_{slope}	time series trend analysis
individual sensor	$c_{correlation}$	time-value correlation
individual sensor	$c_{distribution}$	sensor radiation values distribution
individual sensor	$c_{stationary}$	sensor being stationary
location cluster	$c^{group}_{\#members}$	number of group members
location cluster	c^{group}_{trends}	trend/slope comparison
location cluster	$c^{group}_{recency}$	recency of time series values
location cluster	$c^{group}_{forecast}$	forecast of time series values
location cluster	$c^{group}_{distribution}$	distribution of time series values comparison
location cluster	$c^{group}_{outliers}$	outliers in data of the group's time series values

6.2.5 Radiation Detection: Knowledge Evaluation

Determining the trustworthiness of a particular sensor can be accomplished in a variety of ways. We propose three approaches that can be easily be extended to more complex ones in future work. The goal here is not necessarily presenting the best overall approach but rather discuss how our *TrustKnowOne* framework is flexible enough to support a variety of approaches in a uniform way.

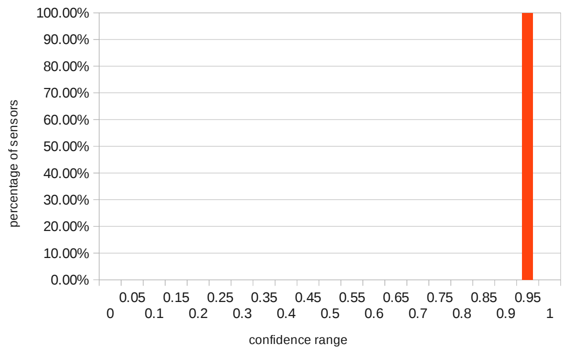
First, the *metrics* of the *knowledge processing* component provide two types of confidence assessments, sensor attributes on an individual level and relations to sensors within the same location cluster. The combination of these assessments can then be used to evaluate the trustworthiness of a sensor overall.

Second, instead of incorporating the continuous values of confidence assessments, e.g., on a 0 to 1 scale we can put them into class such as *low*, *medium*, and *high* based on thresholds. This simplification would allow easier decision making as it abstracts the assessments and presents them in a more user-friendly format.

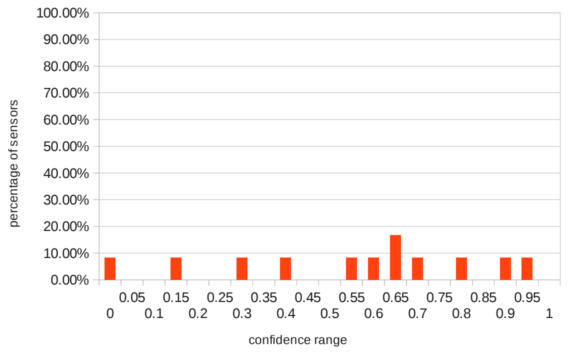
Third, there are two different authorities (owners) providing the radiation data upon which our decisions are based. A natural consequence of this is that we are able to weight assessments differently based on ownership.

6.2.5.1 Individual Sensors

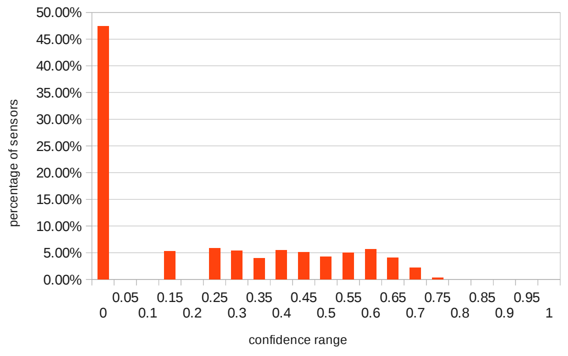
On an individual basis sensors can be assessed in a variety of ways. We described in detail several approaches as part of the *knowledge processing* section. Here, we discuss the results of these *metrics* from applying them to sensors.



(a) IAEA fixed monitoring post, $\mu = 95.8\%$ and $\sigma = 0.0\%$



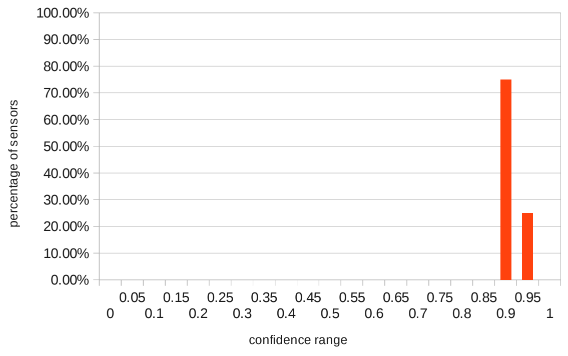
(b) IAEA transportable sensor, $\mu = 57.4\%$ and $\sigma = 29.0\%$



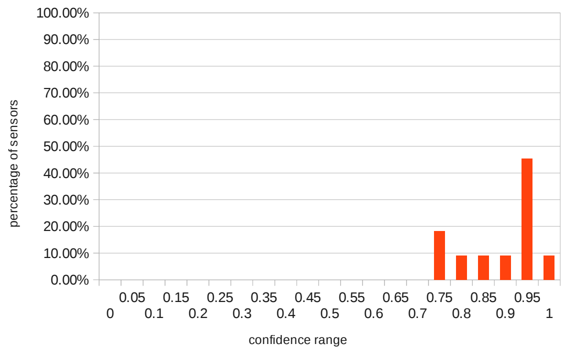
(c) Safecast , $\mu = 24.1\%$ and $\sigma = 25.5\%$

Figure 6.2.13: Individual sensors confidence in number of sensor readings by data source overview

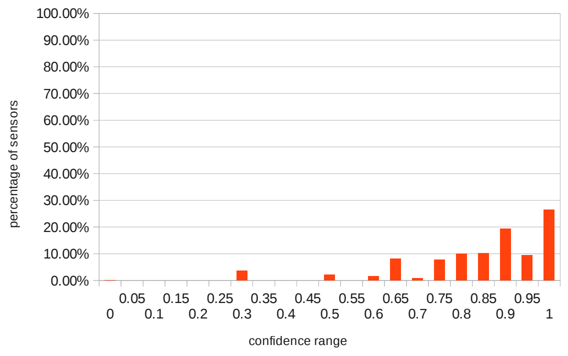
Number of Sensor Readings (expression 6.2.10) As shown in figure 6.2.13 sensors of the fixed monitoring post data set provide constantly high assessments whereas sensors from the transportable sensor or Safecast data set give mixed results. In particular, the Safecast data set exhibits a high number of sensors with no confidence in them. This is may be due to the nature of collaborative filtering where usually only short bursts of measurements are taken rather than over an extended amount of time.



(a) IAEA fixed monitoring post, $\mu =$ 94.1% and $\sigma = 1.6\%$



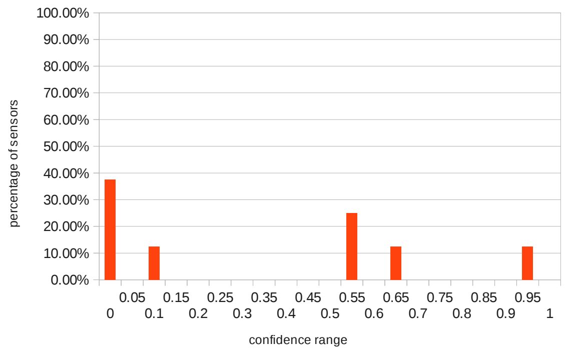
(b) IAEA transportable sensor, $\mu =$ 91.0% and $\sigma = 8.7\%$



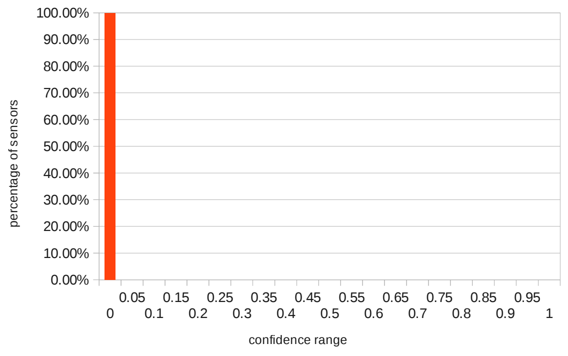
(c) Safecast , $\mu = 85.9\%$ and $\sigma = 16.2\%$

Figure 6.2.14: Individual sensors confidence in outliers in data by data source overview

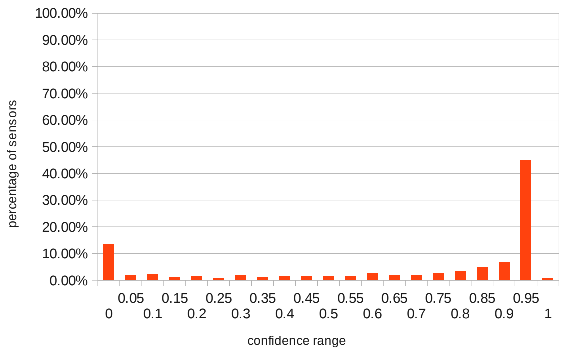
Outliers in Data (expression 6.2.12) Figure 6.2.14 shows that the percentage of outliers is a potentially useful discriminator for deciding trustworthiness as it determines a number of sensors with lower confidence values. This is true for the Safecast as well as the transportable sensor data set. For the fixed monitoring post data set high levels of confidence are to be expected since the sensors contain a large number of time series values thus decreasing the overall percentage of potential outliers.



(a) IAEA fixed monitoring post, $\mu = 36.2\%$ and $\sigma = 37.3\%$



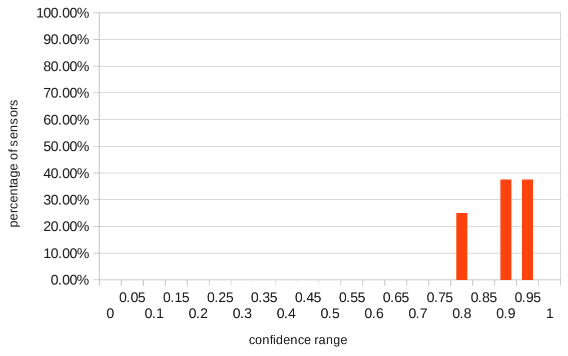
(b) IAEA transportable sensor, $\mu = 0.0\%$ and $\sigma = 0.0\%$



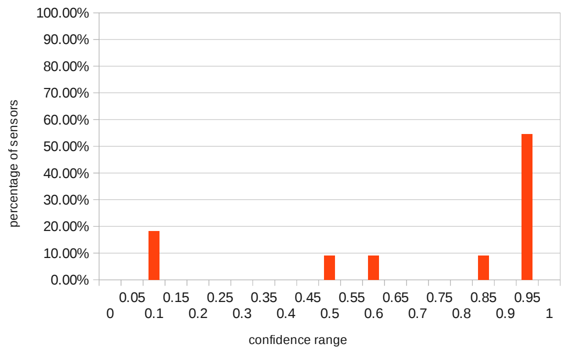
(c) Safecast, $\mu = 70.5\%$ and $\sigma = 37.2\%$

Figure 6.2.15: Individual sensors confidence in time series trend analysis by data source overview

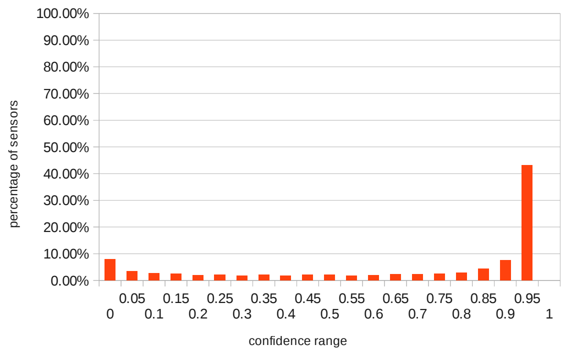
Time Series Trend Analysis (expression 6.2.14) Analysis of the time series trend is difficult and yields mixed results as shown in figure 6.2.15. This is due to the fact that there are a number of parameters affecting the calculation of this confidence such as start time, time interval, and time scale. As such, it may provide good separation just for the Safecast data set. However, it is hard to determine whether this kind of separation is useful.



(a) IAEA fixed monitoring post, $\mu = 92.2\%$ and $\sigma = 5.4\%$



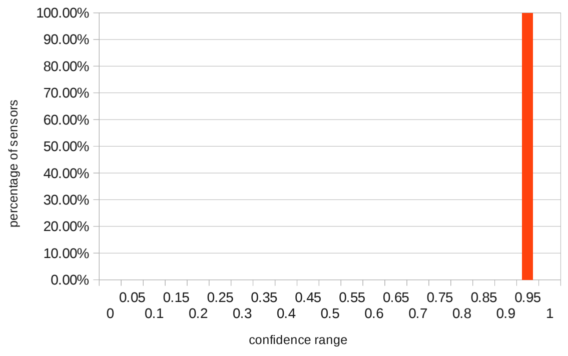
(b) IAEA transportable sensor, $\mu = 74.3\%$ and $\sigma = 35.3\%$



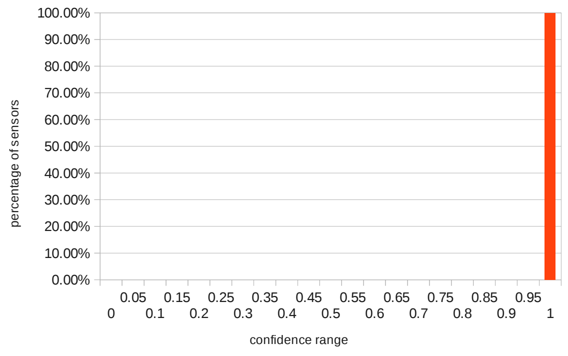
(c) Safecast, $\mu = 60.2\%$ and $\sigma = 47.0\%$

Figure 6.2.16: Individual sensors confidence in time-value correlation by data source overview

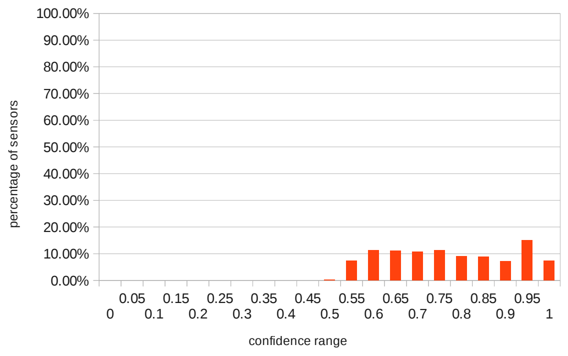
Time-Value Correlation (expression 6.2.16) Figure 6.2.16 shows the results of assessing the time-value correlation of a sensor's radiation time series. This confidence can be interpreted as a sign of how "stable" a time series is. For instance, the fixed monitoring post data set displays high confidence because of high linear correlation of time and values. The Safecast data set shows more of a separation which is useful for distinguishing between sensors of high and low trustworthiness.



(a) IAEA fixed monitoring post, $\mu = 99.9\%$ and $\sigma = 0.0\%$



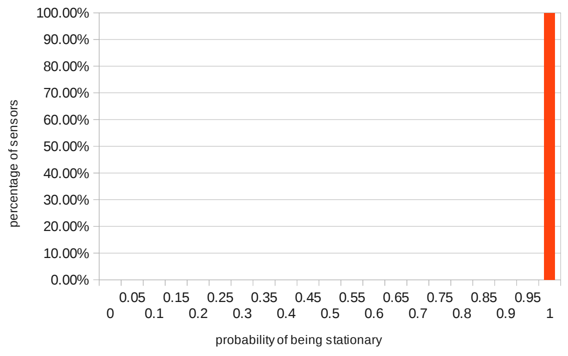
(b) IAEA transportable sensor, $\mu = 100.0\%$ and $\sigma = 0.0\%$



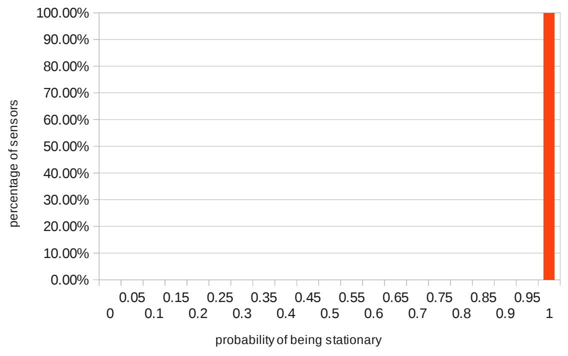
(c) Safecast, $\mu = 79.9\%$ and $\sigma = 14.1\%$

Figure 6.2.17: Individual sensors confidence in sensor radiation values distribution with $\epsilon = 10cpm$ by data source overview

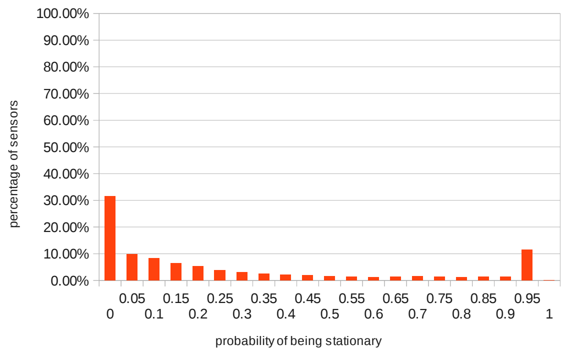
Sensor Radiation Values Distribution (expression 6.2.17) As shown in figure 6.2.17 a lot of sensors have radiation values that appear to be drawn from a normal distribution with a low standard deviation. This *metric* is in a way similar to the outlier percentage as we try to determine if there are any anomalies in the time series radiation levels. For the Safecast data set it provides a useful distribution of confidence assessments that can be used to identify sensors with low and sensors with high trustworthiness.



(a) IAEA fixed monitoring post, $\mu = 100.0\%$ and $\sigma = 0.0\%$



(b) IAEA transportable sensor, $\mu = 100.0\%$ and $\sigma = 0.0\%$



(c) Safecast, $\mu = 30.3\%$ and $\sigma = 34.4\%$

Figure 6.2.18: Individual sensors confidence in sensor being stationary with Gaussian $radius = 0.0001 \approx 100m$ by data source overview

Mobile vs. Stationary (expression 6.2.18) Distinguishing whether a sensors is mobile or stationary is difficult. However, our method with a Gaussian radius provides an approximation that can yield a powerful assessment of confidence in a sensor. As discussed earlier, mobile sensors are impacted more by environmental changes than stationary ones. Here, figure 6.2.18 correctly identifies both IAEA data sets to consist of stationary sensors and provides a wide distribution of confidence for Safecast sensors.

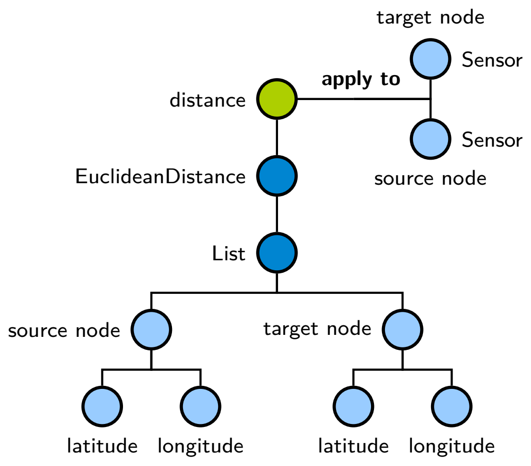
6.2.5.2 Location Clusters

Trustworthiness assessments can also be derived by comparing a sensor with sensors in the same location. The assumption is that physically close sensors often exhibit similar environments and should detect similar radiation values. The sensors in the Safecast and IAEA data sets already have been associated with the appropriate location cluster. However, incorporating new sensors requires identifying the closest location cluster for the particular sensor.

There exist several *metrics* that can be used to calculate the distance between two sensors given their coordinates. Here, we use a basic Euclidean distance computation.

Expression 6.2.35 sensor distance metric

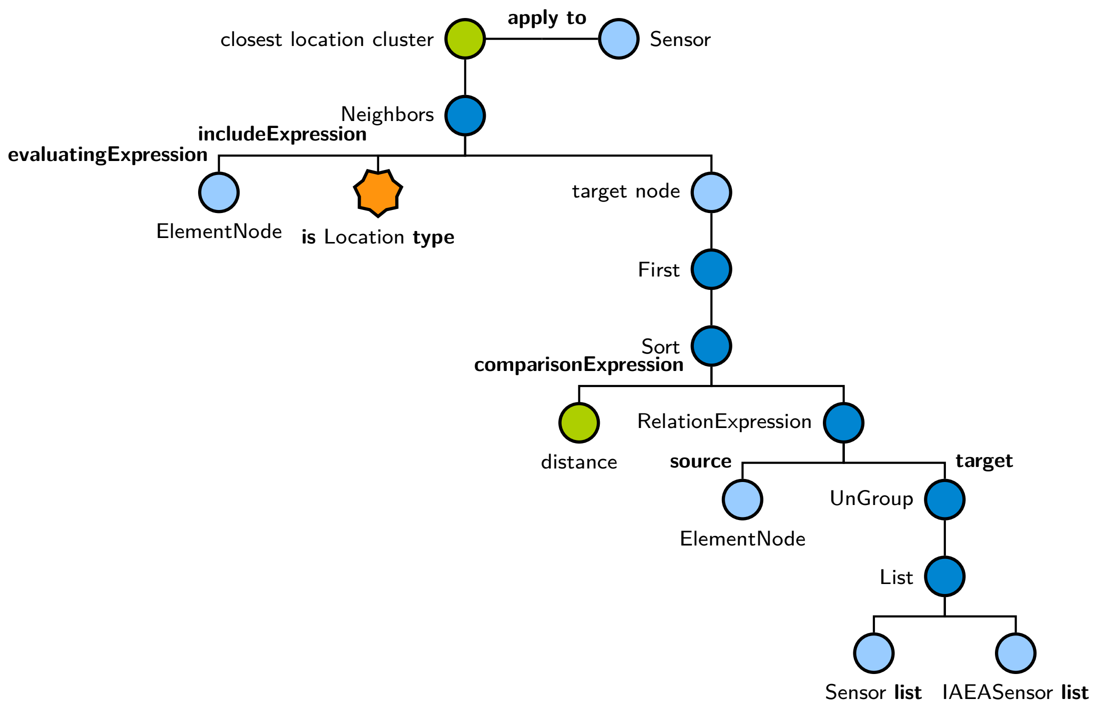
The distance *metric* is applied to two *sensor* or *IAEASensor element nodes*. While the particular distance function may be substituted (e.g., *Euclidean distance model expresssion*) the rest of the *metric* remains the same. We retrieve the coordinates for each of the sensors the *metric* is applied to and use it as the input for the distance calculation. Note that here we only calculate the distance give the last/most recent coordinates. As such the *metric* could be extended to determine the distances at all time instances.

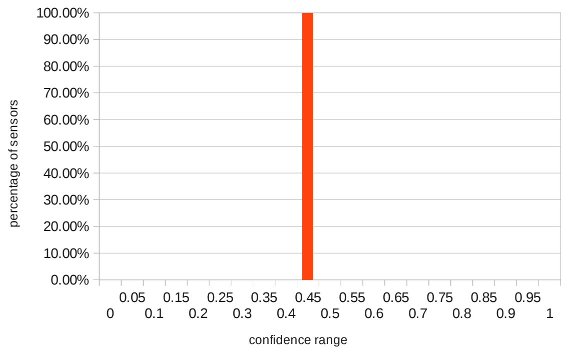


This distance *metric* or another can then be used to determine the sensor's closest location cluster. We need to keep in mind that the smallest distance may be greater than the maximum radius that was used for density-based clustering. In this case the sensor would not be part of the location. However, we can relax this requirement and assume that unless the distance is too great, comparisons within location cluster group members yield useful confidence assessments.

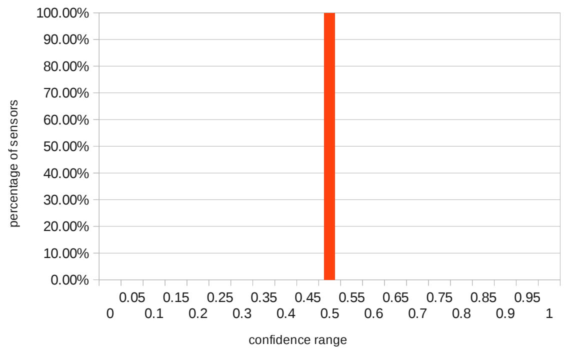
Expression 6.2.36 closest location cluster metric

In order to determine the closest location cluster we need to determine the closest sensor. Note that while this *metric* uses the pair-wise comparison approach to find that sensor an improved version could make use of a spatial index such as *R-trees* [66]. First we combine the list of *Sensor* and *IAEASensor element nodes* using the *list* and *ungroup model expressions*. Then we create *relation edges* to those sensors with the *relation model expression* which are then sorted by the result of the *distance metric*. Finally, we derive the *Location element node* respectively.

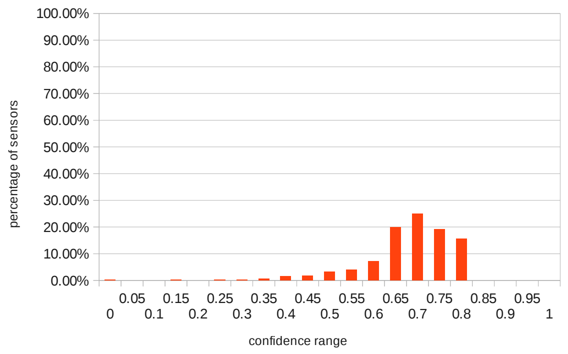




(a) IAEA fixed monitoring post, $\mu = 46.4\%$ and $\sigma = 0.0\%$



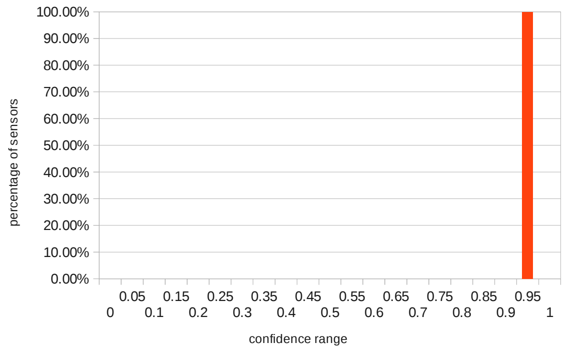
(b) IAEA transportable sensor, $\mu = 52.5\%$ and $\sigma = 0.0\%$



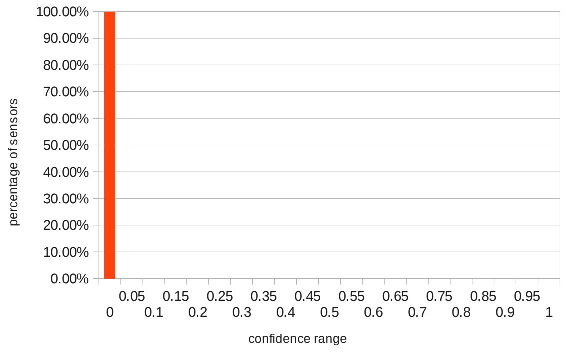
(c) Safecast, $\mu = 70.4\%$ and $\sigma = 10.9\%$

Figure 6.2.19: Location cluster sensors confidence in number of location cluster members by data source overview

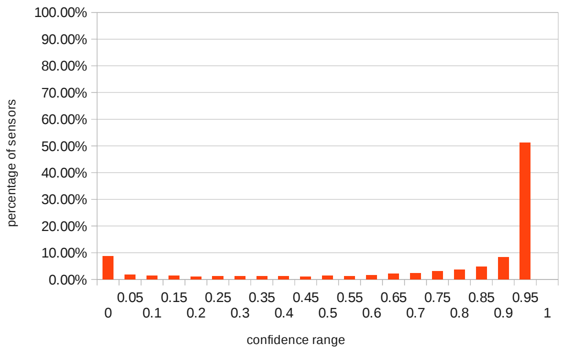
Number of Location Cluster Members (expression 6.2.23) Figure 6.2.19 shows a medium confidence for the sensors from the IAEA data set. This is due to the fact that both data sets contain only a limited number of sensors. On the other hand, Safecast sensors show a distribution that can be used to categorize sensors into low and high trustworthiness.



(a) IAEA fixed monitoring post, $\mu = 99.9\%$ and $\sigma = 0.0\%$



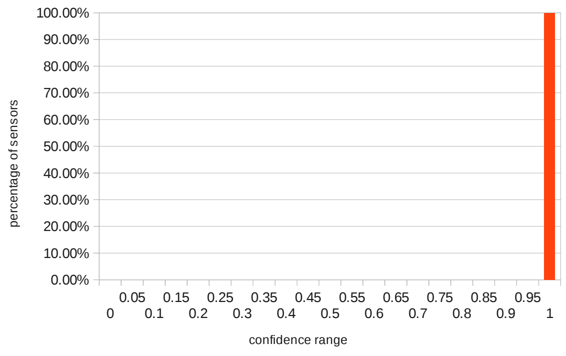
(b) IAEA transportable sensor, $\mu = 0.4\%$ and $\sigma = 1.2\%$



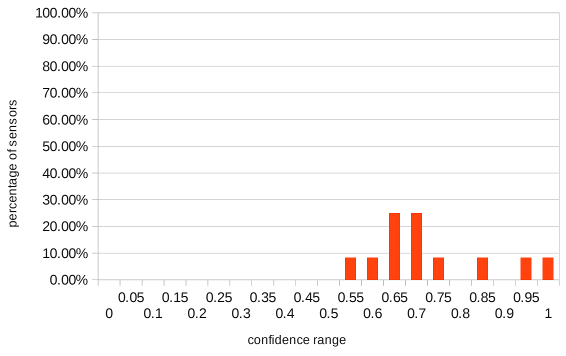
(c) Safecast, $\mu = 76.8\%$ and $\sigma = 33.3\%$

Figure 6.2.20: Location cluster sensors confidence in trend/slope comparison by data source overview

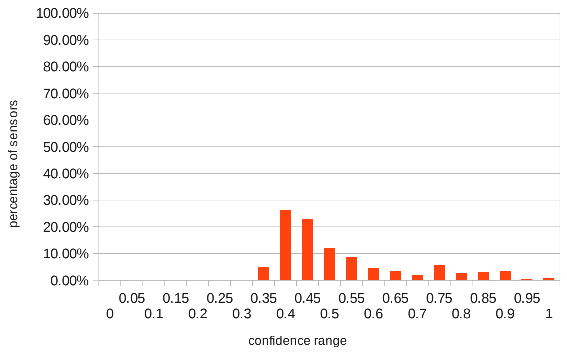
Trend/Slope Comparison (expression 6.2.25) As shown in figure 6.2.20 there is a vast difference between the data sets. For the fixed monitoring post the trend is overwhelmingly similar. Hence, the high confidence. In contrast, the transportable sensor data set seems to be the opposite with all trends being almost completely different. Safecast provides an assessment that ranks the majority of sensors highly while also establishing a wide distribution for other sensors.



