Towards Building A Reputation-based Microservices Trust Model Using Similarity Domains

Zhongyi Lu[1][a], Declan T. Delaney[2][b], Tong Li[3][c] and David Lillis[1][d]

[1]*School of Computer Science, University College Dublin, Dublin 4, Ireland*

[2]*School of Electrical and Electronic Engineering, University College Dublin, Dublin 4, Ireland*

[3]*Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China*

zhongyi.lu@ucdconnect.ie, declan.delaney@ucd.ie, litong@bjut.edu.cn, david.lillis@ucd.ie

Abstract: Microservices have been seen as a solution to open systems, within which microservices can behave arbitrarily. This requires the system to have strong trust management. However, existing microservices trust models cannot fully support open systems. In this paper, we propose a reputation-based trust model designed for open microservices that groups similar microservices within the same "similarity domain" and includes a trust bootstrapping process and a comprehensive trust computation method. Our proposal introduces a new concept called "trust balancing" to assure that all microservices can fairly be incorporated into the operation of the system. The design of the evaluation plan is also introduced to demonstrate the suitability of the proposed model to open microservice systems.

1   Introduction

The microservices architecture (MSA) is the state of the art of large-scale software development (Fritzsch et al., 2019). Typically, microservices within a system are designed to trust their peers (Mateus-Coelho et al., 2021). However, an untrustworthy microservice can cause system breakdowns (Sun et al., 2015). Therefore, trust models are essential to protect open microservice systems from vulnerabilities (Pourghebleh et al., 2019). A number of approaches to microservices trust models have been adopted, including those based on "zero-trust" (Hong et al., 2023), regarding trust as a reflection of its reputation among other microservices (Ruan et al., 2021), decomposing the feedback that the truster gives to an entire solution (Adewuyi et al., 2022), and managing trust using control models (Venčkauskas et al., 2023).

MSA is seen as the solution for building open systems (Fritzsch et al., 2019), which face unpredictable challenges in open environments (Baresi et al., 2006). Our previous survey (Lu et al., 2023) showed that existing microservices models cannot fully support open systems and may cause problems. Thus, a specific trust model for open microservice systems is needed.

In this paper, we propose a reputation-based trust model for open microservice systems. Trust is "a belief of a truster in a trustee that the trustee will provide or accomplish the services that it says it will provide and meet the expectations of the truster within a specific context for a specific period of time." (Lu et al., 2023) The model groups similar microservices into "similarity domains" to manage trust. Microservices' trust scores are based on feedback from recent interactions and various Quality of Service (QoS) metrics. We also outline an evaluation plan to demonstrate the model's suitability for open microservice systems.

2   Related Work

This section briefly reviews existing trust models for MSAs, focusing on the requirements of open systems and the integration of newly-added microservices. Due to the similarities between MSAs and Web services architectures, we also examine relevant literature from the latter, where extensive research has been conducted.

[a] https://orcid.org/0000-0002-6428-8782

[b] https://orcid.org/0000-0001-7028-3307

[c] https://orcid.org/0000-0002-8881-0037

[d] https://orcid.org/0000-0002-5702-4463

2.1 Microservices Trust Models

Microservice systems require trust models to monitor connections and establish trust between individual microservices, thereby mitigating trust-related attacks (Abdelghani et al., 2016) and system damage (Dragoni et al., 2017). While several publications have addressed this issue, the literature on trust in microservices is relatively limited. Existing microservice trust models fall into four categories: Zero-Trust-based, socio-based, composition-based, and control-based (Lu et al., 2023).

- *Zero-trust-based* models trust no microservices. Authentication and authorisation are required before any interaction (Gilman and Barth, 2017).

- *Socio-based* trust models treat the system as a society of microservices. Trust scores of microservices reflect their reputation within society.

- *Composition-based* trust models compose a group of microservices as one solution. The trusters will rate the solution, which will then be decomposed into trust scores for individual microservices.

- *Control-based* trust models manage trust using the same logic as control models, which can control the traffic within the system.

2.2 Trust Models For Open Microservice Systems

Within open systems, microservices from diverse origins may join or leave the system arbitrarily, posing additional challenges for trust management.

Previously, we proposed a set of qualities for trust models for open microservice systems (Lu et al., 2023), including: i) a comprehensive trust bootstrapping process; ii) resilience to missing microservices; iii) defence against trust manipulation; iv) separate calculation of trust value for each microservice; v) tolerance for occasional failures; vi) reflection of recent trustworthiness; vii) equitable distribution of service calls to all microservices; and viii) integration of QoS metrics in trust calculation.

2.3 Web services Trust Management

The MSA extends the Service-Oriented Architecture (SOA), which encapsulates functionalities as network-available services for integration into business solutions (Laskey and Laskey, 2009). Despite MSA being a superior enterprise solution compared to Web services-based SOA (Raj and Ravichandra, 2018), the latter has a longer history, making it valuable to learn from previous Web service trust models.

2.3.1 Web services Trust Model

Wang and Vassileva (Wang and Vassileva, 2007) introduced a three-level hierarchy for Web services trust model classification, which includes:

- **Centralised/Decentralised**: In a centralised system, a central node manages reputations. For example, TRUSS (Tang et al., 2017), which combines objective and subjective trust assessments as the trust evaluation middleware.

  In a decentralised system, there is no central node. (Nguyen et al., 2010) proposed a Bayesian network trust and reputation model for each consumer to build trust with Web services.

- **Agent(Person)/Resource**: Agent systems model the reputation of agents representing people. RATEWeb (Malik and Bouguettaya, 2009a) assesses trust with the reputation of service vendors.

  Resource systems focus on modelling the reputations of resources. Galizia et al. (Galizia et al., 2007) managed trust using Web service Trust Ontology, which embeds the trust-based selection of Web services into a classification problem.

- **Global/Personalised**: Global reputation systems are based on public opinions. Caballero et al. proposed a model to manage reputation using the Web service Modelling Ontology in a P2P environment (Caballero et al., 2006).

  Personalised systems build reputation on subjective opinions. In the trust model proposed by (Liu et al., 2014), the Web service network is regarded as a small-world, where any two services can be connected through at most six services.

2.3.2 Trust Bootstrapping

Trust evaluation relies on past information about instances. Without such history, newcomers have no chance to compete with existing Web services for selection by trusters (Nguyen et al., 2012). The cold start problem then arose as there was typically no way to judge initial trust (Malik and Bouguettaya, 2009b). The initial trust should not be too high or too low, so existing services or the new service would be disadvantaged (Sensoy et al., 2013). Thus, it is vital to strike a balance when assigning initial trust to any incoming component. Previous measures can be divided into three categories: default value, punishment-based, and adaptive (Zhang and Li, 2022).

Default value approaches set the initial trust to a constant default value (Wang and Vassileva, 2007; Malik and Bouguettaya, 2009b; Jøsang, 2016). How-

ever, inappropriate default values may disadvantage either existing services or the new service.

Punishment-based approaches assume newcomers are not trustworthy and set their initial trust value to a low value (Friedman et al., 2007; Wu et al., 2015; Zacharia et al., 2000; Yahyaoui and Zhioua, 2013). However, a low initial trust score increases the risk of never being invoked.

Adaptive approaches determine the initial trust value of newcomer services based on their characteristics or similarity with existing services. Three mechanisms were proposed in (Nguyen et al., 2012): inheritance, referral, and guarantee mechanisms. The inheritance mechanism sets trust based on the vendor's trust degree if they have uploaded other services. The referral mechanism assesses the newcomer's behaviour in other systems. The guarantee mechanism assigns high trust to services promising quality and reimbursement for failures. In (Wu et al., 2015), a reputation bootstrapping method was introduced, which used artificial neural networks to learn correlations between service features and performance. Preset communities were created in (Malik and Bouguettaya, 2009b), which store similar services. The service providers decide which community they believe their services belong to. In (Sensoy et al., 2013), ontology graphs were used to match newcomers with existing patterns for trust bootstrapping.

2.4 Computing the Trust of Microservices And Web services

While both microservices and Web services are related to SOA, their trust computation methods differ.