(a) IAEA fixed monitoring post, $\mu = 100.0\%$ and $\sigma = 0.0\%$



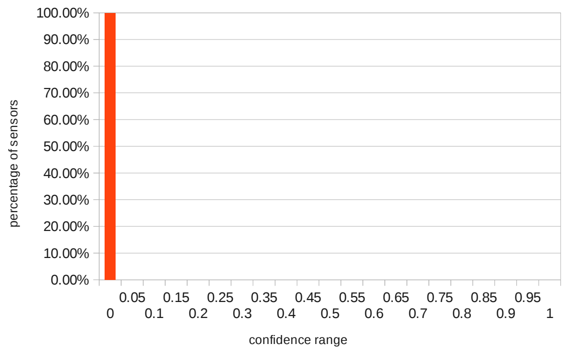
(b) IAEA transportable sensor, $\mu = 74.7\%$ and $\sigma = 13.5\%$



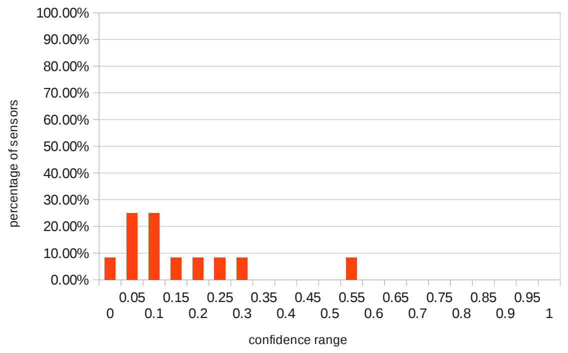
(c) Safecast, $\mu = 54.9\%$ and $\sigma = 15.2\%$

Figure 6.2.21: Location cluster sensors confidence in recency within 10 minutes by data source overview

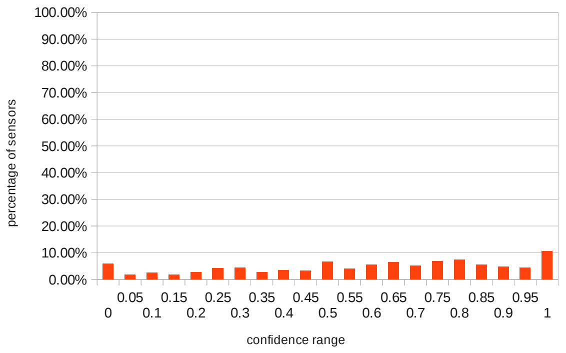
Recency (expression 6.2.28) The comparison of how recent radiation detection values were captured, helps us correlate time series. The idea is that sensor readings that happen close in time to each other should have similar values (given that they are in the same location cluster as well). As shown in figure 6.2.21 we can use this confidence assessment effectively to distinguish between low and high trustworthiness.



(a) IAEA fixed monitoring post, $\mu = 0.0\%$ and $\sigma = 0.0\%$



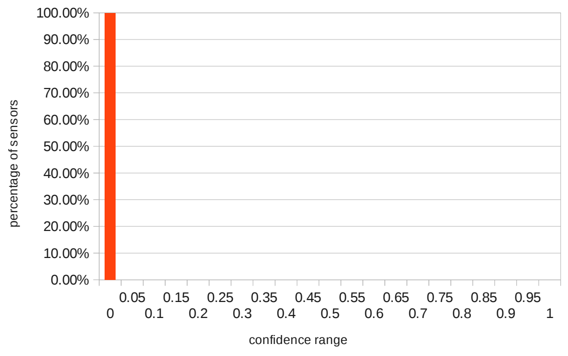
(b) IAEA transportable sensor, $\mu = 18.4\%$ and $\sigma = 15.9\%$



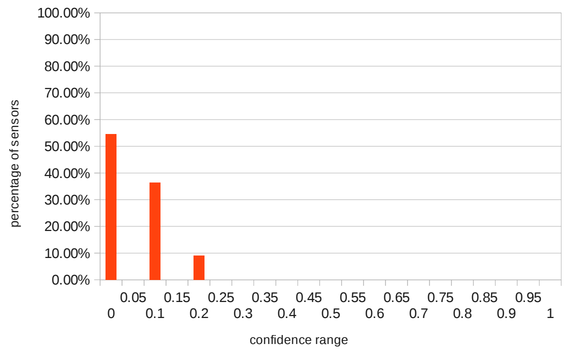
(c) Safecast, $\mu = 60.4\%$ and $\sigma = 29.9\%$

Figure 6.2.22: Location cluster sensors confidence in forecast within 95% confidence interval by data source overview

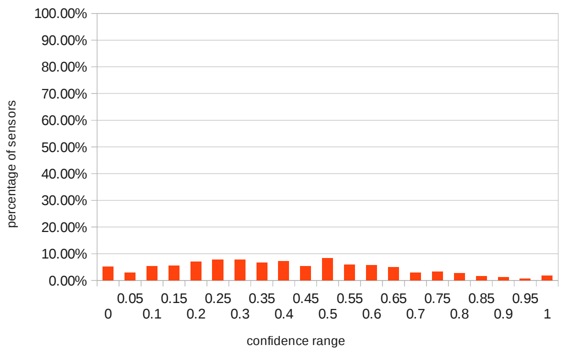
Forecast (expression 6.2.29) In figure 6.2.22 we can see that forecasting can be problematic since sensors from the fixed monitoring post have very low confidence while Safecast sensors exhibit confidence assessments in an almost a uniform distribution. This could indicate that sensors in general do not have have a lot of forecasting power over other sensors. Note that this *metric* aligns both time series' and therefore may not be as useful as others.



(a) IAEA fixed monitoring post, $\mu = 0.0\%$ and $\sigma = 0.0\%$



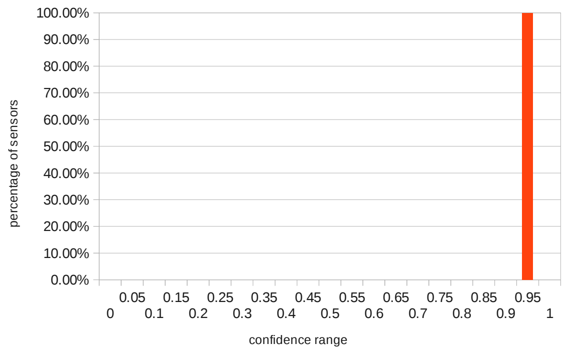
(b) IAEA transportable sensor, $\mu = 5.5\%$ and $\sigma = 6.9\%$



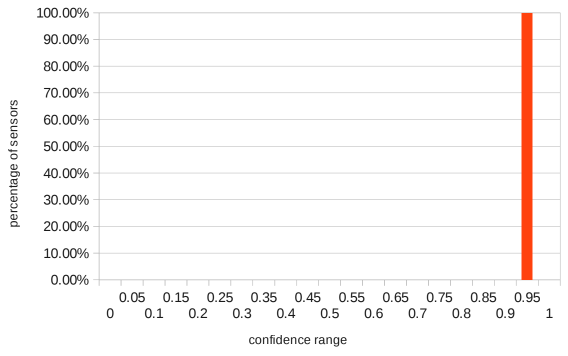
(c) Safecast, $\mu = 42.6\%$ and $\sigma = 24.2\%$

Figure 6.2.23: Location cluster sensors confidence in distribution comparison with linear regression normalized values and significance level of 0.01 by data source overview

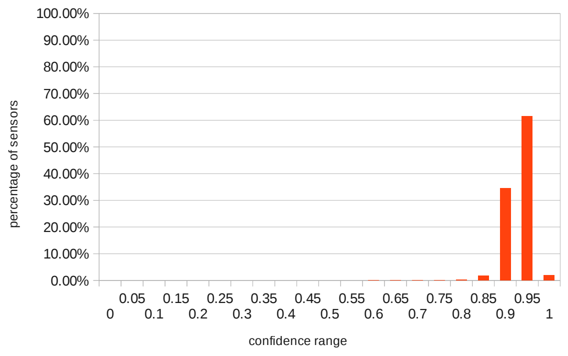
Distribution Comparison (expression 6.2.31) The results comparing the means of two time series' are shown in figure 6.2.23. It is noticeable that they are similar to figure 6.2.22. As such, the separation in terms of distribution for the Safecast data set may prove useful when categorizing sensors by trustworthiness. However, the generally low level of trustworthiness for the IAEA data set is surprising since they are stationary and consist of a large number of time series values.



(a) IAEA fixed monitoring post, $\mu = 97.6\%$ and $\sigma = 2.1\%$



(b) IAEA transportable sensor, $\mu = 98.8\%$ and $\sigma = 0.0\%$



(c) Safecast, $\mu = 95.4\%$ and $\sigma = 2.1\%$

Figure 6.2.24: Location cluster sensors confidence in outliers in data by data source overview

Outliers in Data (expression 6.2.34) Figure 6.2.24 shows a metric that essentially shows a distribution of outlier confidence assessments for an entire location. As such, sensors from the IAEA data set display high levels of confidence and confidence for sensors from Safecast is equally high. Note that this *metric* is primarily impacted by the number of group members and the number of time series radiation values. The more non-outliers the smaller the ratio of outliers to non-outliers and thus the percentage is.

6.2.5.3 Weighted Trustworthiness Assessment

A straightforward approach to assess the overall trustworthiness of a sensor is to consider the confidence assessments discussed above. Here, we have three options. First, we only take into consideration *metrics* that focus on an individual sensor. Second, we evaluate a sensor with regards to others in the same location cluster. Third, we model trustworthiness as a combination of individual and location cluster confidence assessments.

Incorporating individual sensor assessments can be done using a weighted sum approach.

$$trust_{weighted}^{individual}(Sensor) = \sum w_i \times c_i \tag{6.2.12}$$

where w_i is the assigned weight with $\sum w_i = 1$ and c_i one of the following individual sensor *metrics*

$c_{\#values}$ the confidence in the number of sensor measurements

$c_{outliers}$ the confidence in the outliers in data

c_{slope} the confidence in the time series trend analysis

$c_{correlation}$ the confidence in the time-value correlation

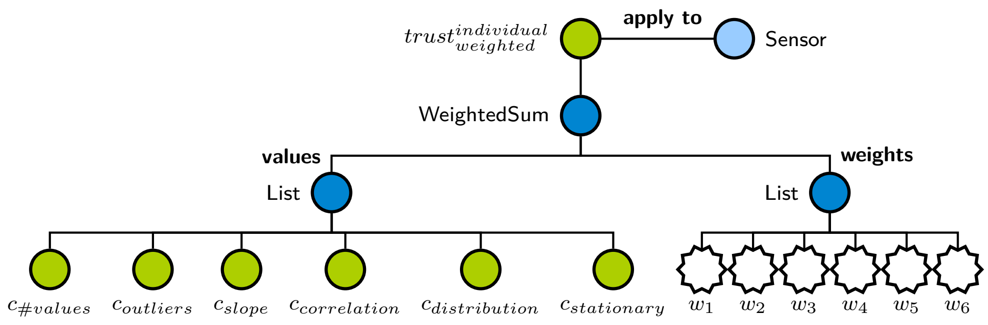
$c_{distribution}$ the confidence in the sensor radiation values distribution

$c_{stationary}$ the confidence in the sensor being stationary

which can be modeled as follows.

Expression 6.2.37 $trust^{individual}_{weighted}$

We express the $trust^{individual}_{weighted}$ *metric* using the *weighted sum model expression.*



In a similar way we can model the confidence assessments regarding a sensor's relationship with others in the same location cluster.

$$trust^{group}_{weighted}(Sensor) = \sum w_i \times c^{group}_i \tag{6.2.13}$$

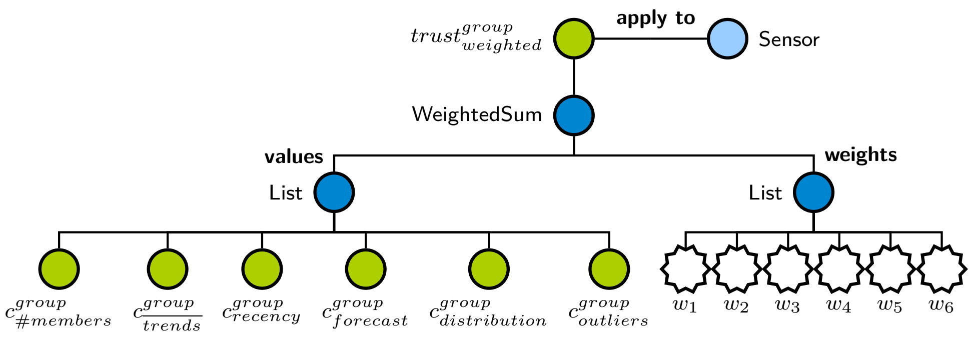
where w_i is the assigned weight with $\sum w_i = 1$ and c^{group}_i one of the following individual sensor *metrics*

$c^{group}_{\#members}$	the confidence in the number of group members
$c^{group}_{\underline{trends}}$	the confidence in the trend/slope comparison
$c^{group}_{recency}$	the confidence in the recency of time series values
$c^{group}_{forecast}$	the confidence in the forecast of time series values
$c^{group}_{distribution}$	the confidence in the distribution of time series values comparison
$c^{group}_{outliers}$	the confidence in the outliers in data of the group's time series values

and can be expressed using the following *expression.*

Expression 6.2.38 $trust^{group}_{weighted}$

We can model the $trust^{group}_{weighted}$ *metric* using the *weighted sum model expression.*



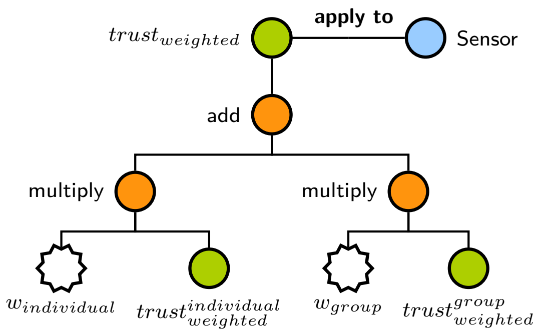
The combination of both of these assessments can then be captured as follows:

$$trust_{weighted}(Sensor) = w_{individual} \times trust^{individual}_{weighted} + w_{group} \times trust^{group}_{weighted} \quad (6.2.14)$$

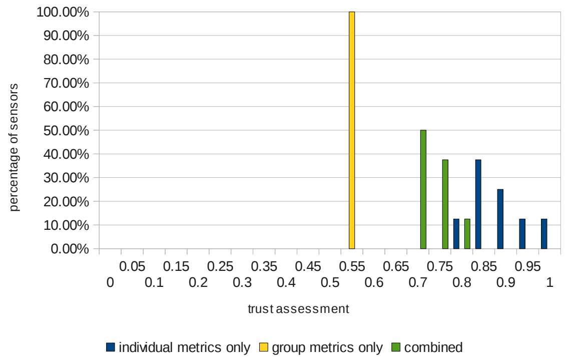
where $w_{individual} + w_{group} = 1$ is the assigned weight to the respective trustworthiness assessment.

Expression 6.2.39 $trust_{weighted}$

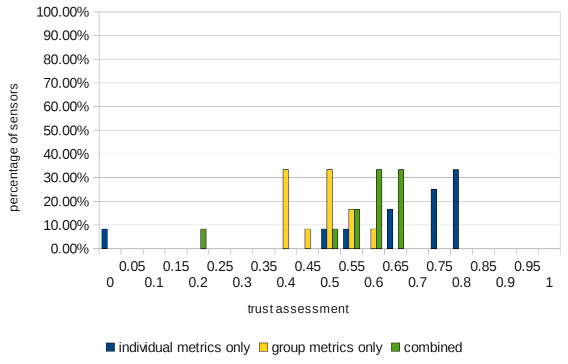
The $trust_{weighted}$ *metric* can be expressed as a simple mathematical weighted sum of the trust *metrics.*



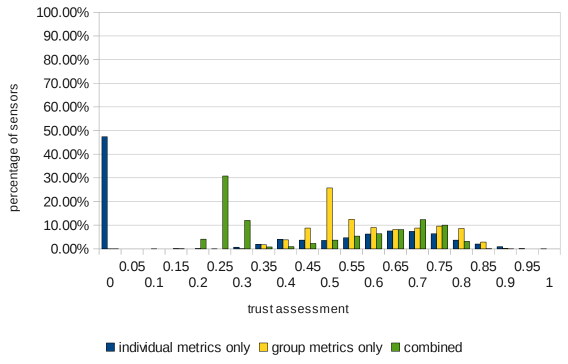
The results of the individual, group, and weighted evaluation are shown in fig-



(a) IAEA fixed monitoring post



(b) IAEA transportable sensor



(c) Safecast

Figure 6.2.25: Weighted trustworthiness assessment by data source overview

ure 6.2.25. In general, the combined version of our trust assessment ranks between only using individual *metrics* and only using location cluster *metrics*. Note that sensors from the fixed monitoring post tend exhibit high levels of trust whereas data from the transportable sensor data set shows reduced trustworthiness. Sensors derived from the Safecast data source display a mix of trustworthiness. This follows the intuition that information that is from a non-government, collaborative sensing environment is less trustworthy than official. Furthermore, this is validated through our analysis which shows that radiation measurements are more likely to be taken in short bursts (i.e., few measurements) than over the long term (i.e., many measurements).

6.2.5.4 Determining Trust Classes Using Thresholds

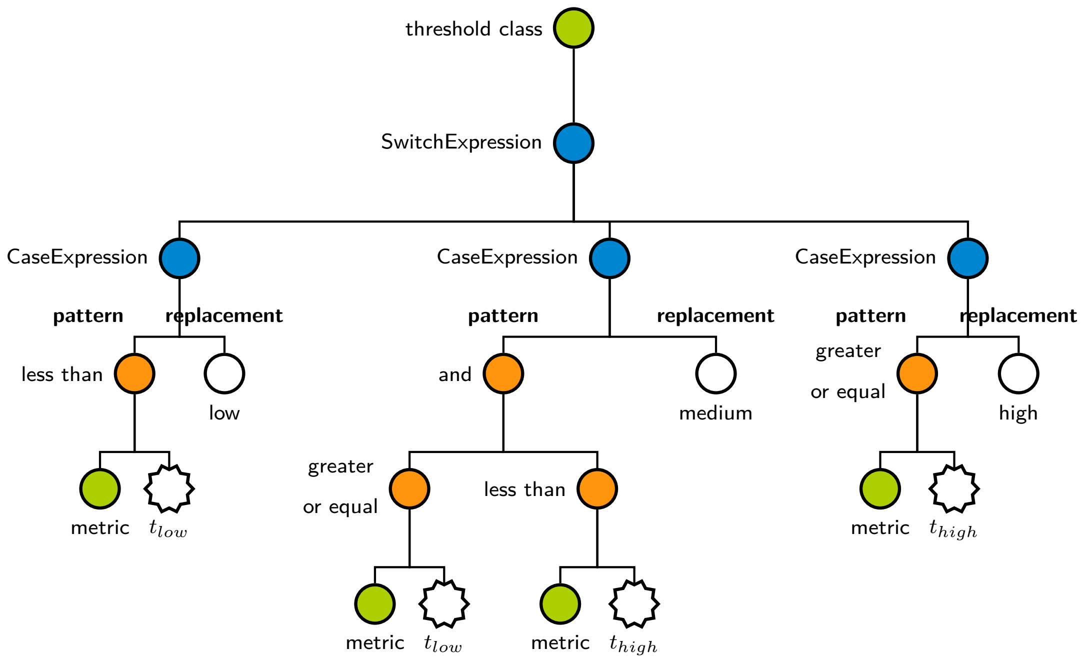
Another approach to assess the trustworthiness of sensors is by moving away from specific values to classes. This makes it easier to quickly get an overview of trustworthiness assessments. Here, we categorize the trustworthiness of sensors into the distinct classes *low*, *medium*, and *high* to represent sensors that fall within specific thresholds. Hence, we need to define a mapping function such as the following that given a particular confidence assessment will yield an appropriate class.

$$trust_{threshold}(metric) = \begin{cases} low & \text{if } metric < t_{low} \\ medium & \text{if } metric \geq t_{low} \text{ and } metric < t_{high} \\ high & \text{if } metric \geq t_{high} \end{cases} \tag{6.2.15}$$

Note that t_{low} and t_{high} represent the assessment thresholds used to distinguish between classes. As part of our framework we are able to transform the confidence assessments discussed above into classes using a variety of *expressions.*

Expression 6.2.40 threshold class

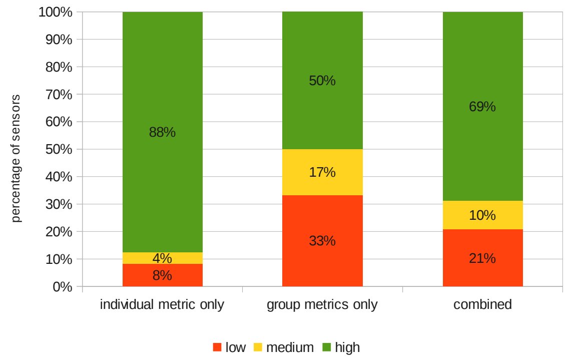
We are able to convert values to classes utilizing the *switch model expression* where each threshold class represents a *case model expression.*



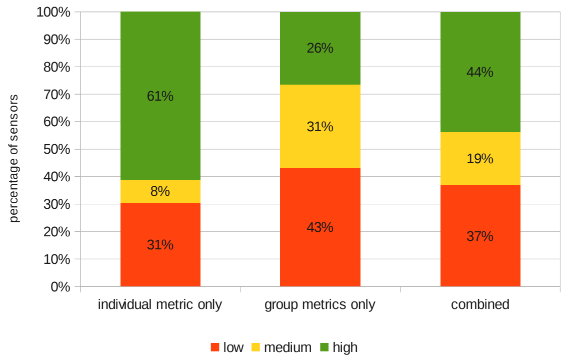
Applying the *threshold class metric* to all confidence assessment results in the trust assessments shown in figure 6.2.26. We can see that there is often a stark contrast between the results when incorporating only individual *metrics* versus group *metrics*. In particular, for the fixed monitoring post individual *metrics* generally rate the trustworthiness of sensors higher than is the case when considering group *metrics*. This is similar for the transportable sensor data set. However, for the Safecast data set group *metrics* seem to assess sensor higher. The combination of the individual and group *metrics* represents the average (i.e., arithmetic mean) of the two. Note that these combined distributions follow our intuition that sensors from the fixed monitoring post data set rank higher than from the transportable sensor and Safecast data set.

6.2.5.5 Impact of Ownership on Trust

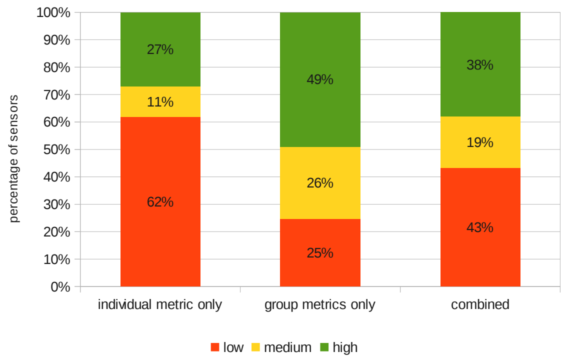
Since there are two distinct authorities providing radiation measurements it is of interest to evaluate how varying levels of ownership trust affects trustworthiness assessments of sensors. Our framework is able to incorporate different weights based on sensor ownership through means of lineage of sensor type.



(a) IAEA fixed monitoring post



(b) IAEA transportable sensor

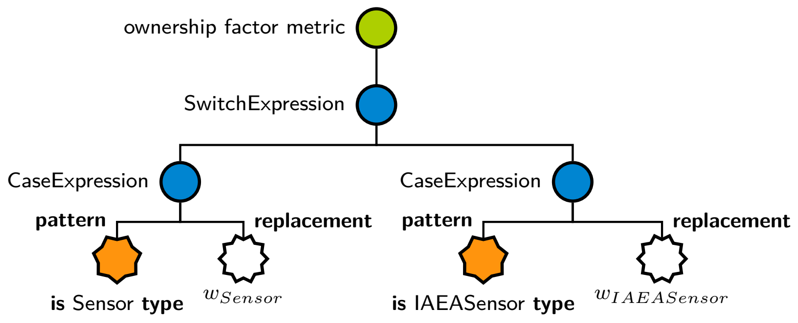


(c) Safecast

Figure 6.2.26: Threshold trustworthiness assessment distributions by data source overview with $t_{low} = 33\%$ and $t_{high} = 66\%$

Expression 6.2.41 ownership factor metric

We express the factor that is used as a weight in evaluating *graph expression* using a *switch model expression*. The *case model expressions* represent the two ownership types which in the *abstract graph model* of this scenario is incorporated as the type of an *element node*.

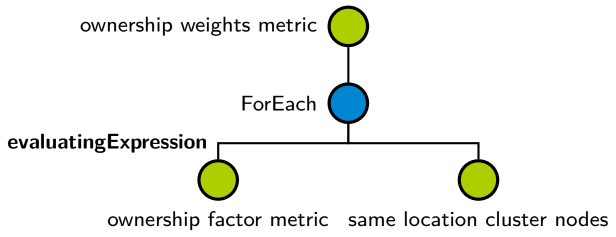


These weights only factor into a confidence assessment that is based on comparing attributes of a sensor to others. This is the case for the location cluster *metrics*. However, two of the *metrics*, $c_{\#values}$ and $c_{outliers}$ are independent of ownership in their calculation. This leaves four *metrics* for which we will evaluate the impact of ownership.

In order to derive the respective weights for the location cluster *metrics* we can use the following *expression*.

Expression 6.2.42 ownership weights metric

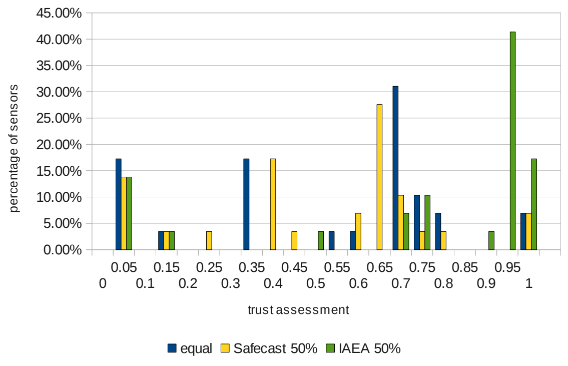
We can model the list of weights as the result of applying the *ownership factor metric* to each of the *element nodes* in the same location.

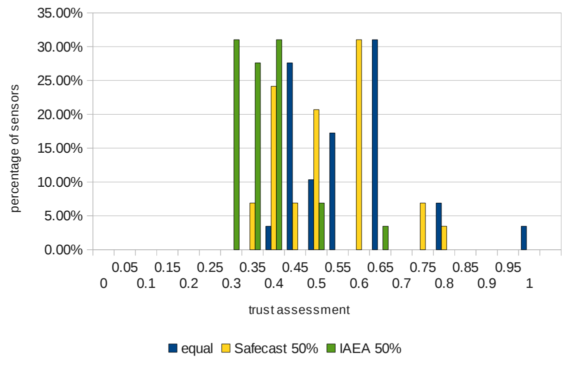


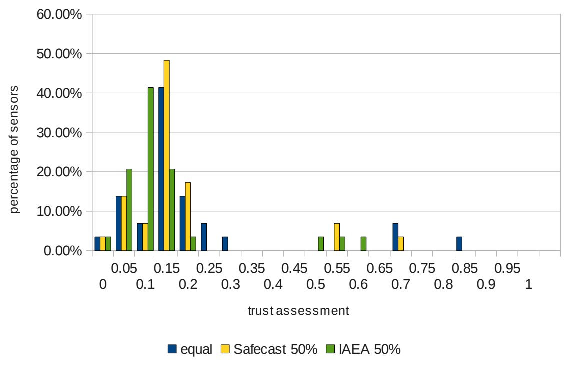
Instead of computing the simple average of the confidences for the location cluster *metric expressions* we then need to replace the *average math expression* with a weighted by ownership average *expression*.

Expression 6.2.43 ownership adjusted average metric

The *values* of the *weighted sum model expression* reflect the *for each model expression* of the location cluster *metrics* which is then combined with the *ownership weights* list. The weighted average is then simply a result of


(a) trend/slope comparison


(b) recency


(c) forecast

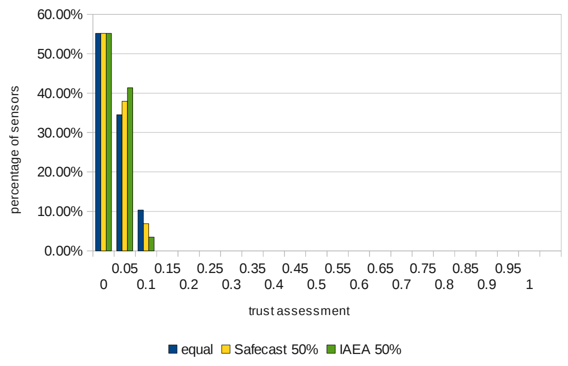
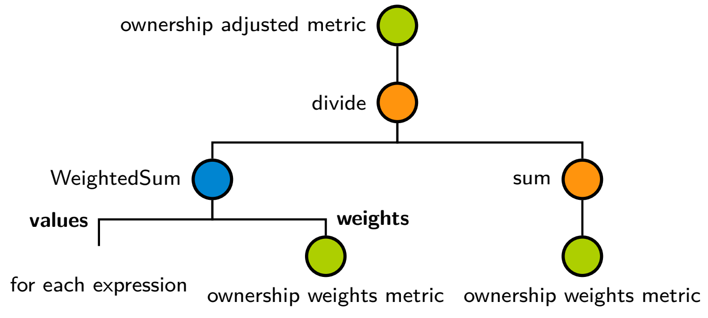

(d) distribution comparison

Figure 6.2.27: Comparison of location cluster trustworthiness assessment metrics based on weighting ownership

the weighted sum divided by the sum of the *ownership weights*.



Because of the limited number of IAEA sensors provided there is only one location overlap with Safecast sensors. Hence, in figure 6.2.27 we show results of the location cluster *metrics* in that particular location. In order to analyze the impact of different ownership we compare three cases. First, equal weights between Safecast and IAEA sensors make no distinction in terms of their trustworthiness based on ownership. Second, we discount the trustworthiness of Safecast sensor by 50% in order to favor IAEA government sensors. Third, the 50% discount of trustworthiness is applied to every IAEA sensor and the Safecast data is favored.

For trend/slope comparison the different ownership preferences yield mixed results. In particular, when the Safecast data is favored trustworthiness assessments go up while favoring IAEA sensors provides a wide distribution of trustworthiness assessments. This is the quite the opposite for the *recency metric* where preferring Safecast sensors results in lower trustworthiness assessments. For the *forecast metric* as well as the *distribution comparison* ownership does not seem to have the impact we hoped for as it does not provide a clear enough separation for the trust assessments.

In general, incorporating ownership seems to have mixed results. While for some *metrics* discounting Safecast improves the average trust assessments, for others it actually decreases them. This is different from the results that we saw for the threshold based trust classes approach.

6.2.6 Radiation Detection: Summary

Sensor networks can be used in a variety of applications such as weather monitoring, cargo tracking, and radiation detection. Here we analyzed a collaborative sensing environment related to the Japan Fukushima nuclear plant incident in 2011. Our *Trust-KnowOne* framework is not only able to model the data from various radiation level data sources but also process and evaluate it in light of several trustworthiness assessment approaches.