While Web services trust models are person-centred or agent-centred, microservices trust models are resource-centred (Wang and Vassileva, 2007). Web services aim to build trust and reputation for vendors selling services over the World Wide Web (Kreger, 2003), whereas microservices focus on eliminating vendor lock-in (Kecskemeti et al., 2016), suggesting less emphasis on building trust for vendors.

MSA favours decentralised governance compared to Web services and requires fine-grained services, unlike Web services, which can encapsulate multiple responsibilities in one service. This complexity poses challenges for trust modelling, demanding more robust trust models for MSA..

Existing Web services trust models struggle with malicious services in composite architectures (Wahab et al., 2015). MSA, being more vulnerable than Web services, requires stronger defences against trust attacks (Baresi and Garriga, 2019).

Therefore, existing Web service trust models are not directly applicable to microservices. Separate trust models are needed for both microservice systems and open microservice systems, albeit some aspects of microservices trust models may be informed by pre-existing Web services models.

3 Trust Model Overview

We propose a trust model that is suitable for an open microservice system.

In a MSA, "domains" are organisational units that form the systems (Steinegger et al., 2017). Allocating microservices into different domains based on some criteria (e.g., the context of the system (Rademacher et al., 2018)) is a common way to manage microservices. This has inspired us to group functionally similar microservices into the same domain. We identified the similarity metric that we proposed in (Lu et al., 2024). Unlike the similarity mentioned in (Malik and Bouguettaya, 2009b), this similarity metric is computed fully by the system, which avoids the risk that may be brought about by dishonest vendors being permitted to choose a domain for their microservices.

There are five architectural components in the model: Domain Manager, Similarity Evaluator, Similarity Domains, Trust Managers, and Monitors.
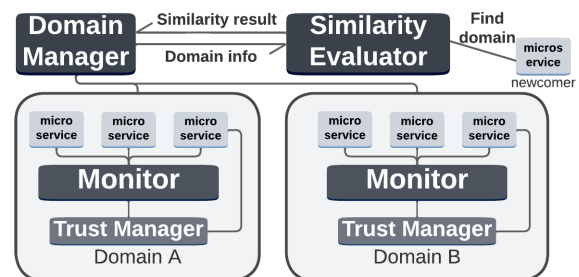

Figure 1: An example structure of the model.

Similar microservices are allocated to the same **Similarity Domain**, which are all managed by one **Domain Manager**. The Domain Manager is a microservice that maintains domain lists, receives similarity evaluation results, and allocates newcomer microservices to domains. The **Similarity Evaluator** is a microservice that encapsulates a similarity metric, which evaluates the similarity between a newcomer microservice and existing services, finds it a matching domain, and sends it to the Domain Manager.

Within each Similarity Domain, there is a **Trust Manager**, which is a microservice that manages trust scores of the services and returns trustee services to trusters based on a probabilistic selection mechanism. A key feature of the Trust Manager is that it main-

tains a "trust record" for each microservice to compute trust. This records data relating to the most recent interactions that the microservice has engaged in. A **Monitor** also monitors the services within the domain to support a penalty scheme that deters trust manipulation and the QoS performance of services.

In order to support large-scale systems, data will be distributed. Trust Managers will broadcast their data to other Trust Managers to avoid any runtime failures. There will also be backups for the Domain Manager and Similarity Evaluators to avoid failures.

4 Trust management

Trust management comprises three major processes: trust bootstrapping for newcomer microservices; trust value computation; and trust balancing with a probabilistic selection strategy.

4.1 Trust Bootstrapping

When a new microservice joins the system, the Similarity Evaluator will use the similarity metric to find a suitable domain. The result will then be passed to the Domain Manager and the Trust Manager of the matching domain.

The Trust Manager of the domain that the newcomer microservice belongs to has responsibility for computing a bootstrapping trust score. The initial trust score of the new microservice i is the mean value of the trust score of all the other microservices in the domain. However, if the newcomer service is the only microservice in the domain, the initial trust value will be set to 0.5. Formally, the initial trust score for microservice i (denoted as T_i) is given by

$$T_i = \begin{cases} \frac{\sum_{j \in D_i \setminus \{i\}} T_j}{|D_i| - 1} & : |D_i| > 1 \\ 0.5 & : |D_i| = 1 \end{cases} \quad (1)$$

where D_i is the similarity domain that i is associated with (i.e., microservices within that domain).

4.2 Trust Computation

Later, once the microservice has been added to the system and begun to satisfy requests, its overall trust score is computed based on its reputation score R_i, its QoS score Q_i, and possibly a penalty if any trust manipulation has been observed by the Monitor.

The reputation score of a microservice is the aggregation of its previous trusters' opinions. It is computed based on the ratings r that the trusters give to the trustees after each interaction. A rating is an integer

between 0 and n, reflecting the opinion of the truster. The higher r is, the more trustworthy the truster thinks the trustee is during their interaction.

Equation 2 demonstrates a rating scale for $n = 5$.

$$opinion = \begin{cases} \text{cannot perform,} & \text{if } r_{i,t} = 0 \\ \text{extremely poor,} & \text{if } r_{i,t} = 1 \\ \text{bad,} & \text{if } r_{i,t} = 2 \\ \text{average,} & \text{if } r_{i,t} = 3 \\ \text{good,} & \text{if } r_{i,t} = 4 \\ \text{excellent} & \text{if } r_{i,t} = 5 \end{cases} \quad (2)$$

The reputation score that trustee microservice i receives for the t-th interaction that is stored in its interaction record is normalised to lie between 0 and 1, as follows:

$$R_{i,t} = \frac{r_{i,t}}{n} \quad (3)$$

The QoS score is used to reflect how well microservices comply with the QoS metrics. QoS is commonly used in Web service trust models as a factor to evaluate trust. Combining the definition of QoS given by W3C (Connolly, 1997) and the selection of QoS metrics that are often used in previous trust models, the QoS metrics used in the proposed model are: i) **Availability** that shows if a microservice instance can be retrieved and reached via the network; ii) **Latency**, which is the time interval between the creation of a request and the completion of the request; iii) **Cost**, which is the resource usage during service execution time, e.g., CPU utilisation, storage, etc.; iv) **Throughput**, which is the number of requests that the microservice instance has served in a certain period of time; and v) **Reliability**, which is the ratio of successful responses to all responses that the service instance returns. Responses that do not match the expectations based on the API file are unsuccessful. Reliability reflects whether a microservice can provide services that are compliant with its API description. Availability, latency and throughput quantify whether a microservice can be accessed by trusters in a certain period of time. Cost is also a major factor that will affect trusters' views of the microservice.

Equation 4 explains the calculation of the QoS score, which is computed by the Monitor within each domain. In order to avoid the overhead of constant recalculations, several QoS metrics are calculated periodically based on microservice interactions that have occurred within a specified time period (which may be in the order of minutes). In the following formulae, x denotes the duration of the most recent time period for which such a recalculation has occurred. Equations 5 to 10 explain how each metric score is calculated.

The overall QoS for the t-th interaction in i's trust record, $Q_{i,t}$ is defined as a weighted sum of several component metrics, as follows:

$$Q_{i,t} = \alpha \times A_{i,t} + \beta \times L_{i,t} + \gamma \times \Theta_{i,t} + \delta \times \rho_{i,t} + \varepsilon \times C_{i,t}$$

(4)

where α through ε are the respective positive weights associated with the component metrics defined below. Specific weights can be chosen to suit particular system requirements, provided that the sum of the weights equals 1.

The *availability* score associated with microservice i for the t-th interaction in its trust record, $A_{i,t}$ is given by

$$A_{i,t} = \frac{a_{i,x}}{min(\tau_i, x)}$$

(5)

where $a_{i,x}$ is the time that i has been available during the calculation period, and τ_i is the lifetime duration of i (i.e., the elapsed time since i joined the similarity domain), to adjust for new microservices that were not in operation at the start of the calculation period.

The *latency* score for the t-th interaction of i, $L_{i,t}$ is given by

$$L_{i,t} = 1 - \frac{l_{i,t} - l_{min}}{l_{max} - l_{min}}$$

(6)

where $l_{i,t}$ is the latency of the t-th interaction of i. The score is normalised. l_{max} and $lmin$ are the maximum and minimum latencies of any interaction in the record of any microservice in the domain.