Throughout this section we discussed in detail how heterogeneous data, relationships and different data sources can be modeled using our *abstract graph model*. Furthermore, we incorporated the fact that sensors depending on whether they are mobile or stationary exhibit should be trusted differently. This scenario also shows how time series data can be stored, managed and processed using *graph expressions*. We also described a complex transformation of *graph components* through the application of density-based clustering that resulted in sensors being derived from simple radiation measurements.

The assessment of sensor attributes was performed in two ways. First, we presented several *metrics* that only incorporate individual attributes of a sensor. Second, we provided group *metrics* that take into consideration that sensors within the same location cluster should be related and have similar attributes (e.g., time series values, trends).

In terms of evaluating the overall trustworthiness of sensors based on the individual and group *metrics* derived throughout the *knowledge processing* phase of our framework, we discussed three approaches. First, a basic weighting scheme of individual and group *metrics* can be used to assess the trustworthiness of a sensor from a pure analytical standpoint. Second, we proposed a method to transform trustworthiness assessments into trust classes such that it becomes easier to determine the state of a sensor as well as groups of sensors. Finally, we discussed what impact ownership can have on the overall trustworthiness assessment of sensors.

6.3 Trust in Collaborative Intrusion Detection Networks

Intrusion detection is a growing field in the the area of computer networks. As data has become more distributed across a diverse set of resources, one faces the challenge of providing reliable access to data as well as protecting it from a variety of security threats. Additionally, resources are often diverse in terms of hardware (e.g., network interfaces, processing capabilities, memory and storage availability), operating systems (e.g., Windows, MacOS, Linux, etc.) and software (e.g., applications, tools) components. This is due to the fact that systems need to be flexible enough to serve a large number of users each with their own requirements. However, it is exactly this provided flexibility that makes it difficult to implement data protection mechanisms and enforce security policies.

Instead of having to constantly adapt and extend security systems to work across different platforms and configurations, the focus has shifted towards utilizing agent-based approaches [9, 68, 82]. In particular, resources are being equipped with sensors on the hardware or software level that constantly monitor the state of the resource and possibly connected components. They then often utilize a common security protocol for event and alert notification. This approach has several advantages such as being easily extendable and reducing management requirements.

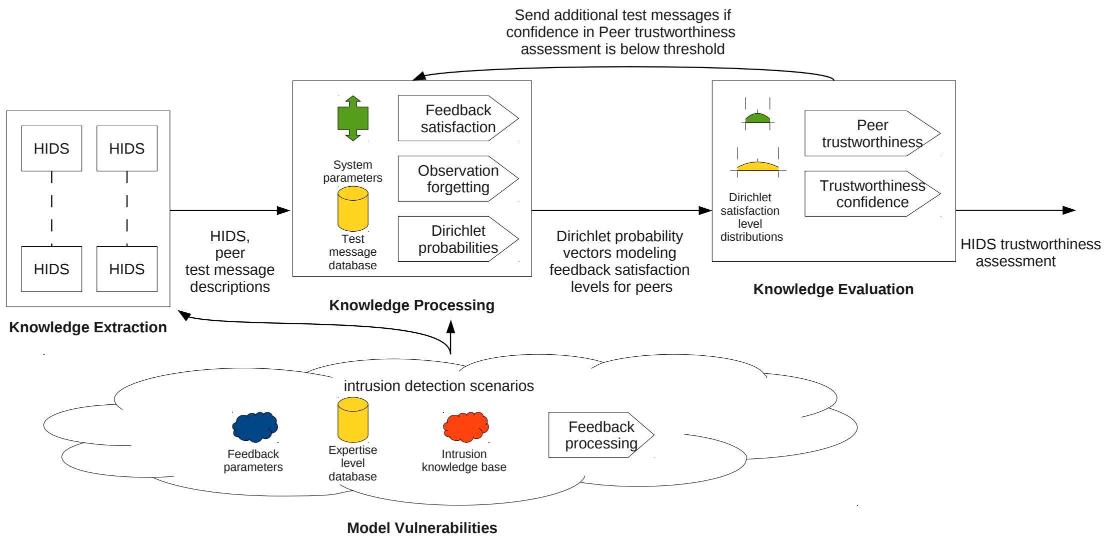
Several collaborative trust modeling and management systems have recently been proposed [110, 136, 140]. These systems have been shown to be able to deal with a variety of attacks (e.g., sybil, newcomer, betrayal, collusion, inconsistency [54]). However, the approaches taken on the hardware, software or protocol layers often differ drastically. This makes the comparison of their complexity and performance difficult. Furthermore, it requires significant effort to adapt (i.e., exchange one particular component or algorithm for another) and extend (i.e., implement additional monitoring and management capabilities) these systems.

6.3.1 Intrusion Detection: Framework Modeling Approach

We have developed a framework to model various entities and their relationships in an abstract graph model on which graph expressions can be evaluated. In particular, our approach incorporates the trustworthiness and quality of data. This makes it well suited for the intrusion detection domain. In order to show that it is feasible to use our framework for the modeling, analysis and evaluation of intrusion detection approaches, we will discuss a collaborative trust management system for intrusion detection by Fung et al. [54] as a case study. The system makes use of several heterogeneous entities and relationships. Furthermore, it is fully dynamic as new observations are incorporated over time into the trustworthiness assessment of entities changes.

Here we discuss how our framework can be used to evaluate attacks on the knowledge derivation process. To test their approach Fung et al. [54] simulate the responses to test messages probabilistically. To generate sensed intrusion data we also incorporate a simulator into our framework. There are two distinct options for modeling attacks on knowledge. First, we can incorporate attacks such as malicious nodes into the data before the *knowledge extraction* phase. This is the classical approach which allows all data to be processed and evaluated in a “static” manner where graph components do not change after the initial *knowledge extraction.* Second, the basic scenario can be described in generic terms for the *knowledge extraction* phase and attacks can be modeled as part of the *knowledge processing* phase. This approach has the advantage that attacks are formally specified parts of the framework. Hence, graph components that are part of the *element instance graph* are “dynamic” and can be added, modified or removed according to specific types of attacks. For the scenario discussed here, we choose to incorporate the modeling of attacks into our framework in order to enable a discussion of the following aspects of the *TrustKnowOne* framework.

- Modeling dynamic entities (Host-Based Intrusion Detection components, peers, test messages) and relationships in our *abstract graph model* (section 4.1)
- Formalizing components of an intrusion detection system as *belief engines* (section 5.3.1)
- Representing trustworthiness assessments as formal *decision processes* that incorporate confidence assessments (section 5.4.1)
- Evaluating the impact of attacks on knowledge derivation using robustness measures represented by *graph expressions* (section 5.5)



Send additional test messages if confidence in Peer trustworthiness assessment is below threshold
HIDS
HIDS
HIDS
HIDS
Knowledge Extraction
HIDS, peer test message descriptions
Feedback satisfaction
System parameters
Observation forgetting
Test message database
Dirichlet probabilities
Knowledge Processing
Dirichlet probability vectors modeling feedback satisfaction levels for peers
Peer trustworthiness
Dirichlet satisfaction level distributions
Trustworthiness confidence
Knowledge Evaluation
HIDS trustworthiness assessment
intrusion detection scenarios
Feedback parameters
Expertise level database
Intrusion knowledge base
Feedback processing
Model Vulnerabilities

Figure 6.3.1: Framework overview for the trust in collaborative intrusion detection scenario

Figure 6.3.1 shows an overview of how the scenario can be modeled using our framework. As such, we are able to describe in detail how our framework is able to model the approach by Fung et al. [54].

Knowledge Extraction This phase consists of a general description of the components for the collaborative trust management approach of Fung et al. [54]. In particular, we provide the foundation for adding intrusion detection components such as hosts, peers, and test messages as well as the relationships between them. As a result, the *element description graph* is fully specified and the *element instance graph* remains empty.

Knowledge Processing As part of a particular intrusion detection scenario graph components can be added, modified, and removed from the *element instance graph*. Which components and relationships this entails is dependent on system parameters (e.g., number of satisfaction levels, test message generation rate) and the type of attacks performed. Note that we can assess the trustworthiness and data quality aspects proposed by [54] using formal *graph expressions* dynamically as the system evolves in time (enabled by simulator) as well as at a specified time or state. This gives us the ability to evaluate and compare approaches as well as analyze system parameters.

Knowledge Evaluation There are several decisions that can be made using the intrusion detection data modeled in our framework. For instance, we can determine which hosts are trustworthy based on thresholds or other decision processes. Additionally, given the trustworthiness assessments provided by the *knowledge processing* phase we are able to compare patterns against different types of attacks and provide an estimate of how likely it is that a particular attack occurred.

Model Vulnerabilities Throughout the phases of the knowledge derivation process we can analyze the impact of particular type of attack. Specifically, given a specific scenario (i.e., *system parameters* fully defined) we can decide whether a type of attack has occurred and determine how it affected specific intrusion detection components. Furthermore, our framework provides the formal means for performing these robustness checks and as such can be utilized to improve and optimize intrusion detection approaches against a variety of attacks. In a similar manner, different approaches can be compared in terms of their complexity (i.e., *graph expressions*), performance (i.e., time and *graph components* necessary to reach decision), and robustness (i.e., performance during attacks).

In the next sections we provide a detailed discussion of the individual *TrustKnowOne* framework components as the intrusion detection scenario relates them.

6.3.2 Intrusion Detection: Knowledge Extraction

Collaborative intrusion detection systems operate on the basis of building relationships between individual monitoring agents or intrusion detection systems. As such, Fung et al. [54] propose a system in which individual host-based intrusion detection

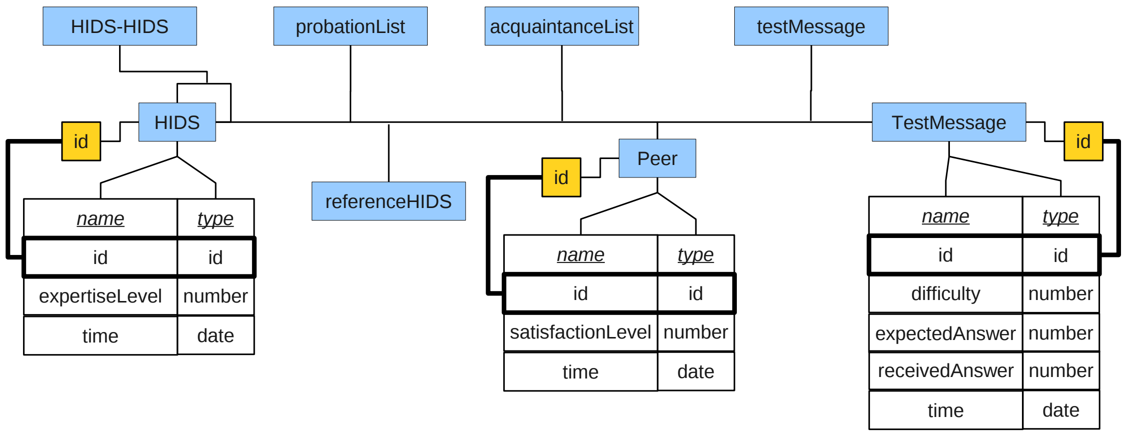


Figure 6.3.2: Intrusion detection *element description graph*

systems (HIDS) collect information about peers through a series of test messages (i.e., challenges which are often in the form of knowledge base questions). Test messages consist of an associated difficulty (i.e., the ability of a peer responding correctly), expected answer (i.e., knowledge base classification of alert risk)[3], and received answer (i.e., the actual classification by a peer). Peers then provide feedback (i.e., responses) based on their respective expertise level. Each HIDS is able to manage its set of peers dynamically in the form of an *acquaintance list* for trustworthy peers and a *probation list* for those that are less trustworthy. As more information is gathered about the peers they can be promoted from the *probation list* to the *acquaintance list* or discarded.

Using the *TrustKnowOne* framework we model this intrusion detection system as shown in figure 6.3.2. The individual hosts are described by the *HIDS element*. The main component of information that needs to be modeled is the *expertise level* which reflects the *HIDS*' ability to respond to test messages. Fung et al. [54] describe their collaborative system as potentially fully connected. This means that there exists a *relation* between all *HIDS* without any constraints, i.e., no *metric* attached.

Each *HIDS* keeps track of the perceived trustworthiness of its peers. We model this

[3]Note that Fung et al. [54] use the term alert risk and expected answer interchangeably. However, most of the formulas refer to expected answer which is why we chose expected answer in our discussion.

trustworthiness by introducing a separate *Peer element* that contains a time series of *satisfaction levels*. The details of how these *satisfaction levels* are derived from test messages is discussed in the next section. The *Peer element* is a reflection of how a *HIDS* perceives other *HIDS elements*. Hence, we introduce a *reference HIDS relation* that correlates each *Peer* with the *HIDS* whose trustworthiness is being assessed. A *HIDS* manages these *Peers* in two separate lists, a probation list and an acquaintance list depending on perceived trustworthiness. We model both of these lists as *relations* between a *HIDS* and a *Peer*. The reason Fung et al. chose this approach is that it is necessary to bound the complexity of the relationships between *HIDS* and *Peer elements*. Note that the existence of a *relation* description only implies that there is the potential for a *relation edge* to be present in the *element instance graph*. Thus, without limiting the number of *Peers* each *HIDS* is evaluating, we would see an exponential growth of relationships (*relation edges* in our model) that we need to keep track of, specifically

$$\text{number of } \textit{relation edges} = n \times (n-1) \tag{6.3.1}$$

compared to a linear bound for the list approach used here that results in

$$\text{number of } \textit{relation edges} = (|P| + |A|) \times n \tag{6.3.2}$$

where n is the number of *HIDS elements* and $|P|$ and $|A|$ the size of the probation and acquaintance lists respectively. Fung et al. [54] discuss the size limits of these lists in terms of resource constraints which means that their approach defines a combined maximum for both, i.e., $|P| + |A| \leq size_{max}$.

In order to determine the trustworthiness of *HIDS elements*, a *HIDS* will send out test messages which we describe in our model as *TestMessage elements*. These *TestMessage elements* are associated with a certain level of *difficulty* which directly impacts the ability of another *HIDS* to respond with the *expected answer*. As we will see later, Fung et al. [54] actually abstract the generation and response of these

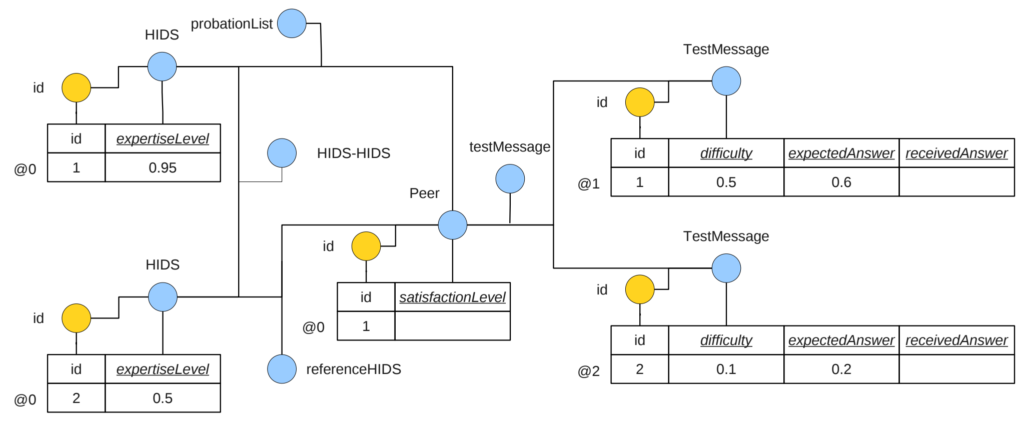


Figure 6.3.3: Intrusion detection *element instance graph*

test messages for evaluation purposes. As part of this abstraction real world intrusion detection test messages are replaced by test messages consisting of tuples of numbers between 0 and 1 for keeping track of a test message's difficulty, expected answer and received answer. This also simplifies the modeling process in our framework where instead of having separate *request* and *response elements* we can use one *TestMessage element* which includes the *received answer* as an additional attribute. Because the trustworthiness of a *HIDS* is modeled in terms of *Peer elements* we form a *relation* between a *Peer* and *TestMessage elements*.

Note that all *elements* contain an *id* as well as a *time attribute* in order to uniquely identify individual *element nodes* in our graph model and to allow for dynamic changes of *attribute* values as the process evolves.

In figure 6.3.3, we show a small example of a possible *element instance graph* derived from the *element description graph* we described. The graph consists of two *HIDS element nodes* that are related (*HIDS-HIDS relation edge*). Both *HIDS elements* have a different expertise level for responding the test messages. *HIDS 1* models the trustworthiness of *HIDS 2* in terms of *Peer 1* where *Peer 1* is in the probation list (*probationList relation edge*) of *HIDS 1* and references *HIDS 2* (*referenceHIDS relation edge*). Initially, the trustworthiness of *HIDS 2* is based on some a priori trustworthiness because test messages have not been assessed and hence there is no history of satisfaction levels from which trustworthiness could be derived. We also show in figure 6.3.3 two *TestMessage elements* each with varying degrees of difficulty, expected answers and request times.

6.3.3 Intrusion Detection: Knowledge Processing

In this section we will describe in detail how the framework proposed here is used to determine trustworthiness aspects of a *HIDS* based on the approach by Fung et al. [54]. In particular, we will discuss how our framework is able to perform the same assessments while utilizing a flexible abstract graph model. There are several advantages of modeling various trust management approaches in a unified framework such as the ability to compare and evaluate performance of the approach as a whole and on an individual component level. Furthermore, this framework enables reuse, reconfiguration, and extension as well as replacement of components across approaches.

6.3.3.1 Evaluating Feedback

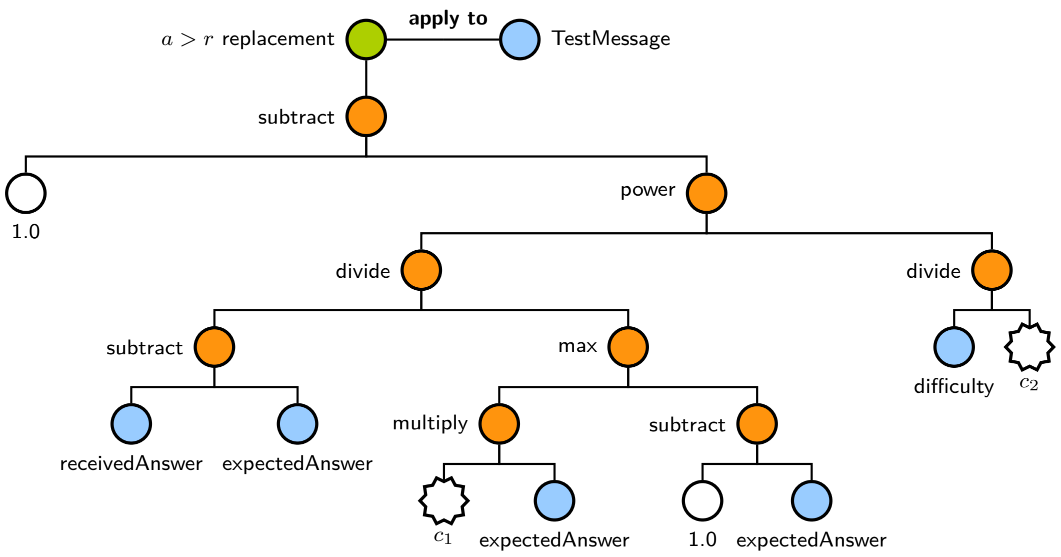
One of the primary components of collaborative trust management concerns the evaluation of the responses to test messages. Fung et al. [54] define a mapping function $Sat(r, a, d) \in [0, 1]$ which determines a satisfaction level based on the difficulty, expected answer, and the received answer as

$$Sat(r, a, d) = \begin{cases} 1 - \left(\dfrac{a - r}{max(c_1 r, 1 - r)}\right)^{\dfrac{d}{c_2}} & a > r \\ 1 - \left(\dfrac{c_1(r - a)}{max(c_1 r, 1 - r)}\right)^{\dfrac{d}{c_2}} & a \leq r \end{cases} \tag{6.3.3}$$

While r, a, and d represent expected answer, received answer, and difficulty respectively, c_1 and c_2 are user selected system parameters. Each *TestMessage element node* contains these values in the form of *attributes.* One needs to distinguish between two cases. In order to maintain flexibility we model both result terms as separate *metrics*, $a > r$ and $a \leq r$ whose *expression trees* are as follows.

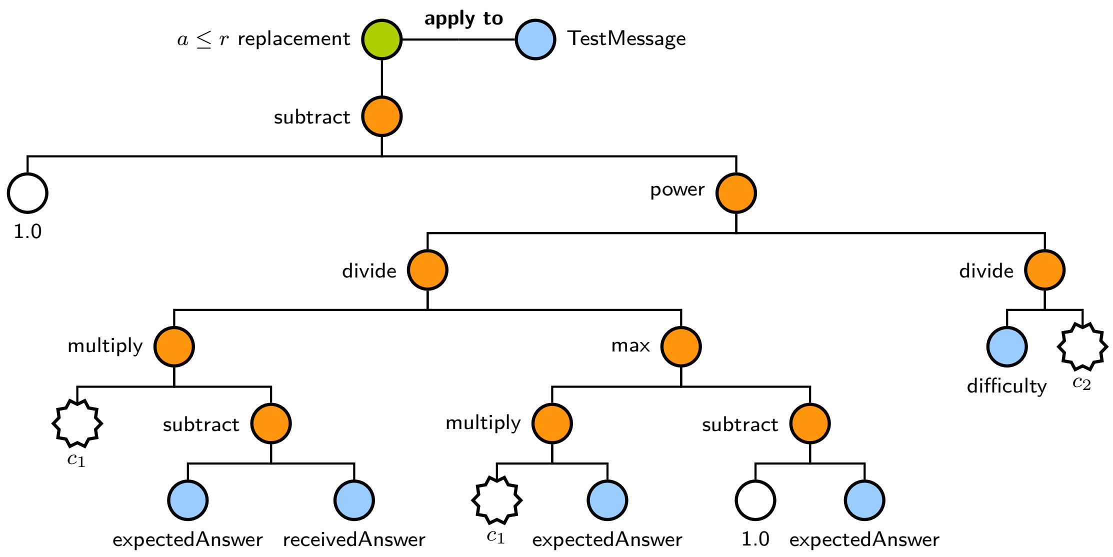
Expression 6.3.1 $a > r$ replacement

The $a > r$ replacement *metric* is modeled using basic mathematical *expressions* on *attributes* and *system parameters*.



Expression 6.3.2 $a \leq r$ replacement

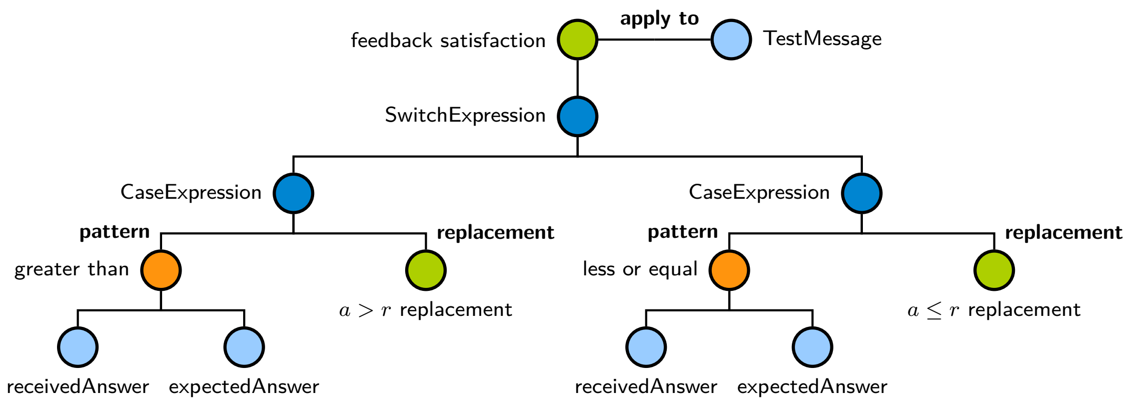
The $a \leq r$ replacement *metric* can be expressed using basic mathematical *expressions* on *attributes* and *system parameters*.



Our framework supports a *switch model expression* that takes *case model expressions* as parameters. This approach is similar to one that would be used in a regular programming language. For each *case model expression*, the *pattern* parameter is evaluated and if *true* the current *expression* is replaced by the *replacement* parameter. Since *case model expressions* are evaluated in sequence, the evaluation stops when one of the *pattern* parameters is *true*. We model the entire process of determining the satisfaction of a test message as the *feedback satisfaction metric*.

Expression 6.3.3 feedback satisfaction

The feedback satisfaction *metric* makes use of the *switch model expression* and according *case model expressions* that reflect the cases as well as the $a > r$ and $a \leq r$ replacements defined by equation 6.3.3 .



The result is inserted into the satisfaction level attribute of the *Peer element node* thus creating a time series of test message feedback satisfaction levels.

6.3.3.2 Probabilistic Representation of Observations

The basis of performing trustworthiness assessments on each of the *HIDS* used by Fung et al. [54] is the Dirichlet distribution. It can be used to model the probability of a set of discrete states S based on concentration parameters α which in this case are updated as more data becomes available. Fung et al. [54] chose to divide the possible satisfaction levels defined in equation 6.3.3 into k number of ranges (i.e., 10 by default ranging from 0.1 to 1.0). Here, the discrete states of the Dirichlet distribution represent various satisfactions level ranges $s_1, \ldots, s_k$ and the α parameters are modeled as the combination of initial beliefs and subsequent observations of satisfaction levels $\overrightarrow{\gamma} = \gamma_1, \ldots, \gamma_k$. Given that the probability of $p_i = P(S = s_i)$ from [54] we have

$$Dirichlet(p_1, \ldots, p_k | \gamma_0, \gamma_1, \ldots, \gamma_k) = \frac{\Gamma(\sum_{i=1}^{k} \gamma_i)}{\prod_{i=1}^{k} \Gamma(\gamma_i)} \prod_{i=1}^{k} p_i^{\gamma_i - 1} \tag{6.3.4}$$

where

$$\gamma_0 = \sum_{i=1}^{k} \gamma_i \tag{6.3.5}$$

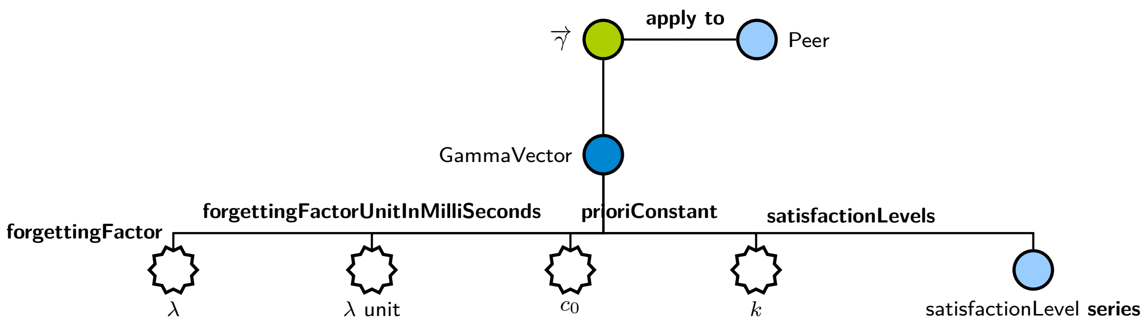
In order to improve the impact of recent observations, a forgetting factor is introduced which discounts prior observations based on the time difference between the previous observations and the newest observation. The combination of the initial belief, the series of observations, and the forgetting factor results in the observation vector $\overrightarrow{\gamma}^{(n)}$ at a particular time n which is recursively defined by Fung et al. [54] as

$$\overrightarrow{\gamma}^{(n)} = \begin{cases} c_0 \overrightarrow{S}^0 & n = 0 \\ \lambda^{\triangle t_n} \times \overrightarrow{\gamma}^{(n-1)} + \overrightarrow{S}^n & n > 0 \end{cases} \tag{6.3.6}$$

where c_0 is a priori constant used for the initial beliefs represented by $\overrightarrow{S}^0$, $\lambda^{\triangle t_n}$ the discounted forgetting factor[4], and $\overrightarrow{S}^n$ the discretized satisfaction level at time n. The observation vector yields the basis for modeling the probability distribution of satisfaction levels at specific times using the Dirichlet distribution. While the time dynamic observation vector approach could have been implemented using a combination of basic *expressions*, we decided to create a separate reusable *gamma vector model expression.*

Expression 6.3.4 Observation vector $\overrightarrow{\gamma}$
The $\overrightarrow{\gamma}$ *metric* can be represented using the *gamma vector model expression* which parameters include the forgetting factor, its unit, the priori constant and the number of satisfaction levels as system parameters λ, c_o, and k respectively as well as a time series of satisfaction levels.

[4]Note that Fung et al. [54] do not explicitly state the unit of this forgetting factor. As such, we assume the factor is per day.



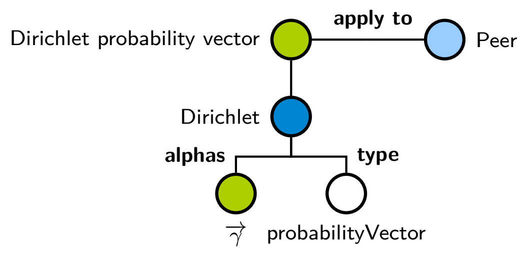
This is an example which shows that complex techniques and formulas can be abstracted as *model expressions* with parameters that can be reused more easily and incorporated into other models.

The combination of initial belief and a time series of satisfaction level observations adjusted using a forgetting factor forms the basis of the Dirichlet concentration parameters. As shown above we model the evolution of the prior for the Dirichlet distribution from a time series of satisfaction levels as the $\overrightarrow{\gamma}$ *metric*. This allows us to use a standard *model expression* for a Dirichlet distribution where we use the observation vector $\overrightarrow{\gamma}$ in place of the regular α parameters.

In general, we model probability distributions as *model expressions* with all necessary parameters (i.e., set of α for Dirichlet, α and β for Beta distributions, etc.) and include a *type* parameter that specifies the result of the *expression* such as probability density or cumulative probability. As for the probabilistic representation of observations and satisfaction levels, we define the *Dirichlet probability vector metric*.

Expression 6.3.5 Dirichlet probability vector

The Dirichlet probability vector *metric* is modeled as the *Dirichlet distribution model expression* using the $\overrightarrow{\gamma}$ *metric* as a basis (α concentration parameters) for deriving a distribution of satisfaction levels in form of a probability vector.