The *throughput* score for i at its t-th iteration, $\Theta_{i,t}$, is given by

$$\Theta_{i,t} = \frac{req_{i,x}}{min(\tau_i, x)}$$

(7)

where $req_{i,x}$ is the number of requests processed by i during the time interval x.

The *reliability* score associated with the t-th interaction of i, $\rho_{i,t}$ is given by

$$\rho_{i,t} = \frac{s_{i,x}}{s_{i,x} + u_{i,x}}$$

(8)

where $s_{i,x}$ is the number of requests that i successfully satisfied during time period x, and $u_{i,x}$ is the number of requests that i did not successfully satisfy during that time period.

The final component of the QoS computation is the cost associated with i satisfying request t (denoted as $cost_{i,t}$). This is defined flexibly in the model to allow for multiple measures of cost to be dynamically chosen according to the specific use case of a system. Each such measure is described as a "usage perspective". Common usage perspectives would include CPU or memory usage, but consumption of other scarce or expensive resources may be a critical feature of certain systems. It is given by:

$$cost_{i,t} = \sum_{j=0}^{k} \kappa_j \times U_{i,j,t}, \qquad for \sum_{j=0}^{k} \kappa_j = 1, 0 \leq U_{i,j,t} \leq 1$$

(9)

where k is the number of usage perspectives to be included and $U_{i,j,t}$ is the cost of microservice i engaging in interaction t, as measured by usage perspective j. This cost is defined in the model as a positive value that is less than 1, with specific methods for measuring this dependent on the nature of the usage perspective in question (e.g., memory usage cost may be quantified in comparison to the memory usage of other microservices within the same domain). Each usage perspective has a weight, κ_j associated with it, according to system requirements. The sum of all such weights must equal 1.

This cost score is normalised to give the *normalised cost score* for the t-th interaction of i, denoted as $C_{i,t}$, which is computed by

$$C_{i,t} = \frac{cost_{i,t} - cost_{min}}{cost_{max} - cost_{min}}$$

(10)

where $cost_{max}$ and $cost_{min}$ are the maximum and minimum costs of any interaction in the record of any microservice in the domain.

4.3 Trust Manipulation and Trust Balancing

In an open system, it must be assumed that microservices may act in bad faith, particularly given the existence of a notion of trust. A penalty scheme is incorporated into this model to punish trust manipulation (i.e., malicious attempts to illegitimately gain a high trust score). Each Monitor seeks to detect evidence of trust manipulation. The following possible signs of trust manipulation are anticipated:

- A service instance POSTed a review of another service instance but had never interacted with the service instance being rated.

- A service instance keeps POSTing negative reviews to other service instances.

- A service instance POSTed a review of a service instance that is contrary to the rest of the trust record of the trustee.

- A service instance only POSTs reviews to one specific service instance when there are other trustworthy services within the domain that the reviewed service is in.

- A trusted service instance starts to perform untrustworthily (e.g., provides services that are different from its description).

With the behaviour of each microservice being recorded, its associated Monitor will be able to analyse its behaviour and detect evidence of trust manipulation. The trust of any microservice i will be 0 if manipulative behaviour is detected. This will be achieved with Prometheus, which is a third-party tool for microservice activity monitoring. We combine the activities of microservices with the review record of microservices to detect manipulation behaviours.

For a microservice i, with other trust scores being computed, the trust score that microservice i received for the t-th interaction that is stored in its trust record, which is their weighted sum, can be calculated as:

$$T_{i,t} = \zeta \times R_{i,t} + \mu \times Q_{i,t}, \qquad for \quad \zeta + \mu = 1 \quad (11)$$

However, if the interaction shows evidence of trust manipulation, then $T_{i,t}$ should be 0.

A record discarding mechanism is adopted so that the trust score of microservices reflects their more recent trustworthiness to handle open environments. The Trust Manager for each domain stores the N most recent trust record of a microservice, where N is either a fixed minimum threshold or all trust records that have been created since the end of the last QoS calculation period, whichever is greater. This is to ensure that all interactions required to calculate the QoS metrics for the next calculation period are present.

The system can then calculate the overall trust score of i with its trust records. The importance of each record decays over time. Recent trust records should be of significantly higher importance to the trust score compared to past records. The formula is:

$$T_i = \sum_{t=0}^{N-1} \frac{2(t+1) \times T_{i,t}}{N(N+1)} \quad (12)$$

4.3.1 Trust Balancing

Trust balancing is a novel concept introduced in this model. It is derived from the concept of load balancing. The aim is to distribute the chances of microservices being recommended to trusters using a probabilistic strategy. The probability of a microservice n being recommended is related to its trust score. Equation 13 explains how P_i is calculated with T_i:

$$P_i = \frac{e^{\omega \times T_i}}{\sum_{j \in D_i} e^{\omega \times T_j}} \quad (13)$$

where ω is a scaling factor that can give a disproportional higher probability of being recommended to more trustworthy microservices while ensuring all microservices with a trust score higher than 0 a chance of being recommended, so their trust value can converge from the initial trust value and reflect their actual trustworthiness. A byproduct is the avoidance of overload: as all the microservices are involved in the operation, the possibility of a microservice being called too many times can be reduced.

5 Evaluation Plan

This section outlines the evaluation design that will be conducted on the proposed model.

Firstly, we have built a microservice system based on the real-world OpenAPI dataset introduced in (Lu et al., 2024). To evaluate the accuracy of trust, we will manually set the trustworthiness of microservices, adjust the code accordingly to reflect this trustworthiness, and then see how well the computed trust score matches the designed trustworthiness. It is worth noting that this does not mean computed trust scores need to have exactly the same value as the designated trust score. As long as the trust score is within an acceptable margin of error and, more importantly, the ranking of microservices' trust scores reflects their actual trustworthiness, the model will be considered effective. We will upload microservices to the system gradually while removing others in order to see how it can support systems of different scales.

To minimise the effect of other objective factors on QoS, we deploy all microservices on the same server but on different ports. It is not realistic in practice that a MSA-based system is deployed on merely one single machine. However, as the main objective of the evaluation is the effectiveness of the trust computation, this kind of implementation will not largely affect the evaluation. In terms of single-machine deployment, as it does not need to interact with other servers, the latency could be shorter, and the cost and throughput could be lower as the host is only working on one system. All the experiments will be conducted on a MacBook Pro with an Apple M1 Pro chip, 32GB of memory, and 994.66GB of storage. The operating system is macOS Ventura Version 13.5.2 (22G91). To monitor the performance of the model, we use Prometheus to monitor the activity of each port.

The evaluation is designed to demonstrate whether the proposed model can fulfil the eight qualities for open microservice trust models proposed in (Lu et al., 2023). The evaluation will be carried out by comparing the performance of a microservice system with a basic trust model and a microservice system with the proposed model on all aspects of the qualities, namely:

Trust Bootstrapping: Trust bootstrapping establishes initial trust in a newcomer to accommodate it in the system. The time interval between a new microservice joining and it being assigned a trust value

and the time interval between a new microservice joining and it being called by a trustee is recorded.

Resilience to missing microservices: Microservice systems should not be affected if a commonly used microservice is missing. We will remove some commonly used microservices to simulate this. The HTTP request handling time will be recorded to see how long the system finds a substitute and recovers.

Resistance to trust manipulation: Trust-manipulative behaviours should not affect the trust score of microservices. We will manually adjust the code or API file of microservices to perform the manipulative behaviours mentioned in Section 4.3 and monitor whether trust scores are affected.

Individual trust score computation: As Section 4.2 demonstrates, the trust score of each microservice is computed independently. This may not require any further experimentation.

Failure tolerance: The system should still give microservices chances to perform after single failures. We will monitor the system and see how often microservices are called after single failures.

Reflect recent trustworthiness: The trust score should represent recent trustworthiness. We will manually change the code of certain microservices to change their trustworthiness and monitor how many interactions it takes to let the trust score change.

Distribute chances: All microservices should have the opportunity to be called. We will monitor how many microservices are being called in a certain time period to see if all microservices that are not extremely untrustworthy are being recommended.

Incorporate objective metrics: The proposed model includes an objective QoS score. Thus, it incorporates objective metrics to compute trust.