6.3.4 Intrusion Detection: Knowledge Evaluation

Now that we can derive the Dirichlet probability vector representing the distribution of satisfaction levels for individual *Peers*, we are able to assess their trustworthiness. Fung et al. [54] define the trustworthiness of a *Peer* by utilizing the weighted sum of this probability vector

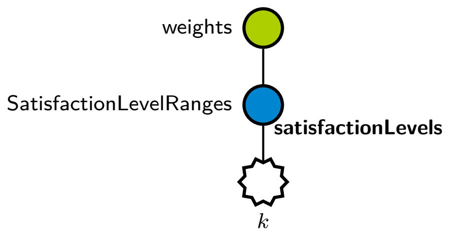
$$T(peer) = \sum_{i=1}^{k} w_i E[p_i] = \frac{1}{\gamma_0} \sum_{i=1}^{k} w_i \gamma_i \tag{6.3.7}$$

where k is the number of satisfaction levels, $\gamma_0 = \sum \gamma_i$, and $E[p_i]$ the expected probability of a particular satisfaction level given the Dirichlet distribution (equation 6.3.4) for the selected *Peer*.

Note that the weights used in the trustworthiness assessment are simply the satisfaction level ranges. For example, if $k = 10$ then the weights vector would be $\{0.1, 0.2, \ldots, 0.9, 1.0\}$. In our framework we can compute these weights by utilizing a custom *model expression*.

Expression 6.3.6 weights

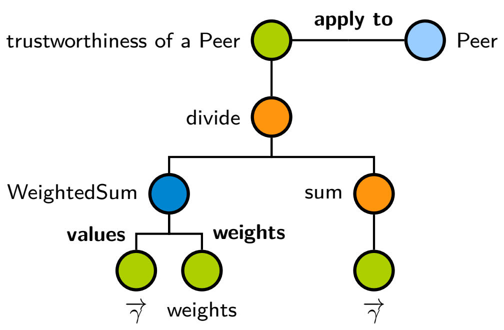
The *weights metric* can be represented using the *satisfaction level ranges model expression* which has as parameter the number of satisfaction levels as system parameter k.



As such the trustworthiness of a particular *Peer* can then be determined using the Dirichlet distribution of satisfaction level observations and the satisfaction level ranges accordingly.

Expression 6.3.7 trustworthiness of a Peer

The trustworthiness of a *Peer metric* is modeled based on equation 6.3.7 as the weighted sum of the *weights* and $\overrightarrow{\lambda}$ normalized by λ_0 which can be expressed by the sum of $\overrightarrow{\lambda}$.


trustworthiness of a Peer
apply to
Peer
divide
WeightedSum
sum
values
weights
$\overrightarrow{\gamma}$
weights
$\overrightarrow{\gamma}$

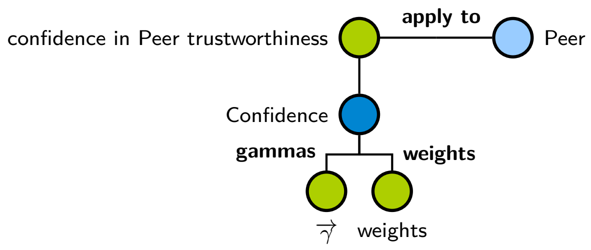
Fung et al. [54] also provide the means to compute a confidence level for this trustworthiness assessment based on the same weights and gamma vector

$$C(peer) = 1 - \frac{4}{\sqrt{1+\gamma_0}} \sqrt{\sum_{i=1}^{k} w_i^2 \frac{\gamma_i}{\gamma_0} - \left(\sum_{i=1}^{k} w_i \frac{\gamma_i}{\gamma_0} \right)^2} \tag{6.3.8}$$

We need to point out that in contrast to trustworthiness which ranges from 0 to 1, the confidence given in equation 6.3.8 ranges from -1 to 1.

Expression 6.3.8 confidence in Peer trustworthiness

The confidence in *Peer* trustworthiness is more complex than the trustworthiness. However, it is based on the same input, *weights* and *gamma vector metrics*. Thus we defined a *Confidence model expression* that reflects equation 6.3.8.



6.3.5 Intrusion Detection: Evaluation of Model Vulnerabilities

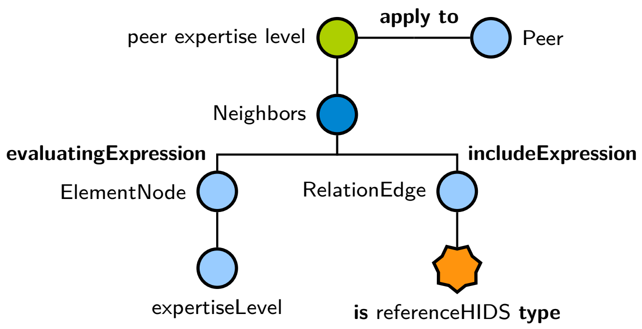
Fung et al. [54] use simulation to generate sensor data (i.e., data for each HIDS) to evaluate their trust assessment approach. Here, we also use simulation to generate sensor data dynamically. In addition we show that this simulation can also be modeled in our framework which enables us to analyze the behavior of the proposed approach in various types of attack scenarios.

6.3.5.1 Generating Test Messages

An important part of evaluating the system is the ability to probabilistically model responses to test messages. Fung et al. [54] base this off the *expertise level* of a *Peer* as well as the *difficulty* and *expected answer* of a particular test message. As described above, *difficulty* and *expected answer* are *attributes* of *TestMessage elements* while the *expertise level* can be retrieved using the *referenceHIDS relation*.

Expression 6.3.9 peer expertise level

The *peer expertise level metric* utilizes the flexible *Neighbors model expression*. This *model expression* has two parameters. First, an *includeExpression* which defines what neighbors should be included in the evaluation process. Here, we set up a *constraint* on the type of relation to *referenceHIDS*. Second, the *evaluatingExpression* is evaluated on the resulting *element nodes* and *relation edges* accordingly. Since we are interested in retrieving the *expertise level* of a *HIDS* we reference its *expertise level attribute*.



The actual test message feedback is modeled as discussed by [54] using the probability density function of the Beta distribution

$$Beta(x|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1} \tag{6.3.9}$$

where $x \in [0,1]$ is the response to the test message. In particular, Fung et al. [54] specify the parameters α and β as follows

$$\alpha = 1 + \frac{l(1-d)}{d(1-l)}\sqrt{\frac{r}{1-r}}\sqrt{\frac{2}{l}-1} \tag{6.3.10}$$

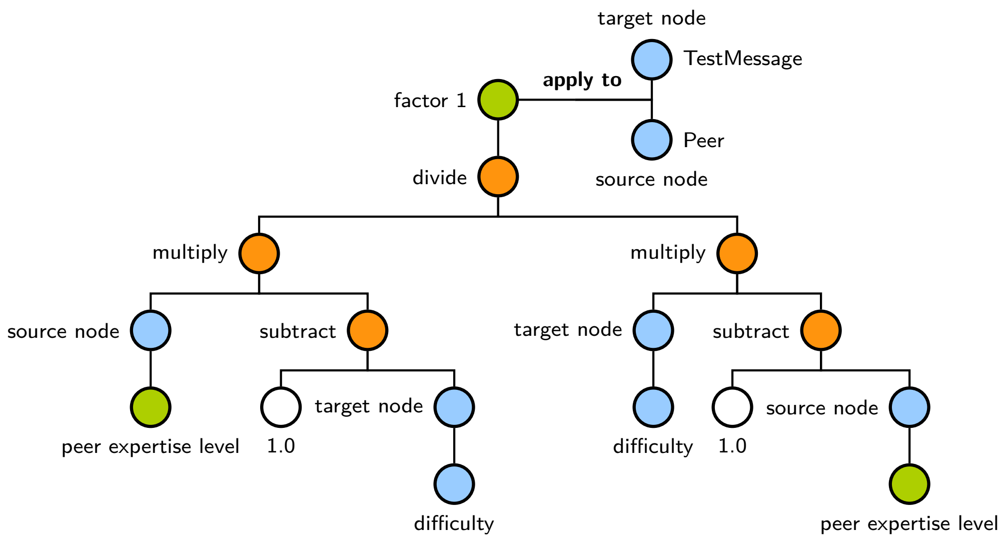
and

$$\beta = 1 + \frac{l(1-d)}{d(1-l)}\sqrt{\frac{1-r}{r}}\sqrt{\frac{2}{l}-1} \tag{6.3.11}$$

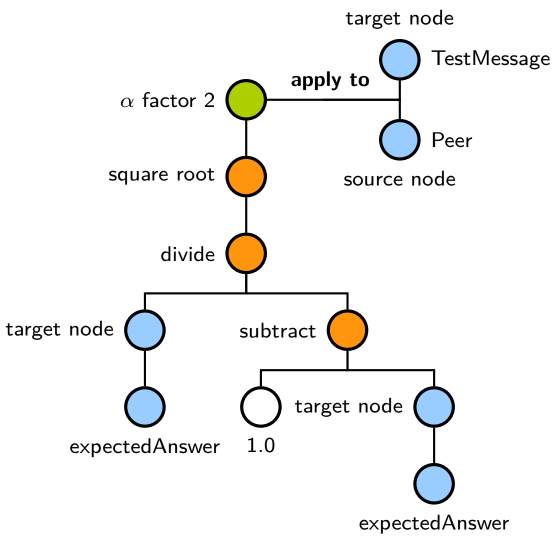
to incorporate the difficulty of the test message d, its expected answer r, and the expertise level of the *HIDS* l. While this represents a complex approach, we use this part to showcase the flexibility of our framework.

Upon closer inspection we can see that both equation 6.3.10 and 6.3.11 are similar except for the second factor. Thus, the calculation of α and β is broken down into individual factors which are represented by *metrics* as follows.

Expression 6.3.10 Beta distribution first factor
The first factor of the Beta distribution $\dfrac{l(1-d)}{d(1-l)}$ can be expressed by a combination of math *expressions* and *attribute references*. Note that, we determine the expertise level by applying the *peer expertise level metric* to the *Peer element node* which is the source node whereas the *difficulty attribute* is derived from the target node of the *test message relation*.

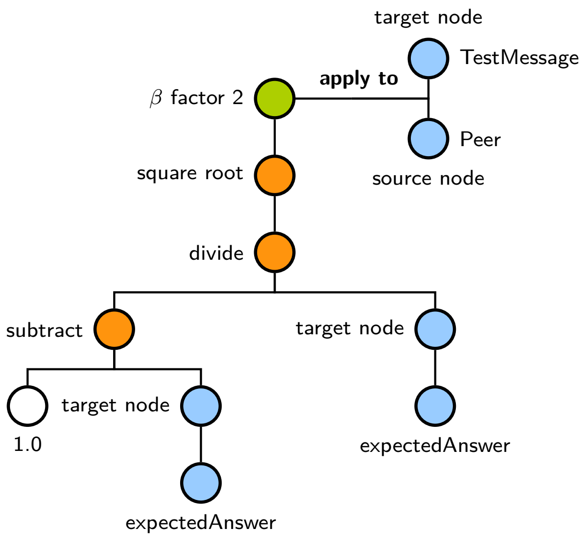


Expression 6.3.11 Beta distribution second factor for α
The second factor of the Beta distribution is $\sqrt{\dfrac{r}{1-r}}$ for the α term. We utilize a series of math *expressions* on the expected answer r to model this.



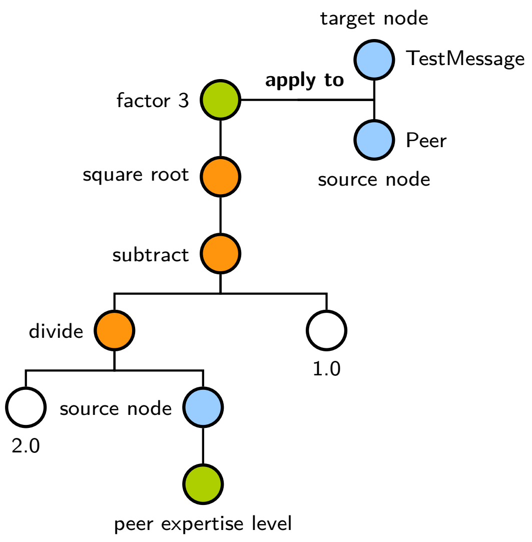
Expression 6.3.12 Beta distribution second factor for β

The second factor for the Beta distribution is $\sqrt{\frac{1-r}{r}}$ for the β term. We can express this is similar terms as the factor for α, a combination of math *expressions*.



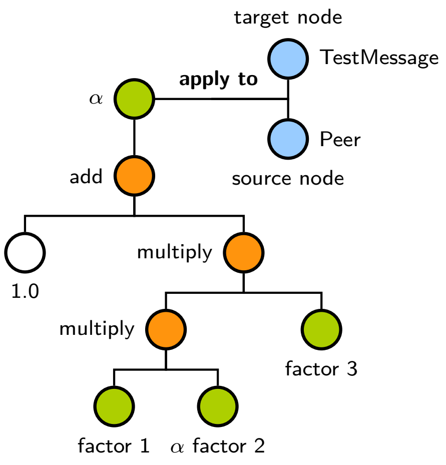
Expression 6.3.13 Beta distribution third factor

The third factor of the Beta distribution $\sqrt{\frac{2}{l}-1}$ is a straightforward computation on the peer expertise level.

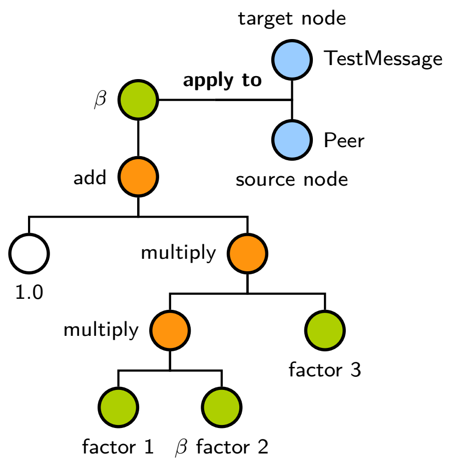


Hence, we can determine the parameters α and β of the Beta distribution accordingly.

Expression 6.3.14 Beta distribution α
The α term $1 + \dfrac{l(1-d)}{d(1-l)}\sqrt{\dfrac{r}{1-r}}\sqrt{\dfrac{2}{l}-1}$ becomes a combination of the previously modeled *expressions*.

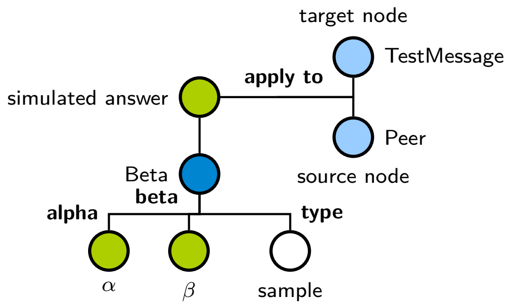


Expression 6.3.15 Beta distribution β
The β term $1 + \dfrac{l(1-d)}{d(1-l)}\sqrt{\dfrac{1-r}{r}}\sqrt{\dfrac{2}{l}-1}$ is modeled using its necessary factor *expressions* accordingly.



The combined α and β *metrics* show the advantage of defining generic reusable *expressions* and *metrics* which make modeling less complex and more efficient. In particular, we utilized the overlapping first and third factors in both *expressions*.

Expression 6.3.16 response to challenge generation
The simulated answer to a test message is based its difficulty, the expected answer as well as the expertise level of the peer. We utilize a *Beta model expression* which models its probability distribution. The required parameters are taken from the α and β *metrics* as defined above.



As such, the simulated answer then simply becomes the result of drawing a value for the test message feedback from a *Beta distribution model expression* with the parameters α and β as discussed above as well as the type *sample*. As we require information from both, we apply this *metric* to a *relation* between a *TestMessage* and a *Peer*.

6.3.5.2 Evaluating Model Robustness

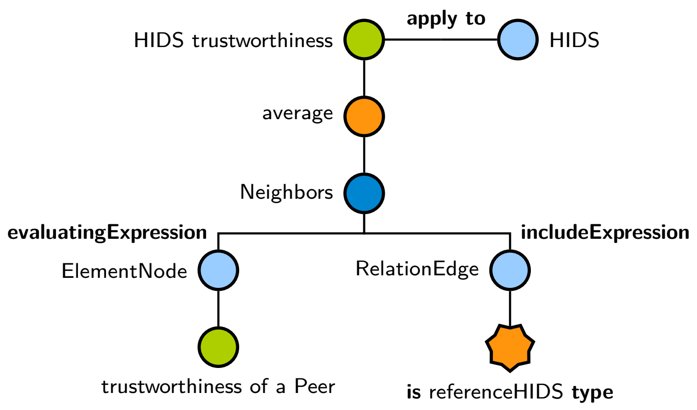
In order to fully evaluate the impact of various types of attacks on the intrusion detection approach proposed by Fung et al. [54] we need to define the appropriate *metrics*. In the following we will compare a basic average trustworthiness *metric* discussed by Fung et al. [54] and a new robustness index *metric* proposed here.

In the intrusion detection system proposed by Fung et al. [54] trustworthiness assessments are based on the modeled Dirichlet distribution of satisfaction levels. As discussed above, each *HIDS* maintains a *probation list* and an *acquaintance list* where *Peers* represent the assessment of neighboring *HIDS*. In order to assess the overall trustworthiness of a particular *HIDS* we therefore need to combine all *Peers* with the same *reference HIDS* accordingly. A straightforward approach is to calculate the average of these *Peer* trustworthiness assessments.

$$\overline{T}(HIDS) = \frac{1}{|peers_{\rightarrow HIDS}|} \sum_{peers_{\rightarrow HIDS}} T(peer) \tag{6.3.12}$$

In our *abstract graph model* we can refer to the specific *relations* shown in figure 6.3.2 and 6.3.3. Specifically, every *Peer* has a relationship with the *HIDS* it is assessing and this can be used to aggregate the trustworthiness assessments.

Expression 6.3.17 average HIDS trustworthiness metric
The average trustworthiness of a particular *HIDS* can be computed by first determining all *Peer element nodes* that have a *relation edge* of type *reference HIDS* with the *HIDS*. This can be done using the *Neighbor model expression*. For each of these *Peers* we can then apply the *trustworthiness of a Peer metric* and compute the respective average.

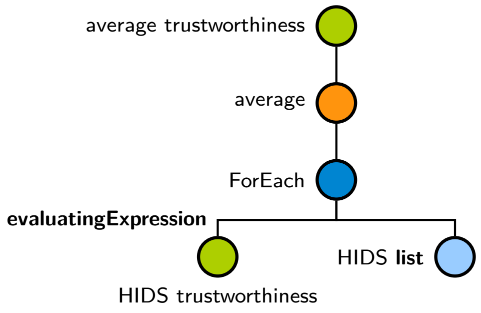


The overall average trustworthiness can be defined as

$$\overline{T} = \frac{1}{|HIDS|} \sum_{HIDS} T(HIDS) \tag{6.3.13}$$

Note that this average can be computed over any number of *HIDS element nodes*. For instance, in the discussion of our simulation results we will apply this *metric* to *HIDS element nodes* with varying *expertise levels*.

Expression 6.3.18 average trustworthiness metric
The average trustworthiness of a list of *HIDS element nodes* can be computed by using the *for each model expression* to determine the trustworthiness of each *HIDS* and then averaging the results.



Our approach to assessing the impact of various attacks on the intrusion detection system by Fung et al. [54] goes beyond looking at the average trustworthiness. The basic idea is that the overall system should provide a measure to clearly separate *HIDS* ele-

Table 6.3.1: Robustness Score Card

	Observed value		
True value	good	normal	malicious
good	1	0.5	-1
malicious	-1	-0.5	1

ments that are good and ones that are malicious based on trustworthiness assessments. Therefore, we propose a robustness index that takes into consideration the average trustworthiness of the system and determines which *HIDS* elements have a trustworthiness assessment significantly higher or lower than the average. During testing we can then check validate these assessments against the known true values.

In order to compute the robustness index we need to consider the possible cases shown in table 6.3.1 for each of the *HIDS* elements.

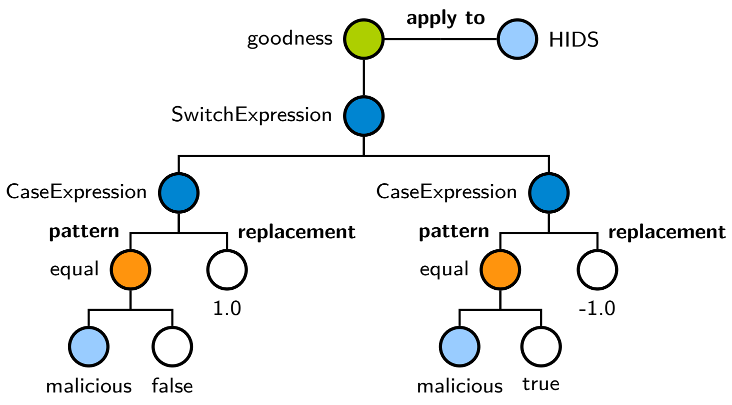
The true value refers to whether a *HIDS* is malicious or not. The observed value represents the assessment of the system where good means significantly higher trustworthiness (i.e., higher than average trustworthiness + threshold), normal reflects uncertainty (i.e., within the average trustworthiness ± threshold), and malicious significantly lower trustworthiness (i.e., lower than average trustworthiness - threshold). We can formally define this robustness index using the following approach.

Note that the robustness assessments for the good *HIDS* are just opposite of the *HIDS*. As such we define a binary function modeling the goodness of a *HIDS*.

$$G(HIDS) = \begin{cases} \text{1 if HIDS is good} \\ \text{-1 if HIDS is malicious} \end{cases} \tag{6.3.14}$$

Expression 6.3.19 goodness metric

The *goodness metric* is defined by two distinct cases and as such we can use the *switch model expression* and according *case model expressions* to determine the value of the *metric*.



We can then define the robustness of a particular *HIDS* according to the cases shown in table 6.3.1 as

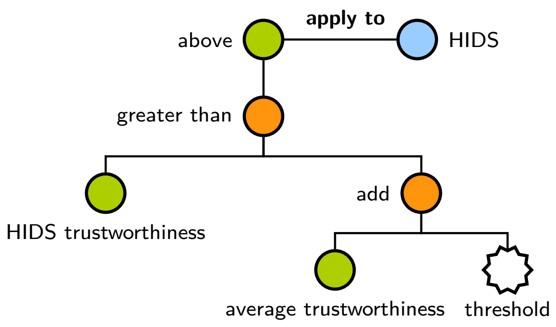
$$\rho(HIDS) = G(HIDS) \times \begin{cases} 1 & \text{if } \overline{T}(HIDS) > \overline{T} + \Delta T \\ 0.5 & \text{if } \overline{T} - \Delta T \leq \overline{T}(HIDS) \leq \overline{T} + \Delta T \\ -1 & \text{if } \overline{T}(HIDS) < \overline{T} - \Delta T \end{cases} \tag{6.3.15}$$

where $\overline{T}(HIDS)$ is the *HIDS* trustworthiness to be assessed, $\overline{T}$ the average trustworthiness, and ΔT the threshold surrounding the average.

For clarity reasons we will break down the modeling of this robustness *metric*. The first case deals with trustworthiness assessments that are significantly above normal.

Expression 6.3.20 above average trustworthiness metric

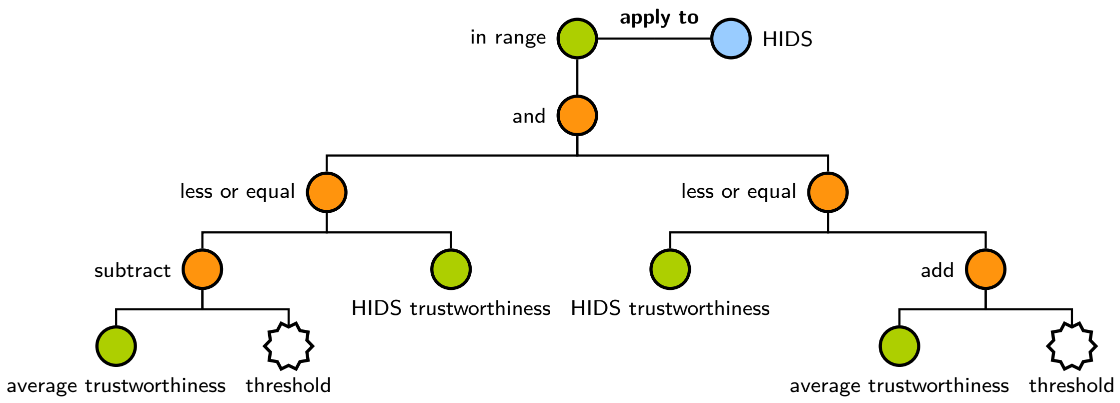
The *above average trustworthiness metric* performs basic math, logic and *attribute reference* calculations to determine whether the *HIDS* has an above normal trustworthiness.



Second, we need to determine whether the trustworthiness assessment falls within the range where the system is uncertain about the status of a *HIDS*.

Expression 6.3.21 within range of average trustworthiness metric

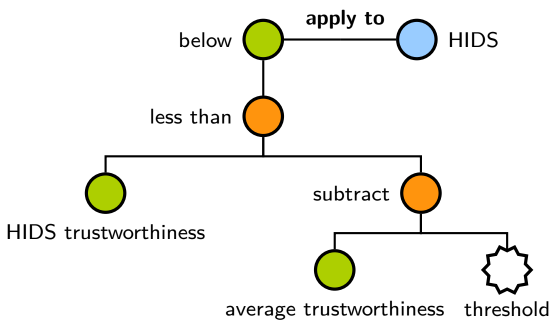
The *within range of average trustworthiness metric* performs a more complex comparison that involves checking two bounds (i.e., average trustworthiness ± threshold).



Third, trustworthiness assessments that are significantly below the average need to be considered.

Expression 6.3.22 below average trustworthiness metric

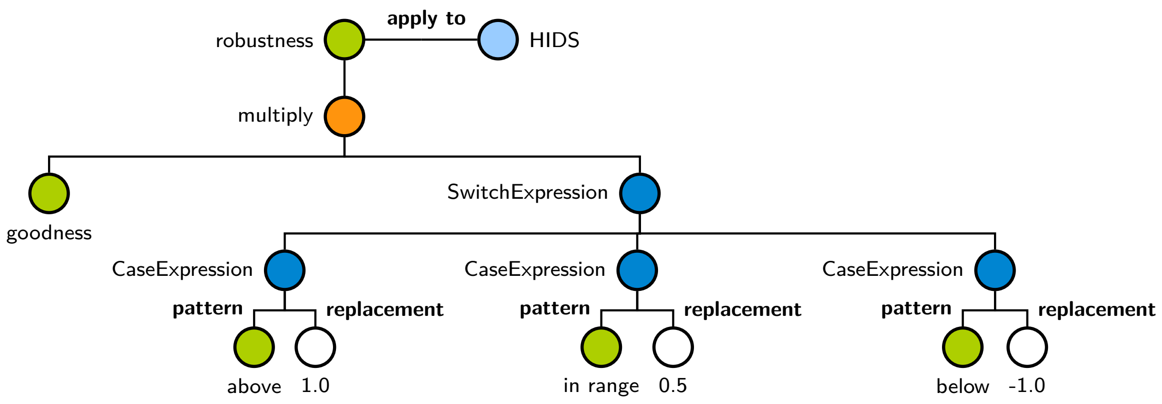
The *below average trustworthiness metric* utilizes a straightforward math and logical *expression*.



The individual robustness of a particular *HIDS* can then be computed using a combination of the *metrics* above.

Expression 6.3.23 robustness of a *HIDS*

The *robustness of a HIDS metric* implements equation 6.3.15 by multiplying the result of the *goodness metric* with the results for the appropriate case of where the particular *HIDS* trustworthiness assessment falls.

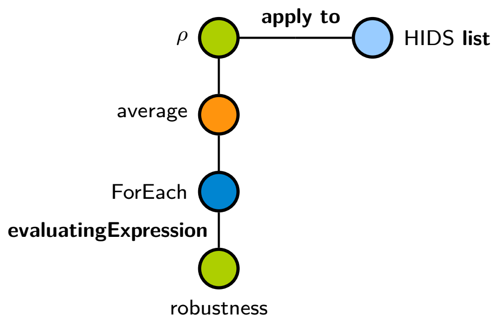


We define the overall robustness index for the intrusion detection system as the average of the individual robustness assessments of the *HIDS* elements.

$$\rho = \frac{1}{|HIDS|} \sum_{HIDS} \rho(HIDS) \tag{6.3.16}$$

Expression 6.3.24 robustness index ρ metric

The *robustness index ρ metric* is computed by averaging the robustness assessments of the given list of *HIDS* elements.

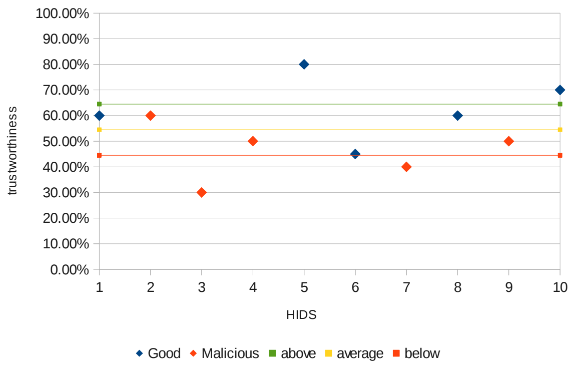


We show some examples of possible robustness indices in figure 6.3.4. Note that here we chose a threshold $\Delta T = 10\%$. Figure 6.3.4a shows a likely case where some good and some malicious *HIDS* are correctly identified while the majority does not deviate significantly from the average trustworthiness. The other three cases represent special occurrences that show the extreme values of the robustness index. In figure 6.3.4b there is not separation between good and malicious *HIDS* elements because all trustworthiness assessments are close to the average. As such the robustness index will approach 0. Figure 6.3.4c show the best case scenario where the trustworthiness assessments of all good and malicious *HIDS* elements fall into the right range. The worst case scenario happens when all *HIDS* elements are misclassified figure 6.3.4d.