6 Conclusion

Trust management is important for systems. Although there is previous research on trust models for microservices and Web services, they cannot be fully adopted by open microservice systems. This paper proposes a reputation-based trust model designed to support open microservice systems. It has three major novelties: a comprehensive trust bootstrapping process for newcomer microservices using similarity; an anti-manipulation trust computation process that computes trust using recent ratings and QoS; and a trust balancing scheme that distributes microservices' chances of being recommended to the trusters using a probabilistic method based on their trust score.

We also proposed the evaluation plan, which will later be used to test the performance of the proposed model in a simulated open microservice environment.

REFERENCES

Abdelghani, W., Zayani, C. A., Amous, I., and Sèdes, F. (2016). Trust Management in Social Internet of Things: A Survey. In Dwivedi, Y. K., Mäntymäki, M., Ravishankar, M., Janssen, M., Clement, M., Slade, E. L., Rana, N. P., Al-Sharhan, S., and Simintiras, A. C., editors, *Social Media: The Good, the Bad, and the Ugly*, pages 430–441, Cham. Springer International Publishing.

Adewuyi, A. A., Cheng, H., Shi, Q., Cao, J., Wang, X., and Zhou, B. (2022). SC-TRUST: A Dynamic Model for Trustworthy Service Composition in the Internet of Things. *IEEE Internet of Things Journal*, 9(5):3298–3312.

Baresi, L., Di Nitto, E., and Ghezzi, C. (2006). Toward open-world software: Issues and challenges. *Computer*, 39(10):36–43.

Baresi, L. and Garriga, M. (2019). *Microservices: The Evolution and Extinction of Web Services?*, page 3–28.

Caballero, A., Botia, J. A., and Gomez-Skarmeta, A. F. (2006). A new model for trust and reputation management with an ontology based approach for similarity between tasks. In Fischer, K., Timm, I. J., André, E., and Zhong, N., editors, *Multiagent System Technologies*, page 172–183, Berlin, Heidelberg. Springer.

Connolly, D. (1997). Quality of Service. `https://www.w3.org/Architecture/qos.html`. Accessed: 2024-04-25.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L. (2017). *Microservices: Yesterday, Today, and Tomorrow*, pages 195–216. Springer International Publishing, Cham.

Friedman, E., Resnick, P., and Sami, R. (2007). Manipulation-resistant reputation systems. *Algorithmic Game Theory*, 677.

Fritzsch, J., Bogner, J., Wagner, S., and Zimmermann, A. (2019). Microservices Migration in Industry: Intentions, Strategies, and Challenges. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 481–490.

Galizia, S., Gugliotta, A., and Domingue, J. (2007). A Trust Based Methodology for Web Service Selection. In *International Conference on Semantic Computing (ICSC 2007)*, page 193–200.

Gilman, E. and Barth, D. (2017). *Zero trust networks*. O'Reilly Media, Incorporated.

Hong, S., Xu, L., Huang, J., Li, H., Hu, H., and Gu, G. (2023). SysFlow: Toward a Programmable Zero Trust Framework for System Security. *IEEE Transactions on Information Forensics and Security*, 18:2794–2809.

Jøsang, A. (2016). *Subjective logic*, volume 3. Springer.

Kecskemeti, G., Marosi, A. C., and Kertesz, A. (2016). The ENTICE approach to decompose monolithic services into microservices. In *2016 International Conference on High Performance Computing & Simulation (HPCS)*, page 591–596.

Kreger, H. (2003). Fulfilling the Web services promise. *Communications of the ACM*, 46(6):29.

Laskey, K. B. and Laskey, K. (2009). Service oriented architecture. *WIREs Computational Statistics*, 1(1):101–105.

Liu, F., Wang, L., Gao, L., Li, H., Zhao, H., and Men, S. K. (2014). A Web Service trust evaluation model based on small-world networks. *Knowledge-Based Systems*, 57:161–167.

Lu, Z., Delaney, D. T., and Lillis, D. (2023). A Survey on Microservices Trust Models for Open Systems. *IEEE Access*, 11:28840–28855.

Lu, Z., Delaney, D. T., and Lillis, D. (2024). Comparing the Similarity of OpenAPI-Based Microservices. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, SAC '24, page 1201–1208, New York, NY, USA. Association for Computing Machinery.

Malik, Z. and Bouguettaya, A. (2009a). RATEWeb: Reputation Assessment for Trust Establishment among Web services. *The VLDB Journal*, 18(4):885–911.

Malik, Z. and Bouguettaya, A. (2009b). Reputation Bootstrapping for Trust Establishment among Web Services. *IEEE Internet Computing*, 13(1):40–47.

Mateus-Coelho, N., Cruz-Cunha, M., and Ferreira, L. G. (2021). Security in microservices architectures. *Procedia Computer Science*, 181:1225–1236.

Nguyen, H. T., Yang, J., and Zhao, W. (2012). Bootstrapping Trust and Reputation for Web Services. In *2012 IEEE 14th International Conference on Commerce and Enterprise Computing*, page 41–48.

Nguyen, H. T., Zhao, W., and Yang, J. (2010). A Trust and Reputation Model Based on Bayesian Network for Web Services. In *2010 IEEE International Conference on Web Services*, page 251–258.

Pourghebleh, B., Wakil, K., and Navimipour, N. J. (2019). A Comprehensive Study on the Trust Management Techniques in the Internet of Things. *IEEE Internet of Things Journal*, 6(6):9326–9337.

Rademacher, F., Sorgalla, J., and Sachweh, S. (2018). Challenges of Domain-Driven Microservice Design: A Model-Driven Perspective. *IEEE Software*, 35(3):36–43.

Raj, V. and Ravichandra, S. (2018). Microservices: A perfect SOA based solution for Enterprise Applications compared to Web Services. In *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, page 1531–1536.

Ruan, L., Guo, S., Qiu, X., Meng, L., Wu, S., and Buyya, R. (2021). Edge In-Network Computing Meets Blockchain: A Multi-Domain Heterogeneous Resource Trust Management Architecture. *IEEE Network*, 35(5):50–57.

Sensoy, M., Yilmaz, B., and Norman, T. J. (2013). Discovering Frequent Patterns to Bootstrap Trust. In Cao, L., Zeng, Y., Symeonidis, A. L., Gorodetsky, V. I., Yu, P. S., and Singh, M. P., editors, *Agents and Data Mining Interaction*, page 93–104. Springer.

Steinegger, R. H., Giessler, P., Hippchen, B., and Abeck, S. (2017). Overview of a domain-driven design approach to build microservice-based applications. In *The Thrid Int. Conf. on Advances and Trends in Software Engineering*.

Sun, Y., Nanda, S., and Jaeger, T. (2015). Security-as-a-Service for Microservices-Based Cloud Applications. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 50–57.

Tang, M., Dai, X., Liu, J., and Chen, J. (2017). Towards a trust evaluation middleware for cloud service selection. *Future Generation Computer Systems*, 74:302–312.

Venčkauskas, A., Kukta, D., Grigaliūnas, Š., and Brūzgienė, R. (2023). Enhancing Microservices Security with Token-Based Access Control Method. *Sensors*, 23(6):3363.

Wahab, O. A., Bentahar, J., Otrok, H., and Mourad, A. (2015). A survey on trust and reputation models for Web services: Single, composite, and communities. *Decision Support Systems*, 74:121–134.

Wang, Y. and Vassileva, J. (2007). A Review on Trust and Reputation for Web Service Selection. In *27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07)*, page 25–25.

Wu, Q., Zhu, Q., and Li, P. (2015). A neural network based reputation bootstrapping approach for service selection. *Enterprise Information Systems*, 9(7):768–784.

Yahyaoui, H. and Zhioua, S. (2013). Bootstrapping trust of Web services based on trust patterns and Hidden Markov Models. *Knowledge and Information Systems*, 37(2):389–416.

Zacharia, G., Moukas, A., and Maes, P. (2000). Collaborative reputation mechanisms for electronic marketplaces. *Decision Support Systems*, 29(4):371–388.

Zhang, J. and Li, D. (2022). A Comprehensive and Unified Approach to Web Service Trust Evaluation Based on Uncertainty Methodology. *Entropy*, 24(22):243.