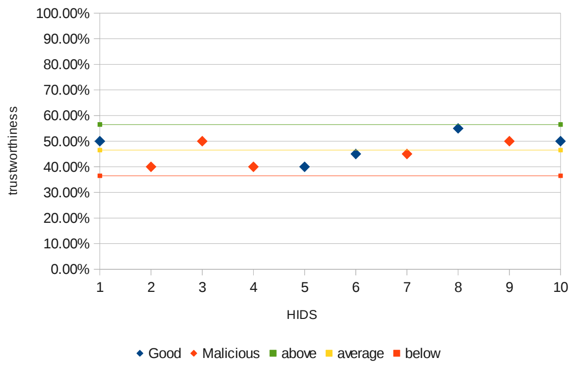
6.3.5.3 Evaluating Attacks

In the following we will discuss how robust the intrusion detection system by Fung et al. [54] is when facing a variety of attacks. For this purpose we developed a model that allows us to dynamically evaluate *graph expressions* even when the *abstract graph model* is changing. We will analyze and evaluate the following four different attack scenarios:

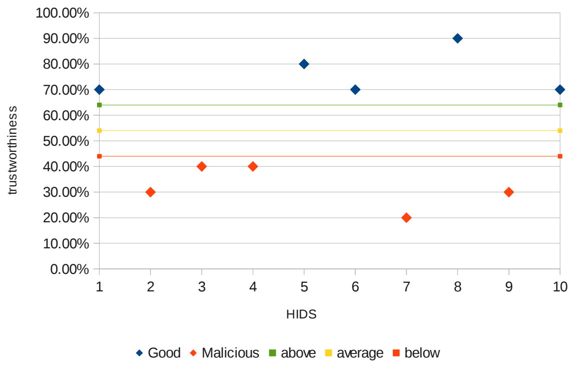
- newcomer attack with constantly joining nodes
- newcomer attack with a sudden flood of nodes



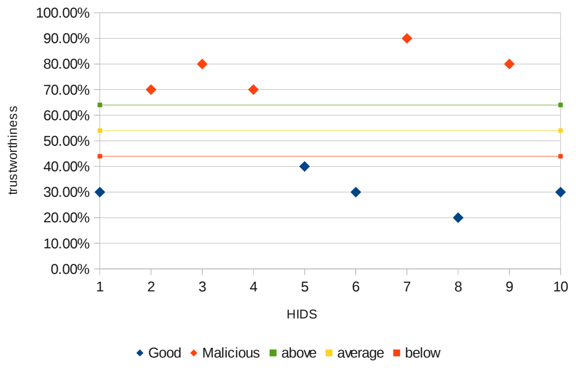
(a) Normal robustness ($\rho = 0.4$) with some good and some malicious *HIDS* correctly identified



(b) No robustness ($\rho = 0.0$) because of missing separation between good and malicious *HIDS*



(c) Perfect robustness ($\rho = 1.0$) with all good and malicious *HIDS* correctly identified



(d) Worst robustness ($\rho = -1.0$) with all good and malicious *HIDS* misclassified

Figure 6.3.4: Comparison of example cases for different types of robustness for 10 HIDS (5 good, 5 malicious) and $\Delta T = 10\%$

- betrayal attack with constantly betraying nodes
- betrayal attack with a sudden flood of betrayals

Since a full system trade-off analysis is out of the scope of this research we made a series of assumptions for our evaluation. Note that the focus of this scenario analysis is to show the flexibility of our *TrustKnowOne* framework, provide approaches to formally measure the impact of attacks on the knowledge derivation process, and deal with dynamically changing environments.

Table 6.3.2: Simulation Parameters

Parameter	Value	Description
R	2/day	test message rate
λ	0.9	forgetting factor
λ unit	day	forgetting factor unit
c_0	10	Priori constant
c_1	1.5	cost rate
c_2	1	satisfaction sensitivity factor
k	10	number of satisfaction levels
S_p	10	probation list size

Difficulty of test messages We select a *difficulty level* between low (0.1), medium (0.5), and high (0.9) with equal probability

Expertise level of HIDS The selection of the *expertise level* will be between low (0.05), medium (0.5), and high (0.95) with equal probability

Alert risk/ expected answer The expected answer is fixed to 0.5 for all test messages

Message rate Instead of dynamically increasing or decreasing the rate at which test messages are being sent, we fix it to 2 per day

Deception strategies A malicious node always evaluates the expected answer of a test message to 0.0 in order to create maximum harm.

Peer management For our evaluation there exists only a single list to maintain *Peers* and after it is filled with random *Peers* it does not change

Given these adjustments we can evaluate the impact of attacks on knowledge derivation using the system parameters shown in table 6.3.2 as defined by Fung et al. [54].

However, we need to discuss what it means for a *HIDS* to be malicious. There are three ways a malicious *HIDS* can behave:

Basic The malicious *HIDS* downgrades the alert risk of a test message to 0.0

Enhanced In addition to basic, a malicious *HIDS* reduces the satisfaction levels for test messages responses sent to other *HIDS* thus decreasing the trustworthiness of *HIDS* elements it is supposed to evaluate

Collaborative In addition to enhanced, the group of malicious *HIDS* increases the satisfaction levels for test messages from malicious *HIDS* elements thus increasing their trustworthiness

Note that in our results we compare these attacks with a baseline that acts as if the malicious *HIDS* element was good. Throughout our evaluation we will discuss the results for all three malicious behaviors. In general, each of the attack scenarios is modeled over 75 days. An overview of the number of malicious nodes in the system is shown in figure 6.3.5.

Newcomer Attack A newcomer attack introduces malicious *HIDS* elements into the system that were not there previously. The ideas is to overwhelm the existing *HIDS*. Here the system consisted of 30 *HIDS* elements with evenly distributed expertise levels exchange test messages for 25 days. Then we add 1 malicious newcomer *HIDS* join the system every day for 30 days, followed by 20 days of no change for stabilization.

As shown in figure 6.3.6a the intrusion detection system by Fung et al. [54] experiences a clear decline in average trustworthiness. However, if malicious nodes are collaborating the system actually behaves incorrectly because it shows a continuous rise in trustworthiness even as more malicious nodes are introduced.

The same pattern is reflected using the robustness index. Figure 6.3.6b clearly shows that the robustness index gives a better indication of how the intrusion detection system

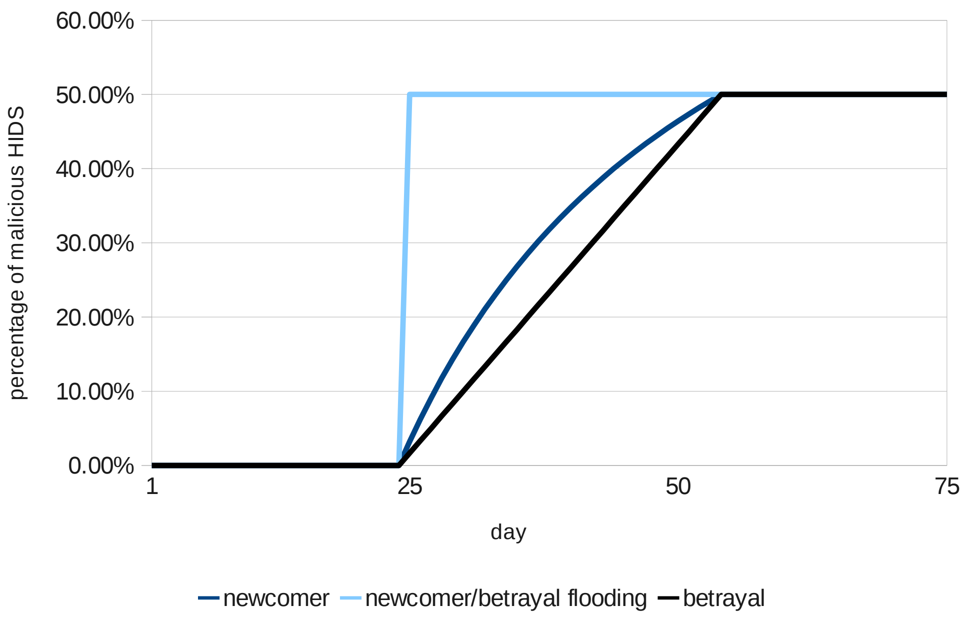
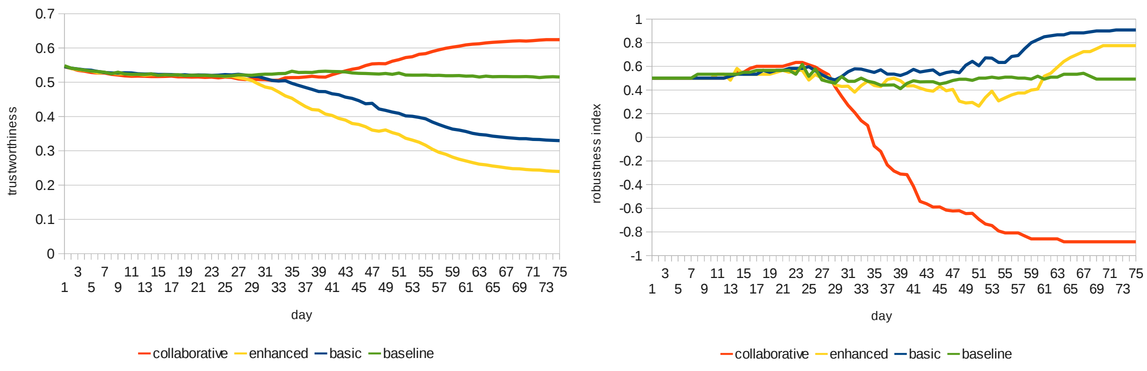


Figure 6.3.5: Intrusion evaluation malicious nodes per scenario overview

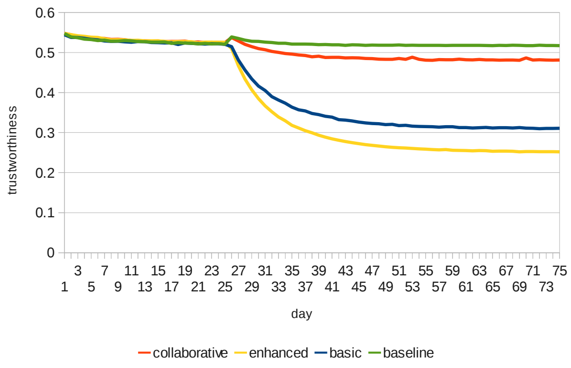


(a) Average trustworthiness

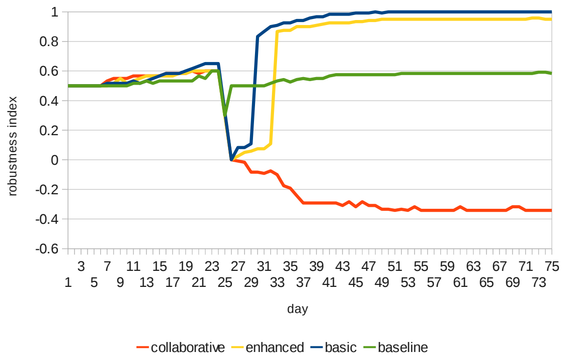
(b) Robustness index

Figure 6.3.6: Comparison of average trustworthiness assessments of HIDS nodes and robustness index for the newcomer attack scenario

is affected. We see that there are some robustness measures that are able to counter the basic and enhanced behaviors of malicious *HIDS*. However, when they are collaborating the system is vulnerable.



(a) Average trustworthiness



(b) Robustness index

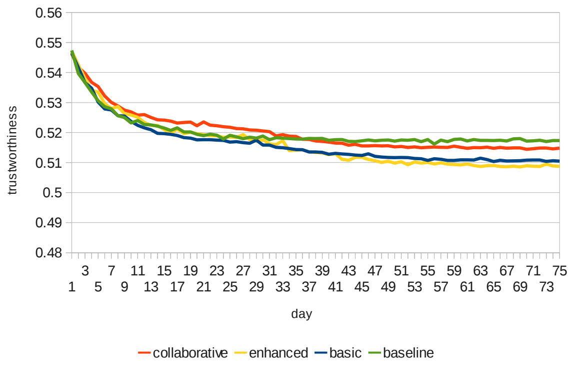
Figure 6.3.7: Comparison of average trustworthiness assessments of HIDS nodes and robustness index for the newcomer flooding attack scenario

Newcomer Flooding Attack A variation of the newcomer attack is when instead of continuously joining the system, an attacker creates a flood of new malicious *HIDS* elements. This system consisted of starting off with 30 *HIDS* elements with evenly distributed expertise levels sending test messages. After 25 days 30 new malicious *HIDS* join the system which then has 50 days of no change to stabilize.

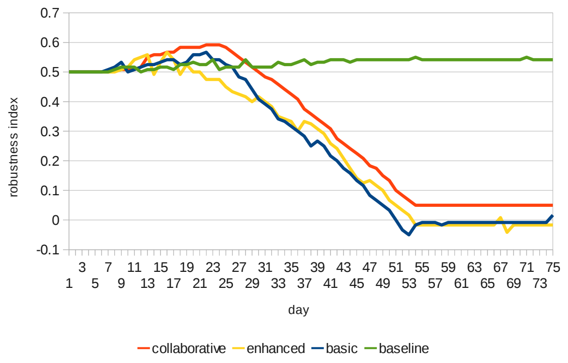
Figure 6.3.7a shows the sudden decrease in trustworthiness across all expertise levels. We can also recognize that for the basic and enhanced malicious behaviors the system behaves as expected and decreases average trustworthiness. As such, it is another indication that the system does not do well with malicious *HIDS* elements that collaborate.

The evolution of the robustness index shown in figure 6.3.7b clearly identifies the flooding attack and the subsequent recovery. Note that when compared to the average trustworthiness figure the robustness index seems to be better suited for detecting certain attack scenarios. We can also notice the impact on the robustness index of malicious *HIDS* elements collaborating.

Betrayal Attack A betrayal attack works by malicious *HIDS* first pretending to be good to the intrusion detection system. After establishing some trust with neighboring



(a) Average trustworthiness



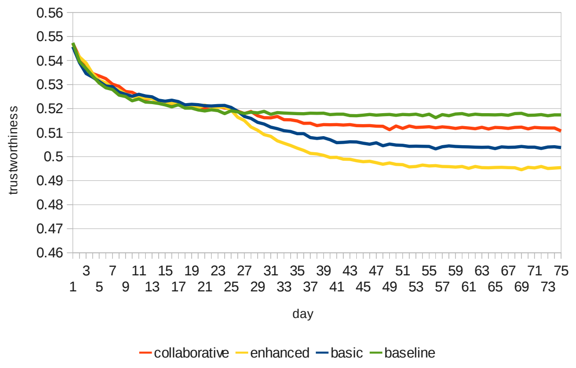
(b) Robustness index

Figure 6.3.8: Comparison of average trustworthiness assessments of HIDS nodes and robustness index for the betrayal attack scenario

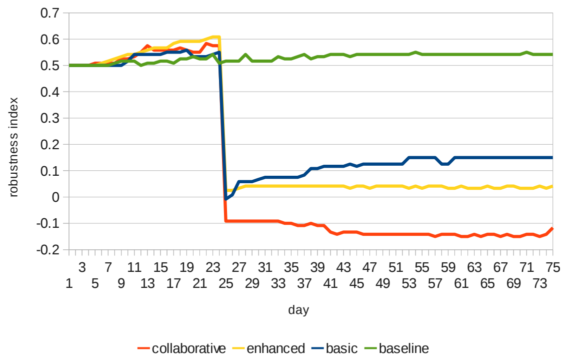
HIDS elements we start turning malicious. The system starts with 60 *HIDS* elements and evenly distributed expertise levels. After 25 days 1 *HIDS* turns malicious every day for 30 days. The remaining 20 days are then used for stabilization.

As shown in figure 6.3.8a the average trustworthiness of the *HIDS* elements in the system starts to decline immediately. Interestingly there is hardly a distinction between the malicious behaviors and their impact on the average trustworthiness. However, when evaluating figure 6.3.8b there is a clear impact on the robustness index once *HIDS* elements start turning malicious. Note that robustness decreases for all types of malicious behavior which is different from the system facing a newcomer attack that was fairly robust against basic and enhanced. The robustness terms also approach 0 which would indicate that the system is unable to clearly separate trustworthiness assessments (see figure 6.3.4b.

Betrayal Flooding Attack A betrayal flooding attack works by turning multiple *HIDS* elements into malicious ones at the same time. In this system we are starting off with 60 *HIDS* whose expertise level was evenly distributed until 25 days have passed. Then we betrayed 30 *HIDS* elements at once and gave the simulation another 50 days to stabilize.



(a) Average trustworthiness



(b) Robustness index

Figure 6.3.9: Comparison of average trustworthiness assessments of HIDS nodes and robustness index for the betrayal flooding attack scenario

The resulting average trustworthiness assessments are similar to the continuous betrayal attack scenario (figure 6.3.9a). However, there is a small difference as the average diverge slightly after the betrayal flooding. The robustness index shows a drastic drop off as soon as the flooding started and recovers slightly afterwards (figure 6.3.9b). Nevertheless, the low robustness values indicate that the intrusion detection system is not able to deal well with betrayal attacks. Note that the robustness index provides a better assessment of the impact an attack has on the system than the average trustworthiness approach.

6.3.6 Intrusion Detection: Summary

Intrusion detection is an important field in the area of computer system security. While there are many proposed approaches they can usually only be broadly classified into categories (e.g., network-based, host-based, centralized, distributed, etc.). More detailed analysis and comparison is often difficult due to the lack of a formal way of describing intrusion detection systems and their performance metrics.

In this section we showed that our framework is capable of modeling a collaborative trust management approach by Fung et al. [54]. We discussed in detail how individual components such as a HIDS, peers and test messages as well as their relationships can be represented by our *abstract graph model.* As such, we showcase how our *framework* is capable of modeling dynamic environments.

Furthermore, we examined the specific trust assessment methodology of the Fung approach and demonstrated in this section that they can be formalized using various graph *metrics* and *expressions.* This includes the extensive modeling of deriving trustworthiness and confidence assessments by determining satisfaction levels of test messages. As such, we showed how our *framework* is able to express the intrusion detection system approach by Fung et al. [54] as *belief engines* and *decision processes.*

Furthermore, we discussed two approaches to evaluating the system against various types of attacks. In particular, we showed that the basic average trustworthiness metric by Fung et al. [54] is insufficient. Therefore, we developed and evaluated a new robustness index which is based on how clearly the intrusion detection system is able to separate good nodes from malicious ones. Here, we provided a detailed analysis of the both metrics with regards to newcomer and betrayal attack scenarios. Even though our analysis is not a complete system design and trade-off analysis we demonstrated that our *TrustKnowOne* framework is flexible and expressive enough to model a complex intrusion detection system.

Additional benefits for this scenario of modeling algorithms and approaches using our generic framework based on the *abstract graph model* include:

Comparison The formalized nature of the components (i.e., *HIDS*, *Peer*, *TestMessage*) and algorithms (i.e., evaluating feedback, peer trustworthiness, peer confidence, feedback simulation) allows for the comparison to other intrusion detection approaches modeled in the same manner. For instance, we introduced a new robustness metric and successfully compared it to the one suggested by Fung et al. [54]

Extension New components as well as new algorithms can be introduced simply on the basis of adding *elements* and *relations* to the abstract graph model. The implementation of new algorithms can be performed using graph *expressions* (e.g., a different trustworthiness approach for *Peers*, a simpler test message feedback formula).

Reusability We showcased how individual graph *metrics* and *expressions* can be combined into more complex ones (e.g., gamma vector, weights, peer expertise level). Common *element* and *relation* descriptions of the graph model could be used across various trust management approaches such that they can be compared easily.

Performance Since algorithms are described as a combination of graph *expressions* we are able to pinpoint cost measures to evaluate them. This allows us to assess aspects of the particular algorithm as well as potential changes to them in detail.

6.4 Chapter Summary

In this chapter we showed how our framework can be applied to a variety of scenarios. Specifically, we discussed three diverse scenarios and how they can be expressed using the methodology the *TrustKnowOne* framework. A detailed comparison and evaluation of these scenarios with regards to other frameworks and approaches will be performed as part of chapter 8.

7 Implementation

We developed a reference implementation of the *TrustKnowOne* framework in Java. While the choice of Java already provides a certain platform independence for the framework, note that the approaches and processes described in the previous chapters (chapter 4, 5) and applied to several scenarios (chapter 6) can be implemented in other languages and on a variety of platforms. As such, our *TrustKnowOne* framework as discussed throughout this dissertation provides a description of interfaces as well as processes necessary for implementing knowledge derivation processes that incorporate trust and quality of data.

As part of our implementation the formal aspects of our approach can be specified according to respective XML schema definitions which have been developed with a focus on interoperability. Hence, all graph components (*elements*, *relations*, *metrics*, etc.), data sources (files, databases, etc.), and mappings (*data extraction adapters*) can be described formally using a set of XML tags. Furthermore, we provide a Java application programming interface (API) that allows future extensions to use visual tools, domain specific languages, and a wide range of graph formats for the description of the *abstract graph model* as well as *graph expressions*.

The incorporation of dynamic aspects is central to our framework. Therefore, in our implementation we timestamp all data instances and can maintain extensive provenance information. This allows us to deal with changes in meaning (values, data types, etc.) and structure/topology (relations, connections, etc.) throughout the graph model. We also differentiate data and data sources using their determined trustworthiness and incorporate these trustworthiness assessments into decision processes thus improving them.

Flexibility, one of the key aspects of our approach is reflected in the implementation of the *TrustKnowOne* framework. We support various *abstract graph model* storage approaches (initially in-memory and graph database) that can be incorporated by implementing an interface accordingly. Additionally, the computational model used for applying *graph expressions* to *graph components* can be exchanged as well to support additional approaches than the ones currently implemented (sequential, thread-based parallelization). Note that both storage and computational model are independent from other parts of the framework as they only need to adhere to formal interface definitions. Thus, the framework can be adapted to specific application scenarios simply by exchanging one implementation for another.

In this chapter we showcase implementation aspects[1] by discussing parts of the Trusting Smartphone Apps scenario (section 6.1) where we applied our framework to the domain of trustworthiness assessment of Android Apps. In particular, we investigated why basic App attributes such as average rating and a number of positive reviews are not necessarily good indicators of an App's trustworthiness. Hence, we considered three different data sources for the *data extraction* component: rating information, reviews, and permissions. As part of the *data processing* component, we then developed two *metrics* for each type of information (ratings, reviews, permissions) that incorporate a variety of meta and relationship information. Furthermore, in order to derive

[1]The examples used in this chapter are based on the Trusting Smartphone Apps application scenario described in section 6.1. They are simplified for illustration purposes and do not reflect the full complexity of the scenario.

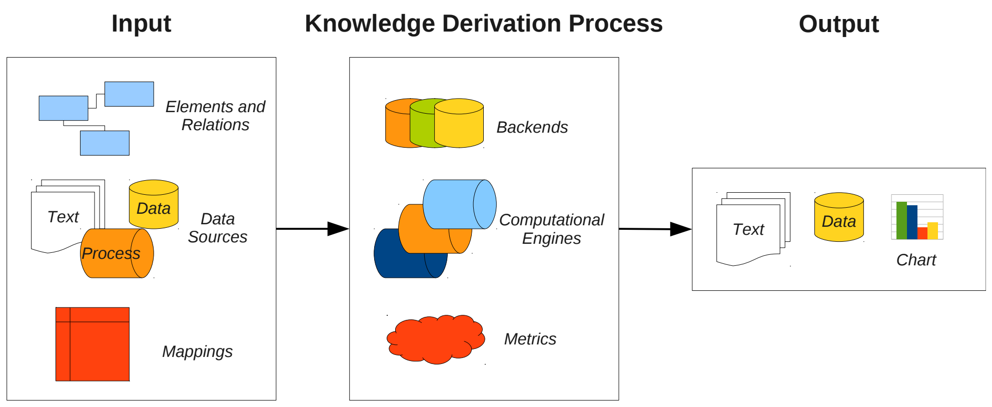


Figure 7.1: Implementation components overview

a trustworthiness assessment for individual Apps we implemented a linear weighting scheme that functions as the decision engine in our *evaluation* component.

An overview of the components required for an implementation of our *TrustKnowOne* is shown in figure 7.1. In the following sections we will discuss the components and provide details of our reference implementation.

7.1 Input

Formally specifying the input to our knowledge derivation framework follows the methodology described in section 5.2.1. As such our implementation provides techniques to describe the *elements* and *relations* of a particular scenario, the respective data sources, and the mappings from data sources to *graph components* in the *abstract graph model*.

7.1.1 Elements And Relations

We provide a number of options to describe the *graph components* used to model a particular application scenario. First, a flexible, interoperable, and extensible technique involves the use of Extensible Markup Language (XML). As shown in listing 7.1, we can

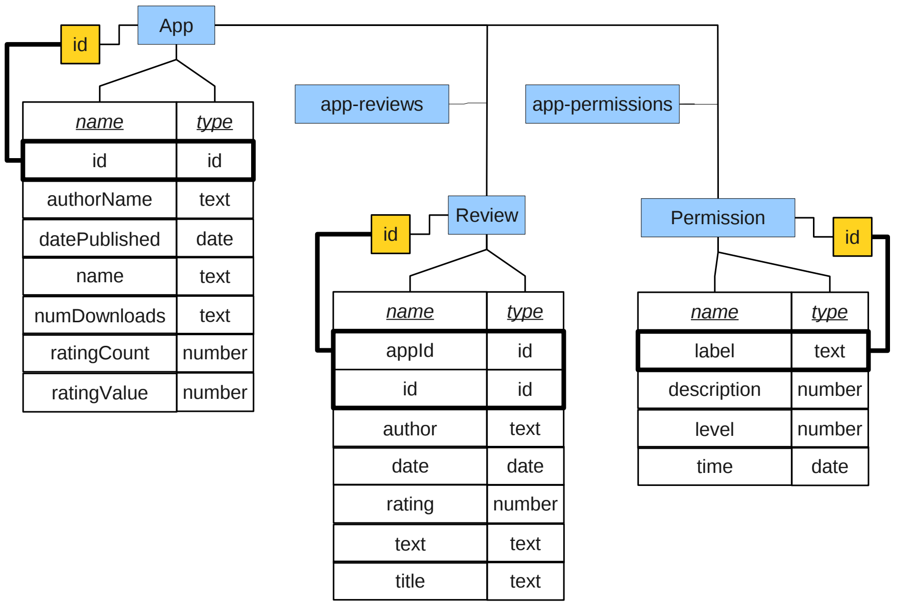


Figure 7.2: Simplified *element description graph*

define basic *graph components* such as *elements* and *relations* using tags and attribute values. In particular, we show three *elements* where the unique identifier and timestamp properties are derived from a particular attribute of an *element*. Furthermore, we specify an appropriate type for each of the attributes.

Figure 7.2 shows the same simplified version of the *abstract graph model* in graphical form. This type of graphical modeling can be used to organize more complex application scenarios.

In order to provide the most flexibility we also provide an application programming interface (API) to our framework implementation for describing *elements* and *relations*. As shown in listing 7.2 this approach achieves the same objective of providing a flexible and extensible way to model *graph components* of the trusting Smartphone Apps scenario.

```xml
<element name="App">
  <id>
    <attribute ref="id"/>
  </id>
  <time name="datePublished" pattern="yyyy-MM-dd"/>
  <attribute name="authorName" type="xs:string" />
  <attribute name="id" type="xs:ID" />
  <attribute name="name" type="xs:string" />
  <attribute name="numDownloads" type="xs:string" />
  <attribute name="ratingCount" type="xs:positiveInteger" />
  <attribute name="ratingValue" type="xs:double" />
</element>

<element name="Permission">
  <id>
    <attribute ref="label"/>
  </id>
  <time name="time" pattern="yyyy-MM-dd" />
  <attribute name="description" type="xs:string" />
  <attribute name="label" type="xs:string" />
  <attribute name="level" type="xs:string" />
</element>

<element name="Review">
  <id>
    <attribute ref="appId"/>
    <attribute ref="id"/>
  </id>
  <time name="date" pattern="yyyy-MM-dd" />
  <attribute name="appId" type="xs:ID" />
  <attribute name="id" type="xs:ID" />
  <attribute name="author" type="xs:string" />
  <attribute name="rating" type="xs:positiveInteger" />
  <attribute name="text" type="xs:string" />
  <attribute name="title" type="xs:string" />
</element>

<relation name="app-reviews" sourceRef="App" targetRef="Review"
  type="one-to-many"/>
<relation name="app-permissions" sourceRef="App" targetRef="Permission"
  type="one-to-many"/>
```

Listing 7.1: Simplified XML *Element* and *Relation* description

```java
Element appElement = new Element("App", new IdAttribute("id"));
appElement.setTimeAttribute(new TimeAttribute("datePublished");
appElement.add(new TypedAttribute<>("id", String.class));
appElement.add(new TypedAttribute<>("authorName", String.class));
appElement.add(new TypedAttribute<>("ratingValue", Double.class));
appElement.add(new TypedAttribute<>("ratingCount", Integer.class));

Relation relation = new Relation("app-reviews", "App", "Review");
```

Listing 7.2: Simplified Java *Element* and *Relation* description

7.1.2 Data Sources

Formally describing data sources is difficult due the complexity of non-standard interfaces. Our reference implementation provides an approach that is flexible and extensible since we only need to specify how to connect to a data source and retrieve raw data. The actual integration of data into the *abstract graph model* is performed as part of the mapping process. This allows us to incorporate a variety of data sources such as databases and files into the knowledge derivation process.

In addition, by using the API we are able to efficiently deal with the continuous integration of new data. In case of the trusting Smartphone Apps scenario, we developed a web crawler that provided a snapshot of the data we wanted to model. While this allowed us to perform a detailed analysis of trust aspects (see section 6.1), the data remained static. However, we could have easily extended our scenario to have a dynamic web crawler that continuously fed new data into the framework. As discussed in section 5.2.2, dynamic retrieval of additional data can improve decision processes and is a necessary step in enable knowledge derivation in real-time and streaming applications. Hence, we can also model a data source as a process that either continuously provides data or provide data when requested.

An example of dynamically creating an additional App *element node* for the application scenario is shown in listing 7.3.

```java
ElementNode one = new ElementNode(appElement);
ElementInstance oneInstance = one.createInstance();
oneInstance.set("id", "com.google.android.maps");
oneInstance.set("authorName", "Google");
oneInstance.set("ratingValue", "4.6");
oneInstance.set("ratingCount", "9020");
one.add(oneInstance);
```

Listing 7.3: Dynamically creating a new *element node* using the API

7.1.3 Mappings

Given the description of the *graph components* and the data sources for an application scenario, we are able to define a mapping process that will transform raw data from a data source into a respective *graph component*. Our reference implementation provides an approach based on XML in order to perform the mapping. As discussed in section 5.2.1, we enable incorporating custom adapters for handling application specific data transformations efficiently.

In listing 7.4, we map data from a comma separated value file onto the *abstract graph model* by describing the row format of the file. Note that we split the information found in a row into an *App* and a *Category element node*. In addition, our implementation automatically adds a relationship accordingly.

An efficient way to store information about which entities in a graph are related is an adjacency list where a list of neighboring nodes is kept for each node of the graph. As such, our reference implementation allows the mapping of data onto *element nodes* and relationships onto *relation edges* in a flexible manner. For example, listing 7.5 shows a mapping in which the number of *Permission element nodes* per *App* varies.

7.2 Knowledge Derivation Process

In order to provide an framework implementation that allows knowledge derivation as described in this dissertation we need to discuss the following. The previous section

```xml
<data name="apps">
  <csv file="apps.csv" header="false" separator=","
    quoteCharacter="double" >
    <rowFormat>
      <element ref="App" number="1">
        <attribute ref="id" />
        <attribute ref="name" />
      </element>
      <element ref="Category">
        <attribute ref="name" />
      </element>
      <element ref="App" number="1">
        <attribute ref="authorName" />
        <attribute ref="datePublished" />
        <attribute ref="numDownloads" />
        <attribute ref="softwareVersion" />
        <attribute ref="ratingCount" />
        <attribute ref="ratingValue" />
      </element>
    </rowFormat>
  </csv>
  <relation ref="app-category"/>
</data>
```

Listing 7.4: Simplified XML mapping description

```xml
<data name="appPermissions">
  <csv file="appPermissions.csv" header="false">
    <rowFormat>
      <element ref="App">
        <attribute ref="id"/>
      </element>
      <group max="unbounded">
        <element ref="Permission">
          <attribute ref="label"/>
        </element>
      </group>
    </rowFormat>
  </csv>
  <relation ref="app-permissions"/>
</data>
```

Listing 7.5: Variable number of relationships mapping

discussed in detail how our reference implementation describes *graph components*, data sources, and mappings. However, an efficient implementation our framework requires an efficient *abstract graph model* backend responsible for storage and retrieval of *graph components*. Furthermore, how do we describe *graph expressions* in a flexible and extensible manner and more importantly how do we evaluate them on the *abstract graph model*. Our reference implementation provides a solution that is flexible yet scalable in size and performance.

7.2.1 Abstract Graph Model Backends

There exist a variety of data storage approaches. However, we focus on a particular type that is our *abstract graph model*. In order to remain flexible and extensible our implementation performs storage and retrieval operations on an abstract graph model interface. This allows the underlying storage model to be adapted to specific needs. For instance, if the amount of data is highly dynamic (e.g. streaming data) then storing it in memory improves performance since data does not have to be written and read from disk. On the other hand, relational or graph databases offer advantages in the retrieval of data because of complex index systems. Furthermore, document mining usually follows a file based approach. As such, it is important to note that by not depending on a single type of backend but rather providing the ability to choose our reference implementation allows for the flexibility necessary to deal with a variety of applications scenarios. Note that this enables extension to distributed (e.g., Hadoop [169, 178]) as well as hybrid (e.g., using a database for old data and in-memory for newer data) backend approaches.

7.2.2 Metrics

Our *TrustKnowOne* framework performs knowledge derivation by evaluating *graph expressions* on the *graph components* stored in the *abstract graph model*. Whereas *belief* *engines* assess aspects such trust and quality of data, *metrics* and *decision processes* are usually utilized to compute values from *graph components*. These *graph expressions* can be implemented using several approaches. Our reference implementation utilizes an *expression* interface for which every *graph expression* needs to provide specific implementations (listing 7.6). Note that this can easily be extended to include other graph components or combinations thereof.

```java
public interface Expression {
  public Object evaluate(ElementNode node);
  public Object evaluate(List<ElementNode> nodes);
  public Object evaluate(RelationEdge edge);
}
```

Listing 7.6: Java reference implementation expression interface

We then incorporate this interface when defining possible relationships between *graph expressions*. For instance, a binary comparison as shown in listing 7.7 requires two child *graph expressions* which are then evaluated on a particular graph component.

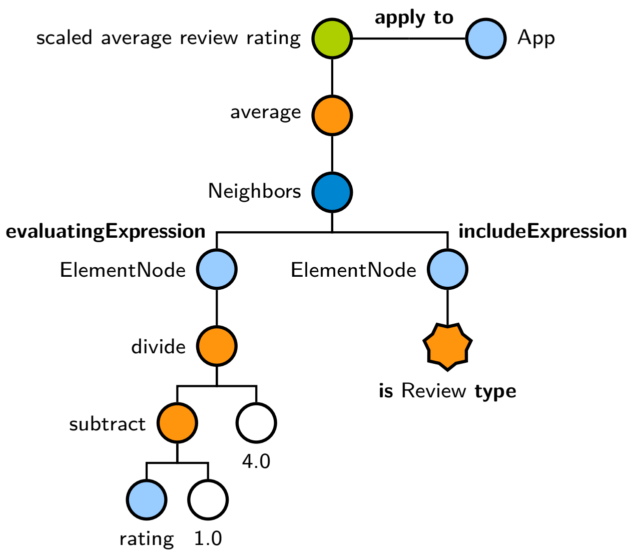
```java
public class Equal implements Expression {
  protected Expression a;
  protected Expression b;

  public Equal(Expression a, Expression b) {
    this.a = a;
    this.b = b;
  }
  public Object evaluate(ElementNode node) {
    return (a.evaluate(node)).equals(b.evaluate(node));
  }
  public Object evaluate(List<ElementNode> nodes) {
    return (a.evaluate(nodes)).equals(b.evaluate(nodes));
  }
  public Object evaluate(RelationEdge edge) {
    return (a.evaluate(edge)).equals(b.evaluate(edge));
  }
}
```

Listing 7.7: Java expression example implementation

The results can then be compared accordingly. This approach maintains great flexibility in terms combining *graph expressions* of all different types and make it straightforward to incorporate additional ones into the framework.

In chapter 4, we already presented an *expression tree* form to define *graph expressions* and in this section we provide two additional methods. As an example, we choose the following metric from the trusting Smartphone Apps scenario.

Expression 7.1 Scaled average review rating



Because of its flexibility and interoperability we also chose XML in our reference implementation to specify *metrics*. An advantage of XML is its inherent tree structure which makes the transformation into an *expression tree* straightforward and efficient (see listing 7.8).

As discussed earlier, our implementation provides an API that allows for *metrics* to be dynamically added (listing 7.9).

7.2.3 Computational Engines

Evaluating *graph expressions* on the *graph components* can be done in a variety of ways. Our reference implementation of the *TrustKnowOne* framework provides an

```xml
<metric name="scaled average review rating">
  <element ref="App">
    <average>
      <neighbors>
        <parameter name="includeExpression">
          <element ref="Review"/>
        </parameter>
        <parameter name="evaluatingExpression">
          <element ref="Review">
            <divide>
              <subtract>
                <attribute ref="rating"/>
                <value>1.0</value>
              </subtract>
              <value>4.0</value>
            </divide>
          </element>
        </parameter>
      </neighbors>
    </average>
  </element>
</metric>
```

Listing 7.8: XML scaled average review rating *metric* definition

```java
Neighbors reviewRatingNeighbors = new Neighbors();
reviewRatingNeighbors.set(NeighborsParameter.includeExpression,
  new ElementReference(new TypeConstraint("Review")));
reviewRatingNeighbors.set(NeighborsParameter.evaluatingExpression,
  new ElementReference(
    new Divide(
      new Subtract(
        new AttributeReference("rating"),
        new Constant<Double>(1.0)),
      new Constant<Double>(4.0))));

Average averageReviewRating = new Average(reviewRatingNeighbors);

Metric scaledAverageReviewRating =
  new Metric("scaled average review rating", averageReviewRating);
```

Listing 7.9: Java scaled average review rating *metric* definition

abstraction layer that makes it possible to utilize currently well-known as well as future computational approaches. In order to maintain scalability our implementation deals with *model keys* that consist of the identifier and type instead of the complete *graph component* whenever possible. This approach in essence follows the key-value mapping approaches found in large-scale data processing frameworks (e.g., [39, 42, 103, 109, 139]). This makes it easy to extend single machine to distributed computations such as Hadoop [169, 178].

Our reference implementation not only allows for sequential and parallel but also agent-based processing. In contrast to frameworks such as Hadoop, we are able to perform dynamic evaluation and reevaluation on parts of the *abstract graph model* without the need for a central control functionality. Essentially, the *abstract graph model* could be maintained independently of the entities that perform queries and computations on it. An advantage of the *abstract graph model* approach is that clustering becomes more efficient since relationships are already known. This also makes it easier to coordinate potential synchronization issues that arise on shared data during parallel processing.

7.3 Output

Providing the results of the knowledge derivation process in a variety of formats is important because it increases the usefulness of our framework. As such we provide several mechanisms to retrieve the results of evaluating *graph expressions.* In particular, one can utilize the Java API to implement *output adapters.* This allows the creation of static results such as tables, files, and charts as well as dynamic ones such as event notifications and propagation to other systems.

7.4 Application Programming Interface

While we chose Java for the reference implementation of our *TrustKnowOne* framework, the approaches and methodologies discussed throughout this dissertation can be incorporated into other existing or future frameworks. The framework itself is language agnostic, platform independent, and because of the clearly defined interfaces such as the ones between *knowledge extraction*, *processing*, and *evaluation*, our framework is flexible. Our implementation provides an *abstract graph model* that can be used for managing data in heterogeneous and dynamic environments. Furthermore, our *TrustKnowOne* framework enables the incorporation of trust and quality of data aspects into decision making process in a formalized manner.

Throughout our reference implementation we have focused on providing the most flexible and open approach possible. By providing an API in addition to the XML and graph based formalization of our reference implementation, interfaces could be provided to other languages such as C++, Python, etc.

7.5 Chapter Summary

In addition to the *TrustKnowOne* framework discussed throughout this dissertation, we provide a reference implementation. In this chapter, we described the components of our implementation including input, knowledge derivation process, and output.

As discussed in section 5.2.2, we need to describe the *graph components* used in the respective application scenario. Here we provided three equivalent ways to model *elements* and *relations* (XML, *expression tree*, and Java). Furthermore, data sources often lack formal descriptions because of their variety of formats. Here we proposed an approach where we capture basic information that can be used to connect to a data source and define mappings that transform raw data into equivalent *graph components*. As part of the mappings, relationships can be determined and added automatically.

In this chapter we also discussed the requirements regarding data management and processing. Specifically, our reference implementation provides an *abstract graph model* interface that allows various backends to be used that range from in-memory to distributed and hybrid approaches. A similar layer of abstraction is provided for computational engines such that they can be adjusted depending on the requirements of the application scenario (e.g., sequential, parallel, distributed, agent-based). *Metrics* have been discussed in detail in section 4.4 and 6. However, in this chapter we provided two additional ways to define them (XML, Java).

The output of the framework can be converted into a variety of formats using the application programming interface (API) provided by our implementation. While we have presented a number of techniques to describe parts of the *TrustKnowOne* implementation, our approaches as discussed in this chapter make the overall framework flexible and extensible.

8 Evaluation

We discussed the details of our *TrustKnowOne* framework in previous chapters. Furthermore, we showed how the framework can be applied in various scenarios. Here, we build on the related work chapter to evaluate *TrustKnowOne* within the context of trust assessment, data modeling, and large-scale processing. We confirm our claims concerning the contributions of our *TrustKnowOne* framework. In particular, we compare important aspects of our approach to representative frameworks found in literature. In order to provide the reader with an accurate assessment of our framework, the literature representatives were chosen based on the fact that they are best in class, highly utilized and widely referenced. As such, we evaluate and compare our *TrustKnowOne* framework with approaches focused on large-scale data processing (i.e., Hadoop [169, 178], Dryad [76]), graph algorithms (i.e., Pregel [107], Pegasus [86], GraphLab [104]), and distributed trust approaches (i.e., EigenTrust [85], TrustRank [67], PowerTrust [187]).

The main advantage of Hadoop [169, 178] is that it bases large-scale data processing on the abstract MapReduce paradigm [37–39] which allows for a formal yet flexible approach to "big data" analysis. While Dryad [76] shares the same goal, its approach is different. Instead of providing an abstract layer where large-scale data processing is performed automatically on a distributed set of computing nodes, Dryad represents a distributed execution engine. In particular, users have to specifically define computational nodes and their processing relationships. Nevertheless, both frameworks have weaknesses in areas such as formalization of data and processes. Furthermore, they do not incorporate trust and quality of data into the knowledge derivation process which we will discuss in detail as part of this chapter.

There exist a variety of graph frameworks that we can compare our *TrustKnowOne* framework against. Specifically, Pregel [107] represents a computational model based on nodes. Programs are defined as a series of iterations in which nodes send messages around to other nodes which then modify data stored in nodes, edges, and graph topology. In similar fashion, Pegasus [86] provides a collection of graph mining algorithms that can be implemented using node-based computations. However, instead of message passing, approaches are modeled as matrix-vector multiplications which limits their ability to be scaled and distributed. GraphLab [104] provides a parallel abstraction for machine learning algorithms with a data graph that models computational dependencies. It incorporates a variety of techniques to address issues such as data consistency and process scheduling. However, most graph frameworks often do not consider heterogeneous data and relationships as well as trust assessment aspects.

Approaches that focus on the assessment of trust relationships are often limited in terms of formalization and flexibility. For instance, EigenTrust [85] provides a technique for distributed trust assessment of nodes in peer-to-peer systems. It is based on local trust values (binary positive or negative assessments of transactions) and normalized local trust value that lead to transitive trust assessments (i.e., trust of peer's friends weighted by trust in peer). While this approach works well in a distributed way and is robust against a variety of attacks, it ignores other trust and quality of data aspects (e.g., non-binary, global relationships). Furthermore, it lacks formalization and flexibility. PowerTrust [187] extends the principles of EigenTrust but instead of using a seed set of known trust nodes, it relies on historical information to determine power nodes from

Table 8.1: Comparison of major aspects in knowledge derivation processes

Aspect	TrustKnowOne	Hadoop [169, 178]	Dryad [76]	Pregel [107]	Pegasus [86]	GraphLab [104]	EigenTrust [85]	TrustRank [67]	PowerTrust [187]
heterogeneous systems	●	◒	◒	◒	○	◒	○	○	○
dynamic systems	●	◒	○	◒	○	○	○	○	◒
formal representations	●	◒	◒	◒	◒	◒	◒	◒	◒
quality and trust assessments	●	○	○	○	○	○	◒	◒	◒
flexibility in approaches	●	●	●	●	◒	◒	◒	◒	◒

○ no support ◒ partial support ● full support

which to propagate trust. TrustRank [67] is similar to PageRank [20] in the way it incorporates trust dampening and splitting techniques but is still based on the same of trust derivation from seed nodes. The most notable aspects of these trust approaches are that they are data agnostic (i.e., only concerned about providing a trust assessment approach) but often lack flexibility which would allow their reuse in heterogeneous and dynamic application domains. In contrast, our *TrustKnowOne* framework provides a complete knowledge derivation approach with the ability to incorporate various data processing as well as trust approaches. With regards to trust assessment techniques, we are able to combine, modify, and evaluate them because of their formalization as *belief engines*.

An overview of the major aspects of knowledge derivation supported by our *TrustKnowOne* and confirmed through the implementation of the three scenarios is shown in table 8.1. Table 8.1 also shows how well these major aspects are supported by the representative frameworks. In the following, we will discuss these aspects and then evaluate the frameworks in the context of the research areas outlined in related work (chapter 3).

8.1 Major Aspects

When developing our *TrustKnowOne* framework we identified five major aspects as requirements for a complete knowledge derivation system: heterogeneous systems, dynamic systems, formal representations, quality and trust assessments, and flexibility in approaches. Knowledge derivation in heterogeneous systems is difficult because one needs to address issues of data integration and conflicting data. However, a framework should be flexible enough to incorporate various types of data as well as relationships in order to allow for known information and more importantly information that is not yet known to be incorporated into decision making. *TrustKnowOne* provides an *abstract graph model* approach that provides this capability in a uniform and formalized way (see chapter 4,6). In contrast, other approaches often focus on particular applications which results in them lacking support for true heterogeneous systems. For instance, trust approaches such as EigenTrust [85], TrustRank [67], and PowerTrust [187] assume a homogeneous system of related peer nodes.

Another aspect that is often disregarded because it increases complexity is dealing with changes in dynamic systems. In its most basic form this means keeping track of changes in data over time. However, relationships between data elements may change as well. Many knowledge derivation approaches only consider snapshots of data which often does not capture all the context. Our *abstract graph model* approach provides a direct methodology to store data in time series.

The formalization of data, data sources, relationships, trust approaches, and decision processes is important to enable analysis, evaluation and comparison of various approaches. Nevertheless, most existing frameworks do not address this aspect which makes it difficult to determine their strengths and weaknesses. In addition, it becomes problematic to improve or exchange existing techniques as well as apply them in different application domains. As discussed in chapter 4 and chapter 5 *TrustKnowOne* provides an approach to knowledge derivation which formalizes all parts of the process.

The premise of our research is that knowledge derivation can be improved by incorporating trust and quality of data. Apart from trust approaches (e.g., EigenTrust [85], TrustRank [67], PowerTrust [187]) our framework is the only one that provides a formal approach to incorporating context, expected behavior and other meta information and deriving trust and quality assessments for data.

Instead of developing a specific quality and trust assessment approach that is limited to a particular scenario our research provides a generic abstract framework that is flexible and extensible enough to support existing as well as future knowledge derivation processes. Note that this includes approaches that only focus on a particular part of the framework such as large-scale data processing or modeling trust assessment techniques.

8.2 Trust Assessment and Management

In order to model knowledge derivation we need to make sure that we are able to evaluate the context of a particular piece of data. Specifically, incorporating trust and quality of data aspects requires a number of features to be supported by frameworks. First, determining what is fact and what not is based on a framework's ability to model heterogeneous and dynamic systems. Second, assessments of the reputation of data and data sources are crucial in evaluating their usefulness. Third, only full transparency throughout the knowledge derivation process ensures better decision making. An overview of supported trust assessment and management aspects of our and other frameworks is shown in table 8.2 and in the following sections we will discuss them in detail. Table 8.2 also includes references to the most relevant scenario that discusses a particular aspect.

Table 8.2: Trust Assessment and Management aspects comparison

Aspect	TrustKnowOne	Hadoop [169, 178]	Dryad [76]	Pregel [107]	Pegasus [86]	GraphLab [104]	EigenTrust [85]	TrustRank [67]	PowerTrust [187]	Scenario
8.2.1 Fact Finding and Data Representation										
heterogeneous entities	●	●	●	●	○	●	○	○	○	A
heterogeneous relationships	●	●	◒	◒	○	●	○	○	○	A
formal data representations	●	◒[1]	◒	◒	○	◒	○	○	○	A
quality and trust assessments	●	○	○	○	○	○	◒	◒	◒	R
dynamic entities	●	●	○	●	○	○	○	○	◒	R
dynamic relationships	●	●	○	●	○	○	○	○	◒	I
interfaces to other systems	◒	●	●	◒	○	○	○	○	○	R
8.2.2 Reputation Management										
data source assessments	●	○	○	○	○	○	◒	○	◒	R
objective challenges	◒	◒	◒	◒	○	◒	○	○	○	R
subjective challenges	●	○	○	○	○	○	●	◒	◒	I
applicable to multiple domains	●	●	○	○	○	◒	◒	◒	◒	A
formalization of trust aspects	●	○	○	○	○	○	◒	◒	◒	R
8.2.3 Data Lineage										
origin tracking	●	○	○	○	○	○	◒	○	○	R
processing lineage	●	◒	◒	◒	●	●	○	○	○	R
formalization of lineage	●	◒	◒	○	◒	◒	○	○	○	R
flexible granularity	●	○	○	○	○	○	○	○	○	R

○ no support ◒ partial support ● full support A Apps (6.1) R Radiation (6.2) I Intrusion (6.3)

[1] Available through data serialization frameworks (e.g., Avro [167], Thrift [154], Protocol Buffers [60])

8.2.1 Fact Finding and Data Representation

As shown in table 8.2, Hadoop provides the most complete approach to model flexible fact finding techniques. However, it does not incorporate trust and quality of data assessment. Heterogeneous systems require the ability to model a diverse set of data and relationships. Large-scale processing approaches have the advantage here as they are flexible and designed to be applicable in a variety of applications. In contrast, trust approaches are often designed for homogeneous systems and deal with relationships only as part of their trust assessment. We provide an *abstract graph model* which is able to model heterogeneous entities as well as relationships in a formal manner as *graph components* (chapter 4) and showed its application in chapter 6.

In addition to enabling its application in heterogeneous systems our approach overcomes the general lack of formal data representations found in other frameworks. For instance, there exist a number of serialization approaches for Hadoop (e.g., Avro [167], Thrift [154], Protocol Buffers [60]). However they do not provide a unified way of describing data and relationships. Another problem found in related approaches such as DFDL [135] and DSPL [62] is the combination of data source information and data description. As discussed in chapter 5 during the *knowledge extraction* our framework deals with data source and data element descriptions separately and defines a mapping using *adapters* which makes this approach more flexible. This flexibility can be seen in the use of our framework for a variety of scenarios (chapter 6).

The biggest challenge for fact finding is that trust and quality assessments are often not directly incorporated into knowledge derivation frameworks. Apart from the trust techniques (i.e., EigenTrust [85], TrustRank [67], PowerTrust [187]) our *TrustKnowOne* framework is the only one that provides mechanisms to incorporate trust and quality assessments in the form of formal *belief engines* (see section 6.1.4, 6.2.4, 6.3.3).

Furthermore, many "big data" analysis frameworks focus on efficient distributed processing on a static data set. However, many real world applications deal with data that is constantly changing. Thus it is necessary to consider time series of data and relationships. While Hadoop [169, 178] does not directly provide support for dynamic data, its input format is flexible enough to address this. Pregel [107] is based on computations perform on graph vertices and message passing between them. As such dynamic aspects could be implemented into the process directly. Our framework provides the flexible *abstract graph model* described in chapter 4 to provide this functionality. We utilized this aspect extensively in the radiation and intrusion detection scenarios (section 6.2, 6.3).

Note that while fact finding represents one of the foundations of knowledge derivation we need to ensure that aspects such as quality and trust assessments can be incorporated from other systems. This means providing a way to query facts as well as annotating or modifying them which requires a flexible approach such as our *abstract graph model* (see chapter 4, section 6.2.4.1) or separate interfaces as in the case of Hadoop [169, 178], Dryad [76], and Pregel [107].

8.2.2 Reputation Management

Reputation management is an important component of fact finding that allows us to weight data and resolve conflicting information. Many of the large-scale processing and graph frameworks do not consider data coming from different sources (e.g., section 6.2) but assume a single data set. Distributed trust approaches such as EigenTrust [85] and PowerTrust [187] incorporate a limited version of reputation of data sources as part of their peer assessments. Our *TrustKnowOne* framework provides a formalization of “local” assessments through *dimension models* during *knowledge extraction* and “global” assessments through *belief engines* during *knowledge processing* (see chapter 5, section 6.2.5.5).

Furthermore, knowledge derivation frameworks need to be able to deal with objective and subjective challenges to data. However, most frameworks do not address both but focus on one or the other. In particular, large-scale processing approaches often incorporate techniques that ensure data consistency and fault tolerance and ignore attack models that would introduce malicious or modify existing data. Trust approaches on the other hand provide the means to identify and often deal with attacks on data. Note that our *TrustKnowOne* framework does not specify particular techniques to deal with either objective or subjective challenges but instead provides a flexible approach based on *belief engines* modeled as *graph expressions* that allows existing and future techniques to be implemented (see section 4.4, section 6.3)

In addition, this formalization is necessary to enable reuse of trust and quality assessment techniques in other application domains. Most of the representative frameworks do not provide mechanisms to incorporate other approaches. Furthermore, while the trust approaches are in some way formally specified their evaluation and comparison against each other is still difficult because of a missing unified formalization model. By defining trust approaches as *graph expressions* that are evaluated on an *abstract graph model* storing the data we solve this problem and enable evaluation and comparison of the approaches (see section 6.2.5, section 6.3.5.3).

8.2.3 Data Lineage

The formalization of the entire knowledge derivation processing as provided by our *TrustKnowOne* framework has the advantage of allowing detailed tracing of data and assessments (chapter 5). In particular, the *knowledge extraction* phase annotates data with meta information about its origin while the *knowledge processing* phase provides additional information about how it is processed. This approach enables us to assess why we reach certain decision results and trace back influential data as well as processes that shaped the decision making. Furthermore, in contrast to other frameworks we are able to use this extensive lineage information to evaluate and improve knowledge derivation (e.g., not incorporating data from untrustworthy sources, only using a subset of the available data, see section 6.2.5.5). In addition, since we provide lineage information as additional attributes to *element nodes* in our *abstract graph model* this approach is flexible enough to allow for varying degrees of granularity ranging from capturing straightforward origin meta information to detailed processing and assessment derivation annotations.

8.3 Data Modeling, Integration, and Fusion

Knowledge derivation is complex because it requires a variety of data processing problems to be addressed. First, frameworks need to provide capabilities to formalize aspects of data processing such as data sources, relationships, assessments and decision processes. This is an important step that is often neglected by existing research resulting in frameworks that are difficult to evaluate and compare. Second, trust and quality of data assessments need to be directly incorporated into the overall knowledge derivation process. Separating data processing from assessments limited our ability to provide the best possible decision options. Third, in order to enable evaluation and comparison of approaches as well as improve the knowledge derivation process we need to be able to define metrics and assess costs. In this section we will discuss data modeling, integration, and fusion aspects of our and other frameworks (see table 8.3 for an overview). In addition, table 8.2 indicates the most relevant scenario discussing these aspect.

8.3.1 Formalization

Many data processing frameworks provide mechanisms to describe the data they use as part of their processing. For instance, there exist a number of serialization frameworks for Hadoop [169, 178] (e.g., Avro [167], Thrift [154], Protocol Buffers [60]) and graph frameworks usually allows arbitrary data to be associated with graph nodes. The problem lies in the fact that without a formal description it becomes difficult to exchange

Table 8.3: Data Modeling, Integration, and Fusion aspects comparison

Aspect	TrustKnowOne	Hadoop [169, 178]	Dryad [76]	Pregel [107]	Pegasus [86]	GraphLab [104]	EigenTrust [85]	TrustRank [67]	PowerTrust [187]	Scenario
8.3.1 Formalization										
description of data elements	●	◒[2]	◒	◒	◒	◒	○	○	○	A
description of data sources	●	◒	○	○	○	○	○	○	○	R
description of relationships	●	◒	◒	◒	◒	◒	○	○	○	A
meta information	●	◒	○	○	○	○	○	○	◒	R
belief engines	●	○	○	○	○	○	◒	◒	◒	R
decision processes	●	◒[3]	○	○	○	○	○	○	○	R
time and location variant	●	◒	○	○	○	○	○	○	◒	R
8.3.2 Trust and Relationship Models										
homogeneous systems	●	○	○	○	○	○	◒	◒	◒	I
heterogeneous systems	●	○	○	○	○	○	◒	◒	◒	A
formal trust integration	●	○	○	○	○	○	◒	◒	◒	A
8.3.3 Metrics										
cost assessments	●	◒	◒	○	◒	○	○	○	○	A
knowledge extraction	●	◒	○	○	○	○	○	○	○	A
knowledge processing	●	◒	◒	○	◒	○	○	○	○	I
knowledge evaluation	●	◒	○	○	○	○	○	○	○	R
formalization of trust metrics	●	○	○	○	○	○	◒	◒	◒	R

○ no support ◒ partial support ● full support A Apps (6.1) R Radiation (6.2) I Intrusion (6.3)

[2] Available through data serialization frameworks (e.g., Avro [167], Thrift [154], Protocol Buffers [60])
[3] Available through built-in functionality and extensions (e.g., Mahout [121, 171], Giraph [168])

data and use it in another application domain. In addition, while our framework directly incorporates aspects such as different classes of nodes and node identities other frameworks do not (see section 5.2). Similar formalization problems affect descriptions of relationships which are often only provided as adjacency lists or matrices. In general, knowledge derivation frameworks tend to ignore heterogeneous aspects that could prove useful but increase complexity such as relationship attribute (e.g., time, weights). In addition, the lack of formalizing data sources, extracting knowledge from them, and their incorporation into the knowledge derivation process makes it difficult to provide proper data lineage and "local" assessments for data. As shown in chapter 5 *knowledge extraction* is fully formalized in our framework. This includes provisions for adding meta information and context to the data as part of *graph components* within the *abstract graph model* (chapter 4).

Because many of the frameworks do not incorporate trust aspects there is no formal approach to modeling trust and quality of data assessments. Note that while trust approaches such as EigenTrust [85], TrustRank [67], and PowerTrust [187] provide a formalization of their techniques they are not generic enough to describe other assessment approaches. Approaches for decision making are often not part of frameworks because they are considered separate from the data processing. However, Hadoop [169, 178] includes some built-in functionality and there exist several extensions (e.g., Mahout [121, 171], Giraph [168]) for formalizing decision processes. Our approach allows data processing, assessment and decision making to be formalized using *graph expressions* (section 4.4). This provides a unified approach that is flexible enough for existing and extensible for future approaches and techniques (see chapter 6).

As discussed earlier, we need to be able to analyze data that is dynamic. Hence, our data model needs to be able to formally describe dynamic aspects such as time and location data and relationships (e.g., section 6.2, 6.3). Our *abstract graph model* inherently stores keeps track of data as a time series and we provide *dimension models* for expressing dynamic data (chapter 4). Hadoop [169, 178] allows processing on arbitrary data and as such provides limited support for this type of dynamic modeling. PowerTrust [187] partially addresses the issue in terms of peer nodes joining or leaving the system.

8.3.2 Trust and Relationship Models

Trust and relation models are often of limited relevance in large-scale data processing frameworks. Similarly, trust approaches (e.g., EigenTrust [85], TrustRank [67], PowerTrust [187]) often ignore data processing aspects and focus on trust and relationship models. In general, it is important to provide support for homogeneous and heterogeneous systems in order for trust approaches to be applicable to a wide range of scenarios. This means that trust and relationship model need to be formal in their description yet flexible enough to support a wide variety of domains. Our approach of expressing data as *graph components* and relationships as *relation edges* between them achieves this. Furthermore, trust approaches can then be modeled as *graph expressions* in a standardized manner (see chapter 6).

As discussed above, providing a formal integration of trust aspects into the knowledge derivation process can improve decision making. In particular, having “local” (*dimension models* chapter 4) and “global” (*belief engines* chapter 5) assessments available allows us to determine what data is of high quality and trustworthy as well as provide confidence levels for decision options.

8.3.3 Metrics

Providing cost assessments of individual components in large scale systems is difficult due to complexity and various interconnecting components being formalized using different methodologies. Hadoop [169, 178], Dryad [76], and Pegasus [86] utilize a limited set of predefined counters and computational statistics for monitoring and analysis of data processing. We allow cost assessments to be attached to data (*graph compo-*

nents) as well as processes (*graph expressions*) which enables evaluation and comparison of data processing and assessment techniques (section 5.2.6, 5.3.6, 5.4.5).

Furthermore, while optimization is often considered one of the primary reasons for defining metrics, many frameworks provide only limited capabilities to formally describe and incorporate them. This means that it is difficult to implement other metrics than the ones existing in the frameworks. Our *TrustKnowOne* framework is based on a flexible *abstract graph model* on which metrics can be defined and evaluated using *graph expressions* (chapter 4). These metrics can then be calculated at each phase (chapter 5) and thus cover the entire knowledge derivation process.

It is important to provide trust metrics that reflect aspects of trust relationships such as confidence in data and reputation of data sources. Especially the assessment of data sources is only factored in in trust approaches and mostly ignored in data processing frameworks. For instance, as part of our framework meta information such as lineage can be used in combination with these assessments to improve decision processes by determining the trustworthiness of data sources (see section 6.2.5.5). This is not possible in frameworks that do not provide formal trust metrics.

8.4 Large-scale Data Processing

In order to provide the scalability and flexibility necessary to model complex application scenarios, frameworks should support a variety of aspects related to “big data”. In particular, scalability is often closely tied to distributed processing whereas flexibility in terms of data modeling comes from approaches that are based in graph theory. As such, large-scale processing requires combination of “big data” and graph framework aspects that maintains the advantages of both areas. Furthermore, every framework needs to be able to provide mechanisms for data processing as well as analysis of data, assessments, and processes in form of a query system. Without the ability to retrieve additional information such as context, cost assessments, and metrics evaluation and comparison becomes difficult. We provide an overview of the supported large-scale data processing aspects in table 8.4 and will discuss details as part of this section. Table 8.4 also shows which of scenario is the most relevant regarding the discussion of a specific aspect or confirmed through reference implementation.

8.4.1 Big Data

There exist a variety of approaches to achieve scalability. However, most important is the choice of flexible data structures. In particular with regards to storage, data structures need to be able to deal with being distributed across various entities such as computing clusters. Furthermore, basic operations such as adding, modifying, or removing data needs to be bound by the number of data entries and should not be exponential. For this reason, using matrix data structures with a large number of values can become problematic. However, most large-scale data processing frameworks have solved this problem either through efficient data structures or scalable implementation of processes.

An approach to solve the problem of storage and computational power limitations of single systems is to perform processing in a distributed manner. Note that this can be considered a natural evolution from single-threaded sequential processing on a single machine to multi-threaded parallel processing on a single machine and finally to parallel processing on multiple machines. Many frameworks have focused on providing this capability through a variety of abstract programming paradigms (e.g., MapReduce [37–39] in Hadoop [169, 178], vertex-based iterative computations in Pregel [107]). Our framework is provides an *abstract graph model* on which *graph expressions* are evaluated (chapter 4). As such nodes as well as edges of the graph can be distributed across machines.

Note that while platform independence is something that is often not considered a priority when designing frameworks, it leads to additional flexibility. In particular, it

Table 8.4: Large-scale Data Processing aspects comparison

Aspect	TrustKnowOne	Hadoop [169, 178]	Dryad [76]	Pregel [107]	Pegasus [86]	GraphLab [104]	EigenTrust [85]	TrustRank [67]	PowerTrust [187]	Scenario
8.4.1 Big Data										
scalability	●	●	●	●	●	●	◒	◒	●	R
distributed processing	●	●	●	●	●	●[4]	●	◒	●	🟩
platform independent	●	●	●	●	◒	●	●	●	●	🟩
flexible input formats	●	●[5]	●	●	○	○	○	○	○	🟩
flexible storage options	●	●	●	●	○	○	○	○	○	🟩
flexible computation engines	●	◒	●	◒	○	◒	○	○	○	🟩
flexible output formats	●	●[5]	●	●	○	○	○	○	○	🟩
incorporation of trust aspects	●	○	○	○	○	○	◒	◒	◒	R
8.4.2 Graph Frameworks										
heterogeneous nodes	●	○	○	○	○	●	○	○	○	A
heterogeneous edges	●	○	○	○	○	●	○	○	○	A
dynamic graphs	●	◒[6]	○	◒	○	○	○	○	◒	I
meta information	●	○	○	○	○	◒	○	○	◒	R
applicable in multiple domains	●	●[6]	○	●	◒	●	○	○	◒	A
8.4.3 Query Systems										
flexibility	●	●[7]	●[8]	◒	◒	◒	○	○	○	A
expressiveness	◒	◒[7]	◒[8]	○	○	○	○	○	○	R
incorporation of trust aspects	●	○	○	○	○	○	◒	◒	◒	R

○ no support ◒ partial support ● full support A Apps (6.1) R Radiation (6.2) I Intrusion (6.3) 🟩 Implementation (7)

[4] Available through an extension of the work called Distributed GraphLab [105]
[5] Available through built-in functionality and data serialization frameworks (e.g., Avro [167], Thrift [154], Protocol Buffers [60])
[6] Available through graph (e.g., Giraph [168]) and machine learning (e.g., Mahout [171]) extensions
[7] Available through query systems built on top of Hadoop (e.g., Pig [118, 172], Hive [170, 173])
[8] Available through query systems built on top of Dryad (e.g., DryadLINQ [186], SCOPE [24])

allows data processing and assessment approaches to be applicable to more scenarios because they can be incorporated into other systems more easily. Similarly, maintaining flexibility in input and output formats is important ensures that a framework can deal not just with existing data sources but future ones as well. Apart from *TrustKnowOne* which uses an *adapter* approach (chapter 5) general large-scale processing frameworks (e.g., Hadoop [169, 178], Dryad [76]) support this kind of integration while trust approaches (e.g., EigenTrust [85], TrustRank [67], PowerTrust [187]) do not.

We already discussed the need to flexible data storage options in terms of scalability. In addition, note that providing multiple storage options through a common abstract interface frameworks could adapt better to various scenarios (see chapter 6). Specifically, data for large-scale data analysis could be stored based on how data is used. For instance, frequently used data can be stored in memory for faster processing while historic data can be stored in databases. This kind of approach for splitting the data across multiple storage systems is supported by our framework (chapter 7). Other frameworks provide a variety of file systems (Hadoop [169, 178]) or process-based storage (e.g., Dryad [76]) but many simply define their own storage approach that often does not translate to other domains.

There exist several approaches to perform large-scale data processing, most notably MapReduce [37–39] (e.g., in Hadoop [169, 178]), user-defined computational topologies (e.g., in Dryad [76]), and vertex-based iterative computations (e.g., in Pregel [107]). Our approach does not dictate one specifically but rather provides a framework based an *abstract graph model* that is conducive to existing as well as future computational paradigms (chapter 4, 5, 7). In particular, MapReduce can be modeled as a sequence of *graph expressions* that represent "map" and "reduce" functions accordingly. Furthermore, *graph expressions* can be nested and thus be used to represent computational dependencies (chapter 4) as in Dryad. Finally, since *graph components* in our *abstract graph model* can be annotated we are able to keep track of information across several iterations while evaluating *graph expressions* in parallel on nodes of the graph like Pregel.

As such our *TrustKnowOne* framework provides a flexible approach to large-scale data processing as compared to the existing frameworks. Note that it is also important to distinguish between defined and derived knowledge. While it is often straightforward to define data processing techniques our framework includes the ability to incorporate meta information and various assessments to derive additional knowledge to be used during decision making.

One of the premises of our research is the incorporation of trust and quality of data into the knowledge derivation process. Large-scale data processing and context analysis such as trust or quality assessments are often seen as diverging interests. The problem is that in order to allow for the fastest and most efficient large-scale processing one needs to ignore these aspects in favor of performance because they increase complexity. However, while providing trust and quality of data assessments decreases performance it can improve decision processes and in some instances overcome additional computational costs (see chapter 6). For instance, determining which data sources are less trustworthy or which data elements are of low quality we could optimize knowledge derivation and thus decision making processes by only performing computations on high trustworthy and high quality data (e.g., section 6.2.5.5). Approaches like the one described would be able to potentially balance out the decrease in performance that is due to the incorporation of trust and quality of data aspects. However, in contrast to the general large-scale processing approaches our *TrustKnowOne* framework is be able to provide a better assessment of decision options that includes confidence levels and ranges.

8.4.2 Graph Frameworks

With the growing number of relationships in data that occur in many real world scenarios (e.g., social networks, distributed sensing), graph frameworks represent a natural fit for providing extensive and yet flexible data modeling. However, it is key to maintain this flexibility by having a graph framework that is able to incorporate heterogeneous entities and relationships. Many large-scale processing frameworks do not provide a graph-based interface which puts the burden of modeling possibly extensive relationships on the developer. In addition, it can lead to custom non-standard representations of graph concepts that are difficult to evaluate, improve, or adapt to other domains. Our *TrustKnowOne* framework uses an *abstract graph model* which as described in chapter 4 is flexible enough to model current and future application scenarios (see section 6.1.2, 6.2.2, 6.3.1). Note that GraphLab [104] is the only one of the representative frameworks that is entirely based on graph theoretic constructs and allows arbitrary data to be stored with graph nodes and edges.

As discussed above another problem with many frameworks is the lack of support for dynamic aspects. With regards to graph frameworks we need to be able to store data as time series in nodes and edges. Furthermore, graph topology may change over time as new nodes are added, nodes are removed, and relationships adjusted. Our *abstract graph model* provides an approach where all data is timestamped which means that we are able to trace back when a relationship was created, adjusted, or removed. This allows us to incorporate dynamic aspects into the knowledge derivation process (see section 6.2, 6.3).

Furthermore, being able to provide methods for associating meta information with nodes and edges in a graph model is important. Without meta information frameworks are able to perform only limited assessments of data, data sources and processes. In addition, it makes it difficult to adapt processing and assessment approaches to new application scenarios. Note that as described in chapter 5 we discuss how we are able to utilize meta information to derive trust and quality assessments as well as provide cost assessments and data lineage.

Overall the aspects described here for graph frameworks determine whether or not a framework is flexible enough to be applicable to multiple domains. Note that of the representative frameworks some can be considered to support this flexibility through the means of graph extensions (e.g., Giraph [168], Mahout [171]). As shown in the application scenarios (chapter 6), our *TrustKnowOne* framework with its *abstract graph model* can be applied to multiple domains effectively.

8.4.3 Query Systems

The main focus of data processing frameworks is to provide support for mechanisms that derive knowledge from data. However, as part of evaluating and comparing trust and quality of data approaches we need to have techniques available that enable metrics and cost assessments to be determined. In particular, we require a flexible and extensible approach to retrieve data and assessments. Furthermore, this approach needs to able to express complex heterogeneous relationships between data. There exist several extensions for large-scale data processing frameworks such as Pig [118, 172] and Hive [170, 173] for Hadoop [169, 178] as well as DryadLINQ [186] and SCOPE [24] for Dryad. Our *TrustKnowOne* provides a flexible and expressive approach using the *abstract graph model* presented in chapter 4 where *graph expressions* are evaluated on *graph components*.

Furthermore, with the incorporation of trust and quality of data assessments into the knowledge derivation process one should be able to provide an approach to include those into the "big data" analysis part as well. This means that instead of treating assessments as secondary results of knowledge derivation, decision processes can be improved by utilizing assessments throughout data processing phases (e.g., only utilize data from trustworthy data sources).

8.5 Chapter Summary

The goal of this chapter is to put our research into context and evaluate how our *TrustKnowOne* framework compares to other representative frameworks in a variety of aspects. Existing frameworks deal with large-scale data processing (i.e., Hadoop [169, 178], Dryad [76]), graph algorithms (i.e., Pregel [107], Pegasus [86], GraphLab [104]), and distributed trust approaches (i.e., EigenTrust [85], TrustRank [67], PowerTrust [187]).

In contrast to our framework none of the existing representative systems are able to incorporate all major aspects of knowledge derivation. Modeling heterogeneous and dynamic systems extensively is often dismissed as the focus of many frameworks is large-scale data processing where performance is most important. Furthermore, the lack of formal representations of data, data sources, and processes makes it difficult to improve upon and reuse existing approaches as well as incorporate future techniques. This is in spite of the flexibility of many existing frameworks with regards to platforms, programming languages, storage options, and computational engines. However, one of the main premises of our work is to incorporate trust and quality of data into the knowledge derivation process. As such, our framework is the only one that fully supports both large-scale data processing as well as quality and trust assessments.

As part of the evaluation we also provide a more detailed assessment of aspects in the areas defined by the related work chapter (chapter 3). First, trust assessment and management deals with fact finding and data representation as well as reputation management and data lineage. Here it is important to note that we provide formal approaches to problems such as incorporating reputation and lineage into the knowledge derivation process. In particular, our *TrustKnowOne* framework enables the modeling of various trust and quality aspects in order to determine the value of data for knowledge derivation.

Second, data modeling, integration, and fusion relates to the topics of formalization, trust and relationship models, and metrics. As discussed throughout this research, formalization allows evaluation, comparison, improvement, and exchange of data and approaches. Our approach consists of a formal *abstract graph model Berlin* and a formal framework *TrustKnowOne* to model the entire knowledge derivation process. Many of the other frameworks provide limited formalization that is often focused on their domain and not able to generalize well. In addition, models for aspects such as meta information and dynamic systems (e.g., time, location) are missing in many frameworks. As large-scale data processing frameworks focus on performance and disregard trust and relationship models the only approaches for formalizing them comes from trust techniques and frameworks. Furthermore, evaluation and comparison of data processing and assessment approaches is dependent on the existence of formal metrics. While our framework enables the definition of various metrics such as cost assessments and lineage in a flexible manner using *graph expressions* (chapter 5) other existing frameworks do not.

Third, large-scale data processing is an important area as the available data that can be used for analysis and decision making has increased drastically. In the era of "big data" there are a number of necessary aspects that every data processing framework needs to address. This is a field in which research has yielded a variety of flexible, distributed, and high performance frameworks (i.e., Hadoop [169, 178], Dryad [76], Pregel [107], etc.). The main issue here is the lack of natural support for trust and quality of data aspects. Furthermore, because there is a growing number of applications that require extensive relationship modeling, graph frameworks have been identified as possible solutions. However, combining aspects of high performance while maintaining flexibility for heterogeneous and dynamic systems is difficult and thus addressed only partially in many frameworks. Our *abstract graph model Berlin* provides an approach that is able to solve this problem (chapter 4). With the amount of data that is analyzed and processed it also becomes important to enable techniques to be incorporated for analyzing data processing and assessment approaches. In particular, frameworks should be able to provide information about how decisions were made and knowledge was derived. Furthermore, when trust and quality assessments are available one needs to be able to incorporate them and other meta information into decision making and data analysis.

9 Conclusion

In this dissertation we presented *TrustKnowOne*, a framework for knowledge derivation incorporating trust and quality of data. Our framework is based on the *Berlin abstract graph model* on which formal *graph expressions* are evaluated. The combination of *graph expressions* in the form of *metrics* forms the basis of trust and data quality assessments. Through these *metrics* we incorporate context, historical behavior, and other meta information as well as relationships between data elements that can be used to implement a variety of *knowledge processing* approaches, *belief engines*, and *decision processes*.

Throughout this dissertation we described in detail our approaches and methods in developing this novel framework. Furthermore, we applied the *TrustKnowOne* framework to three diverse and realistic scenarios to showcase its formalization capabilities as well as its flexibility. As such we demonstrated throughout this dissertation that our research yields a number of key contributions:

- A new abstract graph modeling approach that allows the management of heterogeneous data with dynamic aspects (e.g., time, location) in a variety of application scenarios while inherently incorporating trustworthiness and data quality assessments

- A new formalization approach to describing belief engines and decision processes in the form of graph *expressions*
- A new framework for knowledge derivation that provides a flexible and extensible approach using clearly defined *extraction*, *processing*, and *evaluation* components
- The means to evaluate and compare different belief, trustworthiness, and decision making techniques in a variety of application scenarios using a formal approach

9.1 Future work

We have identified the following areas of our framework for further research.

Computation engines As there exist a number of large-scale data processing approaches, we need to explore in more depth how our *graph expressions* can be efficiently distributed and evaluated in parallel. Furthermore, a computation engine needs to be able to optimize individual as well as groups of *graph expressions* in order to achieve scalability.

Graph model backends While our reference implementation provides two backends (i.e., in-memory and graph database) there is the need to further evaluate the efficient storage and management of the *abstract graph model.* Specifically, distributed approaches such as Hadoop provide inherent benefits that our framework could make use of.

Objective challenges Our framework and in particular the *abstract graph model* supports the notion of objective challenges such environmental factors, calibration issues, and time variance through dimension models. However, despite the usefulness of providing generalized assessments modeling these object challenges it is difficult to construct a repository for these models such that they can be shared among applications.

Query system Using *graph expressions* the data stored in the *abstract graph model* can be processed, compared and evaluated. However, our approach requires thinking in a new declarative paradigm which may make it difficult to transform existing knowledge derivation processes.

References

[1] Jazzy - The Java Open Source Spell Checker. URL `jazzy.sourceforge.net`.

[2] Edoardo M Airoldi. Getting started in probabilistic graphical models. *PLoS computational biology*, 3(12):e252, December 2007. ISSN 1553-7358. doi: 10.1371/journal.pcbi.0030252. URL `http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2134967&tool=pmcentrez&rendertype=abstract`.

[3] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, December 1974. ISSN 0018-9286. doi: 10.1109/TAC.1974.1100705. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1100705`.

[4] Constantin F. Aliferis, Alexander Statnikov, Ioannis Tsamardinos, Subramani Mani, and Xenofon D. Koutsoukos. Local Causal and Markov Blanket Induction for Causal Discovery and Feature Selection for Classification Part I : Algorithms and Empirical Evaluation. *Journal of Machine Learning Research*, 11:171–234, 2010.

[5] T. W. Anderson and D. A. Darling. Asymptotic Theory of Certain "Goodness of Fit" Criteria Based on Stochastic Processes. *The Annals of Mathematical Statistics*, 23(2):193–212, June 1952. ISSN 0003-4851. doi: 10.1214/aoms/1177729437. URL `http://projecteuclid.org/euclid.aoms/1177729437`.

[6] Oana Andrei, Maribel Fernández, Hélène Kirchner, Guy Melançon, Olivier Namet, and Bruno Pinaud. PORGY: Strategy-Driven Interactive Transformation of Graphs. *Electronic Proceedings in Theoretical Computer Science*, 48(Termgraph): 54–68, February 2011. ISSN 2075-2180. doi: 10.4204/EPTCS.48.7. URL `http://arxiv.org/abs/1102.2654v1`.

[7] Jeremy Andrus, Christoffer Dall, Alexander Van't Hof, Oren Laadan, and Jason Nieh. Cells: a virtual mobile smartphone architecture. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles - SOSP '11*, page 173, New York, New York, USA, 2011. ACM Press. ISBN 9781450309776. doi: 10.1145/2043556.2043574. URL `http://dl.acm.org/citation.cfm?doid=2043556.2043574`.

[8] Raju Balakrishnan and Subbarao Kambhampati. SourceRank. In *Proceedings of the 20th international conference on World wide web - WWW '11*, page 227, New York, New York, USA, 2011. ACM Press. ISBN 9781450306324. doi: 10.1145/1963405.1963440. URL `http://portal.acm.org/citation.cfm?doid=1963405.1963440`.

[9] J.S. Balasubramaniyan, J.O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni. An architecture for intrusion detection using autonomous agents. In *Proceedings 14th Annual Computer Security Applications Conference (Cat. No.98EX217)*, pages 13–24. IEEE Comput. Soc. ISBN 0-8186-8789-4. doi: 10.1109/CSAC.1998.738563. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=738563`.

[10] Lerone Banks and Shyhtsun Felix Wu. All Friends Are Not Created Equal: An Interaction Intensity Based Approach to Privacy in Online Social Networks. In *2009 International Conference on Computational Science and Engineering*, pages 970–974. IEEE, 2009. ISBN 978-1-4244-5334-4. doi: 10.1109/CSE.2009.429. URL `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5283725`.

[11] Wang Bao-Jun and Zhan Ying. A survey and performance evaluation on sliding window for data stream. In *2011 IEEE 3rd International Conference on Communication Software and Networks*, volume 1, pages 654–657. IEEE, May 2011. ISBN 978-1-61284-485-5. doi: 10.1109/ICCSN.2011.6014977. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6014977`.

[12] R.W. Beard. Consensus seeking in multiagent systems under dynamically changing interaction topologies. *IEEE Transactions on Automatic Control*, 50(5):655–661, May 2005. ISSN 0018-9286. doi: 10.1109/TAC.2005.846556. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1431045`.

[13] David E. Bell. Disappointment in Decision Making Under Uncertainty. *Operations Research*, 33(1):1–27, 1985. ISSN 0030364X. doi: 10.1287/opre.33.1.1. URL `http://or.journal.informs.org/cgi/doi/10.1287/opre.33.1.1`.

[14] A. Benavoli, B. Ristic, A. Farina, M. Oxenham, and L. Chisci. An Application of Evidential Networks to Threat Assessment. *IEEE Transactions on Aerospace and Electronic Systems*, 45(2):620–639, April 2009. ISSN 0018-9251. doi: 10.1109/TAES.2009.5089545. URL `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5089545`.

[15] Salem Benferhat. Causal reasoning in graphical models. In *2010 International Conference on Machine and Web Intelligence*, pages 15–19. IEEE, October 2010. ISBN 978-1-4244-8608-3. doi: 10.1109/ICMWI.2010.5647857. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5647857`.

[16] Omar Benjelloun, Anish Das Sarma, Alon Halevy, Martin Theobald, and Jennifer Widom. Databases with uncertainty and lineage. *The VLDB Journal*, 17(2):

243–264, January 2008. ISSN 1066-8888. doi: 10.1007/s00778-007-0080-z. URL http://www.springerlink.com/index/10.1007/s00778-007-0080-z.

[17] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Comput. Soc. Press, 1996. ISBN 0-8186-7417-2. doi: 10.1109/SECPRI. 1996.502679. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=502679.

[18] Kenneth A Bollen and Robert W Jackman. Regression diagnostics: An expository treatment of outliers and influential cases. *Modern Methods of Data Analysis*, pages 257—-291, 1990.

[19] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications.* Elsevier Science Publishing Co., Inc., 1976. ISBN 0-444-19451-7. URL http://people.math.jussieu.fr/~jabondy/books/gtwa/pdf.

[20] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, April 1998. ISSN 01697552. doi: 10.1016/S0169-7552(98)00110-X. URL http://linkinghub.elsevier.com/retrieve/pii/S016975529800110X.

[21] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. Efficient processing of spatial joins using R-trees. *ACM SIGMOD Record*, 22(2):237–246, June 1993. ISSN 01635808. doi: 10.1145/170036.170075. URL http://portal.acm.org/citation.cfm?doid=170036.170075.

[22] Sergey Bykov, Alan Geller, Gabriel Kliot, James R Larus, Ravi Pandya, and Jorgen Thelin. Orleans. In *Proceedings of the 2nd ACM Symposium on Cloud Computing - SOCC '11*, pages 1–14, New York, New York, USA, 2011. ACM Press. ISBN 9781450309769. doi: 10.1145/2038916.2038932. URL http://dl.acm.org/citation.cfm?doid=2038916.2038932.

[23] C. Castelfranchi and R. Falcone. Principles of Trust for MAS: Cognitive Anatomy, Social Importance, and Quantification. In *ICMAS '98 Proceedings of the 3rd International Conference on Multi Agent Systems*, 1998.

[24] Ronnie Chaiken, Bob Jenkins, Per-å ke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. SCOPE : Easy and Efficient Parallel Processing of Massive Data Sets. *Proceedings of the VLDB Endowment*, 1(2):1265–1276, 2008.

[25] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, Generators, and Algorithms. *ACM Computing Surveys*, 38(1):2–es, June 2006. ISSN 03600300. doi: 10.1145/1132952.1132954. URL http://portal.acm.org/citation.cfm?doid=1132952.1132954.

[26] Subhash Challa and Mohammad Momani. Survey of trust models in different network domains. *CoRR*, 2010. doi: abs/1010.0168.

[27] Leisong Chen and Guoping Lin. Extending Sliding-Window Semantics over Data Streams. In *2008 International Symposium on Computer Science and Computational Technology*, pages 110–113. IEEE, 2008. ISBN 978-0-7695-3498-5. doi: 10.1109/ISCSCT.2008.187. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4731583`.

[28] Jong-sheng Cherng and Mei-jung Lo. A hypergraph based clustering algorithm for spatial data sets. In *Proceedings 2001 IEEE International Conference on Data Mining*, pages 83–90. IEEE Comput. Soc, 2001. ISBN 0-7695-1119-8. doi: 10.1109/ICDM.2001.989504. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=989504`.

[29] Mung Chiang. *Networked Life: 20 Questions and Answers.* Cambridge University Press, 2012. ISBN 1107024943. URL `http://www.amazon.com/Networked-Life-20-Questions-Answers/dp/1107024943`.

[30] Jin-Hee Cho, Ananthram Swami, and Ing-Ray Chen. Mission-Dependent Trust Management in Heterogeneous Military Mobile Ad Hoc Networks. In *15th ICCRTS*, volume 20783. DTIC Document, 2010. ISBN 3013940492. URL `http://oai.dtic.mil/oai/oai?verb=getRecord&amp;metadataPrefix=html&amp;identifier=ADA525195`.

[31] Jin-Hee Cho, Ananthram Swami, and Ing-Ray Chen. A Survey on Trust Management for Mobile Ad Hoc Networks. *IEEE Communications Surveys & Tutorials*, 13(4):562–583, 2011. ISSN 1553-877X. doi: 10.1109/SURV.2011.092110.00088. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5604602`.

[32] David Chu, A. Deshpande, and J.M. Hellerstein. Approximate Data Collection in Sensor Networks using Probabilistic Models. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 48–48. IEEE, 2006. ISBN 0-7695-2570-9. doi: 10.1109/ICDE.2006.21. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1617416`.

[33] B.N. Chun and A. Bavier. Decentralized trust management and accountability in federated systems. In *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*, volume 00, page 9 pp. IEEE, 2004. ISBN 0-7695-2056-1. doi: 10.1109/HICSS.2004.1265656. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1265656`.

[34] Dorin Comaniciu and Peter Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, May 2002. ISSN 01628828. doi: 10.1109/34.1000236. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1000236`.

[35] R. Dennis Cook. Detection of Influential Observation in Linear Regression. *Technometrics*, 19(1):15, February 1977. ISSN 00401706. doi: 10.2307/1268249. URL `http://www.jstor.org/stable/1268249?origin=crossref`.

[36] Nilesh Dalvi and Dan Suciu. Management of probabilistic data. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '07*, pages 1–12, New York, New York, USA, 2007. ACM Press. ISBN 9781595936851. doi: 10.1145/1265530.1265531. URL `http://portal.acm.org/citation.cfm?doid=1265530.1265531`.

[37] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, pages 10–10, Berkeley, 2004. USENIX Association. URL `http://portal.acm.org/citation.cfm?id=1251254.1251264`.

[38] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107, January 2008. ISSN 00010782. doi: 10.1145/1327452.1327492. URL `http://dx.doi.org/10.1145/1327452.1327492`.

[39] Jeffrey Dean and Sanjay Ghemawat. MapReduce: a flexible data processing tool. *Communications of the ACM*, 53(1), 2010. ISSN 0001-0782. URL `http://portal.acm.org/citation.cfm?doid=1629175.1629198`.

[40] Meghana Deodhar and Joydeep Ghosh. Scoal. *ACM Transactions on Knowledge Discovery from Data*, 4(3):1–31, October 2010. ISSN 15564681. doi: 10.1145/1839490.1839492. URL `http://portal.acm.org/citation.cfm?doid=1839490.1839492`.

[41] Disqus. Pseudonyms drive communities, 2012. URL `http://disqus.com/research/pseudonyms/`.

[42] Leigh Dodds and Ian Davis. *Linked Data Patterns*. 2012-05-31 edition, 2012. URL `http://patterns.dataincubator.org`.

[43] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. Truth discovery and copying detection in a dynamic world. *Proceedings of the VLDB Endowment*, 2 (1), 2009.

[44] Marek J. Druzdzel and Francisco J. Díez. Combining Knowledge from Different Sources in Causal Probabilistic Models. *Journal of Machine Learning Research*, 4:295–316, 2003.

[45] Duo Security. X-Ray, 2012. URL `http://www.xray.io/`.

[46] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*, pages 226–231, 1996.

[47] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security - CCS '11*, page 627, New

York, New York, USA, October 2011. ACM Press. ISBN 9781450309486. doi: 10.1145/2046707.2046779. URL http://dl.acm.org/citation.cfm?id=2046707.2046779.

[48] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices - SPSM '11*, page 3, New York, New York, USA, October 2011. ACM Press. ISBN 9781450310000. doi: 10.1145/2046614.2046618. URL http://dl.acm.org/citation.cfm?id=2046614.2046618.

[49] Adrienne Porter Felt, Kate Greenwood, and David Wagner. The effectiveness of application permissions. In *WebApps'11 Proceedings of the 2nd USENIX conference on Web application development*, page 7, June 2011. URL http://dl.acm.org/citation.cfm?id=2002168.2002175.

[50] Richard Fernandes and Sylvain G. Leblanc. Parametric (modified least squares) and non-parametric (Theil–Sen) linear regressions for predicting biophysical parameters in the presence of measurement errors. *Remote Sensing of Environment*, 95(3):303–316, April 2005. ISSN 00344257. doi: 10.1016/j.rse.2005.01.005. URL http://linkinghub.elsevier.com/retrieve/pii/S0034425705000404.

[51] David Ferrucci, Adam Lally, Karin Verspoor, and Eric Nyberg. Unstructured Information Management Architecture. Technical Report March, OASIS, 2009. URL http://docs.oasis-open.org/uima/v1.0/uima-v1.0.pdf.

[52] B. J. Fogg and Hsiang Tseng. *The elements of computer credibility.* ACM Press, New York, New York, USA, 1999. ISBN 0201485591. doi: 10.1145/302979.303001. URL http://portal.acm.org/citation.cfm?doid=302979.303001.

[53] Daniel T. Fokum, Victor S. Frost, Martin Kuehnhausen, Daniel DePardo, Angela N. Oguna, Leon S. Searl, Edward Komp, Matthew Zeets, Daniel D. Deavours, Joseph B. Evans, and Gary J. Minden. An Open-System Transportation Security Sensor Network: Field-Trial Experiences. *IEEE Transactions on Vehicular Technology*, 59(8):3942–3955, October 2010. ISSN 0018-9545. doi: 10.1109/TVT.2010.2060504. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5518450.

[54] Carol J. Fung, Jie Zhang, Issam Aib, and Raouf Boutaba. Dirichlet-Based Trust Management for Effective Collaborative Intrusion Detection Networks. *IEEE Transactions on Network and Service Management*, 8(2):79–91, June 2011. ISSN 1932-4537. doi: 10.1109/TNSM.2011.050311.100028. URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5871350.

[55] Jing Gao, Wei Fan, Yizhou Sun, and Jiawei Han. Heterogeneous source consensus learning via decision propagation and negotiation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*

- *KDD '09*, page 339, New York, New York, USA, 2009. ACM Press. ISBN 9781605584959. doi: 10.1145/1557019.1557061. URL `http://portal.acm.org/citation.cfm?doid=1557019.1557061`.

[56] Roxana Geambasu. *Regaining Control over Cloud and Mobile Data.* PhD thesis, University of Washington, 2011.

[57] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology - CRYPTO 2010*, pages 465–482. Springer, 2010. doi: 10.1007/978-3-642-14623-7_25. URL `http://www.springerlink.com/index/936U72N2W24522TH.pdf`.

[58] Yolanda Gil and Donovan Artz. Towards content trust of web resources. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(4):227–239, December 2007. ISSN 15708268. doi: 10.1016/j.websem.2007.09.005. URL `http://linkinghub.elsevier.com/retrieve/pii/S1570826807000376`.

[59] Barry L. Gingrich and Gary J. Minden. MANDOLIN - A communications management expert system using a reduced form of the Dempster-Shafer uncertainty theory. In *Proceedings of the third international conference on Industrial and engineering applications of artificial intelligence and expert systems - IEA/AIE '90*, pages 76–85, New York, New York, USA, June 1990. ACM Press. ISBN 0897913728. doi: 10.1145/98784.98799. URL `http://portal.acm.org/citation.cfm?id=98784.98799`.

[60] Google. Protocol Buffers. Technical report, 2008. URL `https://developers.google.com/protocol-buffers/`.

[61] Google. Google Wallet, 2011. URL `www.google.com/wallet/`.

[62] Google. DSPL: Dataset Publishing Language. Technical report, 2011. URL `http://code.google.com/apis/publicdata/`.

[63] Kannan Govindan and Prasant Mohapatra. Trust Computations and Trust Dynamics in Mobile Adhoc Networks: A Survey. *IEEE Communications Surveys & Tutorials*, 2011. ISSN 1553-877X. doi: 10.1109/SURV.2011.042711.00083. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5770276`.

[64] C. Guestrin, P. Bodik, and R. Thibaux. Distributed regression: an efficient framework for modeling sensor network data. *Information Processing in Sensor Networks, 2004. IPSN 2004. Third International Symposium on*, pages 1–10, 2004. doi: 10.1109/IPSN.2004.1307317. URL `http://portal.acm.org/citation.cfm?id=984624`.

[65] Manish Gupta and Jiawei Han. Heterogeneous network-based trust analysis. *ACM SIGKDD Explorations Newsletter*, 13(1):54, August 2011. ISSN 19310145. doi: 10.1145/2031331.2031341. URL `http://dl.acm.org/citation.cfm?doid=2031331.2031341`.

[66] Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. *ACM SIGMOD Record*, 14(2):47, June 1984. ISSN 01635808. doi: 10.1145/971697.602266. URL http://portal.acm.org/citation.cfm?doid=971697.602266.

[67] Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. Combating Web Spam with TrustRank. In *Proceedings of the Thirtieth international conference on Very large data bases*, pages 576—-587. VLDB Endowment, 2004.

[68] P.K. Harmer, P.D. Williams, G.H. Gunsch, and G.B. Lamont. An artificial immune system architecture for computer security applications. *IEEE Transactions on Evolutionary Computation*, 6(3):252–280, June 2002. ISSN 1089-778X. doi: 10.1109/TEVC.2002.1011540. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1011540.

[69] David Heckerman. Bayesian Networks for Data Mining. *Data Mining and Knowledge Discovery*, 119:79–119, 1997.

[70] Sungpack Hong, Hassan Chafi, Edic Sedlar, and Kunle Olukotun. Green-Marl. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS '12*, page 349, New York, New York, USA, 2012. ACM Press. ISBN 9781450307598. doi: 10.1145/2150976.2151013. URL http://dl.acm.org/citation.cfm?doid=2150976.2151013.

[71] Xiangdong Hu and Hui Tang. Modeling on Secure Data Aggregation of Wireless Sensor Networks Based on Evaluation of Creditability. In *2010 International Conference on Internet Technology and Applications*, pages 1–4. IEEE, August 2010. ISBN 978-1-4244-5142-5. doi: 10.1109/ITAPP.2010.5566625. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5566625.

[72] Daoli Huang, Wei Liu, and Xue Li. A survey on context awareness. In *2011 International Conference on Computer Science and Service System (CSSS)*, pages 144–147. IEEE, June 2011. ISBN 978-1-4244-9762-1. doi: 10.1109/CSSS.2011.5972040. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5972040.

[73] Mao Lin Huang and Quang Vinh Nguyen. *A Fast Algorithm for Balanced Graph Clustering.* IEEE, July 2007. ISBN 0-7695-2900-3. doi: 10.1109/IV.2007.10. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4271960.

[74] Kai Hwang and Deyi Li. Trusted Cloud Computing with Secure Resources and Data Coloring. *IEEE Internet Computing*, 14(5):14–22, September 2010. ISSN 1089-7801. doi: 10.1109/MIC.2010.86. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5562490.

[75] International Atomic Energy Agency. Fukushima Monitoring Database, 2012. URL https://iec.iaea.org/fmd/.

[76] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 - EuroSys '07*, page 59, New York, New York, USA, 2007. ACM Press. ISBN 9781595936363. doi: 10.1145/1272996.1273005. URL `http://dl.acm.org/citation.cfm?doid=1272996.1273005`.

[77] Paul Jaccard. Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines. *Bulletin de la Société Vaudoise des Sciences Naturelles*, (37):241–272, 1901.

[78] A.K. Jain and P.W. Duin. Statistical pattern recognition: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000. ISSN 01628828. doi: 10.1109/34.824819. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=824819`.

[79] Mohsen Jamali and Martin Ester. Modeling and Comparing the Influence of Neighbors on the Behavior of Users in Social and Similarity Networks. In *2010 IEEE International Conference on Data Mining Workshops*, pages 336–343. IEEE, December 2010. ISBN 978-1-4244-9244-2. doi: 10.1109/ICDMW.2010.97. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5693318`.

[80] Zhang Jizan. A Data Aggregation and Routing Scheme of cobweb model in wireless sensor networks. In *2009 3rd IEEE International Symposium on Microwave, Antenna, Propagation and EMC Technologies for Wireless Communications*, pages 323–327. IEEE, October 2009. ISBN 978-1-4244-4076-4. doi: 10.1109/MAPE.2009.5355947. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5355947`.

[81] Joseph G Johnson and Jerome R Busemeyer. Decision making under risk and uncertainty. *Wiley Interdisciplinary Reviews Cognitive Science*, 1(5):n/a–n/a, 2010. ISSN 19395078. doi: 10.1002/wcs.76. URL `http://doi.wiley.com/10.1002/wcs.76`.

[82] O. Kachirski and R. Guha. Intrusion detection using mobile agents in wireless ad hoc networks. In *Proceedings. IEEE Workshop on Knowledge Media Networking*, pages 153–158. IEEE Comput. Soc. ISBN 0-7695-1778-1. doi: 10.1109/KMN.2002.1115178. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1115178`.

[83] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement Learning : A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996. doi: 10.1613/jair.301.

[84] Beverly K. Kahn, Diane M. Strong, and Richard Y. Wang. Information quality benchmarks. *Communications of the ACM*, 45(4):184, April 2002. ISSN

00010782. doi: 10.1145/505248.506007. URL `http://doi.acm.org/10.1145/505248.506007`.

[85] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The Eigentrust algorithm for reputation management in P2P networks. In *Proceedings of the twelfth international conference on World Wide Web - WWW '03*, page 640, New York, New York, USA, 2003. ACM Press. ISBN 1581136803. doi: 10.1145/775152.775242. URL `http://portal.acm.org/citation.cfm?doid=775152.775242`.

[86] U. Kang, Charalampos E. Tsourakakis, and Christos Faloutsos. PEGASUS: A Peta-Scale Graph Mining System Implementation and Observations. In *2009 Ninth IEEE International Conference on Data Mining*, pages 229–238. IEEE, December 2009. ISBN 978-1-4244-5242-2. doi: 10.1109/ICDM.2009.14. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5360248`.

[87] U Kang, Hanghang Tong, Jimeng Sun, Ching-yung Lin, and Christos Faloutsos. GBASE. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '11*, page 1091, New York, New York, USA, 2011. ACM Press. ISBN 9781450308137. doi: 10.1145/2020408.2020580. URL `http://dl.acm.org/citation.cfm?doid=2020408.2020580`.

[88] Sukumal Kitisin and Clifford Neuman. Reputation-based Trust-Aware Recommender System. In *2006 Securecomm and Workshops*, pages 1–7. IEEE, August 2006. ISBN 1-4244-0422-3. doi: 10.1109/SECCOMW.2006.359555. URL `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4198815`.

[89] Jon M Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, September 1999. ISSN 00045411. doi: 10.1145/324133.324140. URL `http://portal.acm.org/citation.cfm?doid=324133.324140`.

[90] Andreas Krause and Carlos Guestrin. Optimal Value of Information in Graphical Models. *Journal of Artificial Intelligence Research (JAIR)*, 35:557–591, 2009.

[91] Moritz Kroll and Rubino Geiß. Developing Graph Transformations with GrGen.NET. Technical report, Universität Karlsruhe, 2006.

[92] F.R. Kschischang, B.J. Frey, and Hans-andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001. ISSN 00189448. doi: 10.1109/18.910572. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=910572`.

[93] Martin Kuehnhausen. *Service Oriented Architecture for Monitoring Cargo in Motion Along Trusted Corridors.* Masters thesis, University of Kansas, 2009. URL `http://kuscholarworks.ku.edu/dspace/handle/1808/5455`.

[94] Martin Kuehnhausen and Victor S. Frost. Transportation Security SensorNet: a service-oriented architecture for cargo monitoring. *Journal of Systems and Information Technology*, 13(4):369–388, 2011. ISSN 1328-7265. doi:

10.1108/13287261111183979. URL http://www.emeraldinsight.com/10.1108/13287261111183979.

[95] Martin Kuehnhausen and Victor S. Frost. Trusting Smartphone Apps? To install or not to install, that is the question. In *2013 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support*, 2013.

[96] Martin Kuehnhausen, Victor S. Frost, and Gary J. Minden. Framework for assessing the trustworthiness of cloud resources. In *2012 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support*, pages 142–145. IEEE, March 2012. ISBN 978-1-4673-0345-3. doi: 10.1109/CogSIMA.2012.6188367. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6188367.

[97] Raghavendra V. Kulkarni, Anna Forster, and Ganesh Kumar Venayagamoorthy. Computational Intelligence in Wireless Sensor Networks: A Survey. *IEEE Communications Surveys & Tutorials*, 13(1):68–96, 2011. ISSN 1553-877X. doi: 10.1109/SURV.2011.040310.00002. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5473889.

[98] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web - WWW '10*, page 591, New York, New York, USA, 2010. ACM Press. ISBN 9781605587998. doi: 10.1145/1772690.1772751. URL http://portal.acm.org/citation.cfm?doid=1772690.1772751.

[99] Juan Lang, Matt Spear, and S. Felix Wu. Social manipulation of online recommender systems. In *Second international conference on Social informatics*, pages 125–139, Laxenburg, Austria, October 2010. Springer-Verlag. ISBN 3-642-16566-4, 978-3-642-16566-5. URL http://portal.acm.org/citation.cfm?id=1929326.1929336.

[100] S. L. Lauritzen and David. J. Spiegelhalter. Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):157–224, 1988.

[101] H.K. Khac Le, J. Pasternack, H. Ahmadi, M. Gupta, Y. Sun, T. Abdelzaher, J. Han, D. Roth, B. Szymanski, and S. Adali. Apollo: Towards factfinding in participatory sensing. In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pages 129–130, 2011. ISBN 9781450305129. URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5779079.

[102] Hyo-Sang Lim, Yang-Sae Moon, and Elisa Bertino. Provenance-based trustworthiness assessment in sensor networks. In *Proceedings of the Seventh International Workshop on Data Management for Sensor Networks - DMSN '10*, page 2,

New York, New York, USA, September 2010. ACM Press. ISBN 9781450304160. doi: 10.1145/1858158.1858162. URL http://dl.acm.org/citation.cfm?id=1858158.1858162.

[103] Jimmy Lin and Alek Kolcz. Large-scale machine learning at twitter. In *Proceedings of the 2012 international conference on Management of Data - SIGMOD '12*, page 793, New York, New York, USA, 2012. ACM Press. ISBN 9781450312479. doi: 10.1145/2213836.2213958. URL http://dl.acm.org/citation.cfm?doid=2213836.2213958.

[104] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, and Joseph M Hellerstein. GraphLab : A New Framework For Parallel Machine Learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, July 2010.

[105] Yucheng Low, Joseph Gonzalez, Danny Bickson, Carlos Guestrin, and Joseph M Hellerstein. Distributed GraphLab : A Framework for Machine Learning and Data Mining in the Cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012.

[106] Sara C. Madeira and Arlindo L. Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, 2004. ISSN 1545-5963. doi: 10.1109/TCBB.2004.2. URL http://www.ncbi.nlm.nih.gov/pubmed/17048406.

[107] Grzegorz Malewicz, Matthew H Austern, Aart J.C Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel. In *Proceedings of the 2010 international conference on Management of data - SIGMOD '10*, page 135, New York, New York, USA, 2010. ACM Press. ISBN 9781450300322. doi: 10.1145/1807167.1807184. URL http://portal.acm.org/citation.cfm?doid=1807167.1807184.

[108] D.A. Maluf and M.C. Desmarais. A new uncertainty measure for belief networks with applications to optimal evidential inferencing. *IEEE Transactions on Knowledge and Data Engineering*, 13(3):416–425, 2001. ISSN 10414347. doi: 10.1109/69.929899. URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=929899.

[109] Norbert Martínez-Bazan, Victor Muntés-Mulero, Sergio Gómez-Villamor, Jordi Nin, Mario-A. Sánchez-Martínez, and Josep-L. Larriba-Pey. DEX: High-Performance Exploration on Large Graphs for Information Retrieval. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management - CIKM '07*, page 573, New York, New York, USA, November 2007. ACM Press. ISBN 9781595938039. doi: 10.1145/1321440.1321521. URL http://dl.acm.org/citation.cfm?id=1321440.1321521.

[110] M. R. Mokhtar, U. Wajid, and W. Wang. Collaborative Trust in Multi-agent System. In *16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2007)*, pages 30–34. IEEE,

2007. ISBN 0-7695-2879-1. doi: 10.1109/WETICE.2007.4407122. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4407122`.

[111] B. Moore. Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE Transactions on Automatic Control*, 26(1):17–32, February 1981. ISSN 0018-9286. doi: 10.1109/TAC.1981.1102568. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1102568`.

[112] Arjun Mukherjee, Bing Liu, and Natalie Glance. Spotting fake reviewer groups in consumer reviews. In *Proceedings of the 21st international conference on World Wide Web - WWW '12*, page 191, New York, New York, USA, 2012. ACM Press. ISBN 9781450312295. doi: 10.1145/2187836.2187863. URL `http://dl.acm.org/citation.cfm?doid=2187836.2187863`.

[113] Michael Munz, Klaus Dietmayer, and Mirko Mahlisch. Generalized fusion of heterogeneous sensor measurements for multi target tracking. In *Information Fusion (FUSION), 2010 13th Conference on*, pages 1–8, 2010. URL `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5711926`.

[114] David A. Nevell, Simon R. Maskell, Paul R. Horridge, and Hayleigh L. Barnett. Fusion of data from sources with different levels of trust. In *Information Fusion (FUSION), 2010 13th Conference on*, pages 1–7, 2010.

[115] Finn Å rup Nielsen. A new ANEW : Evaluation of a word list for sentiment analysis in microblogs. In *Proceedings of the ESWC2011 Workshop on 'Making Sense of Microposts': Big things come in small packages*, pages 93–98, 2011. URL `http://ceur-ws.org/Vol-718/paper_16.pdf`.

[116] R. Olfati-Saber and R.M. Murray. Consensus Problems in Networks of Agents With Switching Topology and Time-Delays. *IEEE Transactions on Automatic Control*, 49(9):1520–1533, September 2004. ISSN 0018-9286. doi: 10.1109/TAC.2004.834113. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1333204`.

[117] Jack Olson. *Data Quality: The Accuracy Dimension.* December 2002. ISBN 1558608915. URL `http://portal.acm.org/citation.cfm?id=640678`.

[118] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, page 1099, New York, New York, USA, 2008. ACM Press. ISBN 9781605581026. doi: 10.1145/1376616.1376726. URL `http://portal.acm.org/citation.cfm?doid=1376616.1376726`.

[119] D. Opitz and R. Maclin. Popular Ensemble Methods: An Empirical Study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999. doi: 10.1613/jair.614.

[120] Myle Ott, Yejin Choi, Claire Cardie, and Jeffrey T. Hancock. Finding deceptive opinion spam by any stretch of the imagination. In *HLT '11 Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 309–319, June 2011. ISBN 978-1-932432-87-9. URL `http://dl.acm.org/citation.cfm?id=2002472.2002512`.

[121] Sean Owen, Robin Anil, Ted Dunning, and Ellen Friedman. *Mahout in Action*. 2011. ISBN 9781935182689.

[122] Bryan Parno. *Trust Extension as a Mechanism for Secure Code Execution on Commodity Computers*. Phdthesis, Carnegie Mellon University, 2010.

[123] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 35th SIGMOD international conference on Management of data - SIGMOD '09*, page 165, New York, New York, USA, 2009. ACM Press. ISBN 9781605585512. doi: 10.1145/1559845.1559865. URL `http://portal.acm.org/citation.cfm?doid=1559845.1559865`.

[124] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, December 1988. ISBN 0-934613-73-7. URL `http://portal.acm.org/citation.cfm?id=52121`.

[125] Judea Pearl. Causal Diagrams for Empirical Research. *Biometrika*, 82(4):669, December 1995. ISSN 00063444. doi: 10.2307/2337329. URL `http://www.jstor.org/stable/2337329?origin=crossref`.

[126] Judea Pearl. Decision making under uncertainty. *ACM Computing Surveys*, 28 (1):89–92, 1996. ISSN 03600300. doi: 10.1145/234313.234354. URL `http://portal.acm.org/citation.cfm?doid=234313.234354`.

[127] Judea Pearl. Graphical Models for Probabilistic and Causal Reasoning. In Allen B. Tucker, editor, *Computer Science Handbook*, chapter 70, pages 1–18. CRC Press, second edi edition, 2004. ISBN 158488360X.

[128] Judea Pearl. An introduction to causal inference. *The international journal of biostatistics*, 6(2):Article 7, January 2010. ISSN 1557-4679. doi: 10.2202/1557-4679.1203. URL `http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2836213&tool=pmcentrez&rendertype=abstract`.

[129] Siani Pearson. Toward Accountability in the Cloud. *IEEE Internet Computing*, 15(4):64–69, July 2011. ISSN 1089-7801. doi: 10.1109/MIC.2011.98. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5934852`.

[130] Jean-Philippe Pellet and André Elisseeff. Using Markov Blankets for Causal Structure Learning. *Journal of Machine Learning Research*, 9:1295–1342, 2008.

[131] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–38, August 2005. ISSN 0162-8828. doi: 10.1109/TPAMI.2005.159. URL `http://www.ncbi.nlm.nih.gov/pubmed/16119262`.

[132] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molly. Using Probabilistic Generative Models for Ranking Risks of Android Apps. In *ACM Conference on Computer and Communications Security*, October 2012. ISBN 9781450316514.

[133] Leo L. Pipino, Yang W. Lee, and Richard Y. Wang. Data quality assessment. *Communications of the ACM*, 45(4):211, April 2002. ISSN 00010782. doi: 10.1145/505248.506010. URL `http://doi.acm.org/10.1145/505248.506010`.

[134] Robi Polikar. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, 2006. ISSN 1531-636X. doi: 10.1109/MCAS.2006.1688199. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1688199`.

[135] Alan W Powell, Michael J Beckerle, and Stephen M Hanson. Data Format Description Language (DFDL) v1.0 Specification. Technical report, Open Grid Forum, 2011. URL `http://www.ogf.org/documents/GFD.174.pdf`.

[136] Josep M. Pujol, Ramon Sangüesa, and Jordi Delgado. Extracting reputation in multi agent systems by means of social network topology. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems part 1 - AAMAS '02*, page 467, New York, New York, USA, 2002. ACM Press. ISBN 1581134800. doi: 10.1145/544741.544853. URL `http://portal.acm.org/citation.cfm?doid=544741.544853`.

[137] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* 1994. ISBN 0471619779.

[138] A. Quigley and P. Eades. FADE: Graph drawing, clustering, and visual abstraction. *Lecture notes in computer science*, pages 197–210, 2001. URL `http://www.springerlink.com/index/MLBKPLFWX7CMK4HF.pdf`.

[139] Anand Rajaraman, Jure Leskovec, and Jeffrey D. Ullman. *Mining of Massive Datasets.* 1.1 edition, 2012.

[140] Sarvapali D. Ramchurn, Dong Huynh, and Nicholas R. Jennings. Trust in multi-agent systems. *The Knowledge Engineering Review*, 19(01), April 2005. ISSN 0269-8889. doi: 10.1017/S0269888904000116. URL `http://www.journals.cambridge.org/abstract_S0269888904000116`.

[141] Martin Rehák, Michal Pěchouček, and Petr Benda. Fuzzy number approach to trust in coalition environment. In *Proceedings of the fourth international joint*

conference on Autonomous agents and multiagent systems - AAMAS '05, page 1247, New York, New York, USA, July 2005. ACM Press. ISBN 1595930930. doi: 10.1145/1082473.1082716. URL `http://portal.acm.org/citation.cfm?id=1082473.1082716`.

[142] Stephan Reuter and Klaus Dietmayer. Adapting the state uncertainties of tracks to environmental constraints. In *Information Fusion (FUSION), 2010 13th Conference on*, pages 1–7, 2010.

[143] D. Roth and J. Pasternack. Comprehensive Trust Metrics for Information Networks. In *Proc. of the Army Science Conference (ASC)*, 2010. URL `http://cogcomp.cs.illinois.edu/papers/PasternackRo10b.pdf`.

[144] Safecast. Safecast Radiation Measurements, 2012. URL `www.safecast.org`.

[145] K Sallhammar, S.J. Knapskog, and B.E. Helvik. Using Stochastic Game Theory to Compute the Expected Behavior of Attackers. In *2005 Symposium on Applications and the Internet Workshops (SAINT 2005 Workshops)*, pages 102–105. IEEE, 2005. ISBN 0-7695-2263-7. doi: 10.1109/SAINTW.2005.1619988. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1619988`.

[146] Anish Das Sarma. *Managing Uncertain Data*. Phd thesis, Stanford University, 2009. URL `http://ilpubs.stanford.edu:8090/945/`.

[147] Karl Sentz and Scott Ferson. Combination of Evidence in Dempster- Shafer Theory. Technical Report April, Sandia National Laboratories, 2002. URL `www.sandia.gov/epistemic/Reports/SAND2002-0835.pdf`.

[148] Glenn Shafer. *A Mathematical Theory of Evidence*, volume 73. Princeton University Press, 1976. ISBN 0691081751. URL `http://www.glennshafer.com/books/amte.html`.

[149] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(July):379–423, 1948.

[150] Prakash P. Shenoy. A valuation-based language for expert systems. *International Journal of Approximate Reasoning*, 3(5):383–411, September 1989. ISSN 0888613X. doi: 10.1016/0888-613X(89)90009-1. URL `http://linkinghub.elsevier.com/retrieve/pii/0888613X89900091`.

[151] Ilya Shpitser and Judea Pearl. Complete Identification Methods for the Causal Hierarchy. *Journal of Machine Learning Research*, 9:1941–1979, 2008.

[152] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *ACM SIGMOD Record*, 34(3):31, September 2005. ISSN 01635808. doi: 10.1145/1084805.1084812. URL `http://portal.acm.org/citation.cfm?doid=1084805.1084812`.

[153] Abhijit Sinha, Huimin Chen, D.G. Danu, Thia Kirubarajan, and M. Farooq. Estimation and decision fusion: A survey. *Neurocomputing*, 71(13-15):2650–2656, August 2008. ISSN 09252312. doi: 10.1016/j.neucom.2007.06.016. URL `http://linkinghub.elsevier.com/retrieve/pii/S0925231208002282`.

[154] Mark Slee, Aditya Agarwal, and Marc Kwiatkowski. Thrift : Scalable Cross-Language Services Implementation. Technical report, Facebook, April 2007. URL `http://incubator.apache.org/thrift/static/thrift-20070401.pdf`.

[155] Philippe Smets. Analyzing the combination of conflicting belief functions. *Information Fusion*, 8(4):387–412, October 2007. ISSN 15662535. doi: 10.1016/j.inffus.2006.04.003. URL `http://linkinghub.elsevier.com/retrieve/pii/S1566253506000467`.

[156] Matt Spear, Norman Matloff, and S. Felix Wu. KarmaNET: Leveraging trusted social paths to create judicious forwarders. In *2009 First International Conference on Future Information Networks*, pages 218–223. IEEE, October 2009. ISBN 978-1-4244-5158-6. doi: 10.1109/ICFIN.2009.5339559. URL `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5339559`.

[157] David J. Spiegelhalter and Nicola G. Best. Bayesian approaches to multiple sources of evidence and uncertainty in complex cost-effectiveness modelling. *Statistics in medicine*, 22(23):3687–709, December 2003. ISSN 0277-6715. doi: 10.1002/sim.1586. URL `http://www.ncbi.nlm.nih.gov/pubmed/14652869`.

[158] Peter Spirtes. Graphical models, causal inference, and econometric models. *Journal of Economic Methodology*, 12(1):3–34, March 2005. ISSN 1350-178X. doi: 10.1080/1350178042000330887. URL `http://www.informaworld.com/openurl?genre=article&doi=10.1080/1350178042000330887&magic=crossref||D404A21C5BB053405B1A640AFFD44AE3`.

[159] Peter Spirtes. Introduction to Causal Inference. *Journal of Machine Learning Research*, 11, 2010.

[160] Peter Spirtes and Clark Glymour. An Algorithm for Fast Recovery of Sparse Causal Graphs. Technical report, Carnegie Mellon University, 1990.

[161] M. A. Stephens. EDF Statistics for Goodness of Fit and Some Comparisons. *Journal of the American Statistical Association*, 69(347):730, September 1974. ISSN 01621459. doi: 10.2307/2286009. URL `http://www.jstor.org/stable/2286009?origin=crossref`.

[162] S. Subashini and V. Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications*, 34 (1):1–11, January 2011. ISSN 10848045. doi: 10.1016/j.jnca.2010.07.006. URL `http://linkinghub.elsevier.com/retrieve/pii/S1084804510001281`.

[163] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998. ISBN 0262193981.

[164] Gabriele Taentzer. AGG : A Graph Transformation Environment for Modeling and Validation of Software. In *Applications of Graph Transformations with Industrial Relevance, Second International Workshop, AGTIVE*, pages 446–453, 2003. doi: 10.1007/978-3-540-25959-6_35.

[165] Adelina Tang. Dynamic Bayesian approach to forecasting. In *2010 Sixth International Conference on Natural Computation*, number Icnc, pages 3933–3937. IEEE, August 2010. ISBN 978-1-4244-5958-2. doi: 10.1109/ICNC.2010.5584772. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5584772`.

[166] Steven J. Templeton. Security Aspects of Cyber-Physical Device Safety in Assistive Environments. In *Proceedings of the 4th International Conference on Pervasive Technologies Related to Assisted Environments PETRA*, 2011. ISBN 9781450307727. URL `http://hivemind.cs.ucdavis.edu/_docs_/Templeton-2011-SecurityAspectsofCyber-PhysicalDeviceSafetyinAssistiveEnvironments-pdf`.

[167] The Apache Software Foundation. Avro: A data serialization system, . URL `http://avro.apache.org`.

[168] The Apache Software Foundation. Giraph : Large-scale graph processing on Hadoop, . URL `http://giraph.apache.org`.

[169] The Apache Software Foundation. Hadoop, . URL `http://hadoop.apache.org`.

[170] The Apache Software Foundation. Hive: A data warehouse infrastructure that provides data summarization and ad hoc querying, . URL `http://hive.apache.org`.

[171] The Apache Software Foundation. Mahout: A Scalable machine learning and data mining library, . URL `http://mahout.apache.org`.

[172] The Apache Software Foundation. Pig: A high-level data-flow language and execution framework for parallel computation, . URL `http://pig.apache.org/`.

[173] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu, and Raghotham Murthy. Hive - a petabyte scale data warehouse using Hadoop. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 996–1005. Ieee, 2010. ISBN 978-1-4244-5445-7. doi: 10.1109/ICDE.2010.5447738. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5447738`.

[174] Hanghang Tong, Yasushi Sakurai, Tina Eliassi-Rad, and Christos Faloutsos. Fast mining of complex time-stamped events. In *Proceeding of the 17th ACM conference on Information and knowledge mining - CIKM '08*, page 759, New York, New York, USA, 2008. ACM Press. ISBN 9781595939913. doi: 10.1145/1458082.1458184. URL `http://portal.acm.org/citation.cfm?doid=1458082.1458184`.

[175] Stephanie Watts, G. Shankaranarayanan, and Adir Even. Data quality assessment in context: A cognitive perspective. *Decision Support Systems*, 48(1):202–211, December 2009. ISSN 01679236. doi: 10.1016/j.dss.2009.07.012. URL `http://portal.acm.org/citation.cfm?id=1651932.1652144`.

[176] Yi Wei and M Brian Blake. Service-Oriented Computing and Cloud Computing: Challenges and Opportunities. *IEEE Internet Computing*, 14(6):72–75, November 2010. ISSN 1089-7801. doi: 10.1109/MIC.2010.147. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5617062`.

[177] B. L. Welch. The generalization of "Student's" problem when several different population variances are involved. *Biometrika*, 34(1-2):28–35, 1947. ISSN 0006-3444. doi: 10.1093/biomet/34.1-2.28. URL `http://biomet.oxfordjournals.org/cgi/doi/10.1093/biomet/34.1-2.28`.

[178] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media / Yahoo Press, 2nd editio edition, 2010. ISBN 9781449389734.

[179] J.L. Williams and R.A. Lau. Data association by loopy belief propagation. In *Information Fusion (FUSION), 2010 13th Conference on*, pages 1–8, 2010.

[180] Miao Xie, Song Han, Biming Tian, and Sazia Parvin. Anomaly detection in wireless sensor networks: A survey. *Journal of Network and Computer Applications*, March 2011. ISSN 10848045. doi: 10.1016/j.jnca.2011.03.004. URL `http://linkinghub.elsevier.com/retrieve/pii/S1084804511000580`.

[181] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–78, May 2005. ISSN 1045-9227. doi: 10.1109/TNN.2005.845141. URL `http://www.ncbi.nlm.nih.gov/pubmed/15940994`.

[182] Ronald R. Yager. On the fusion of imprecise uncertainty measures using belief structures. *Information Sciences*, 181(15):3199–3209, August 2011. ISSN 00200255. doi: 10.1016/j.ins.2011.02.010. URL `http://linkinghub.elsevier.com/retrieve/pii/S0020025511000922`.

[183] Xiaoxin Yin, Wenzhao Tan, Xiao Li, and Yi-Chin Tu. Automatic extraction of clickable structured web contents for name entity queries. In *Proceedings of the 19th international conference on World wide web - WWW '10*, page 991, New York, New York, USA, 2010. ACM Press. ISBN 9781605587998. doi: 10.1145/1772690.1772791. URL `http://portal.acm.org/citation.cfm?doid=1772690.1772791`.

[184] Meng Yu, Wanyu Zang, and Barbara Reagor. Decentralized Trust Management based on the Reputation of Information Sources. In *2007 IEEE International Conference on Networking, Sensing and Control*, number April, pages 212–217. IEEE, 2007. ISBN 1-4244-1075-4. doi: 10.1109/ICNSC.2007.372779. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4238992`.

[185] P.S. Yu. Truth Discovery with Multiple Conflicting Information Providers on the Web. *IEEE Transactions on Knowledge and Data Engineering*, 20(6):796–808, June 2008. ISSN 1041-4347. doi: 10.1109/TKDE.2007.190745. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4415269`.

[186] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. In *OSDI'08 Proceedings of the 8th USENIX conference on Operating systems design and implementation*, pages 1–14, 2008. URL `http://www.usenix.org/event/osdi08/tech/full_papers/yu_y/yu_y_html/`.

[187] Runfang Zhou and Kai Hwang. PowerTrust: A Robust and Scalable Reputation System for Trusted Peer-to-Peer Computing. *IEEE Transactions on Parallel and Distributed Systems*, 18(4):460–473, April 2007. ISSN 1045-9219. doi: 10.1109/TPDS.2007.1021. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4118688`.