StoRM: A Social Agent-based Trust Model for the Internet of Things Adopting Microservice Architecture

Kalliopi Kravari, Nick Bassiliades
School of Informatics, Aristotle University of Thessaloniki, GR-54124, Thessaloniki, Greece,
{kkravari, nbassili} AT csd.auth.gr

Correspondence concerning this article should be addressed to:
Kalliopi Kravari, School of Informatics, Aristotle University of Thessaloniki, GR-54124 Thessaloniki, Greece.
E-mail: kkravari AT csd.auth.gr
Tel: 00302310998231
Fax: 00302310998418

Abstract

Over the last years, the Internet of Things attracted much attention mainly due to its potential to change our daily life. It attempts to create a world where everyone and everything will be connected while knowledge will be diffused effortlessly. Yet, this open, distributed and heterogeneous environment raises important challenges, such as intelligence and trustworthiness. Intelligent Agents can deal with these challenges since they form an alternative to traditional interactions among people and objects while, at the same time, they are involved in a rich research effort regarding trust management. Additionally, intelligent agents seem able to deal with potential societal impacts and relationships, although they are not primary social networks, as well as the heterogeneity in the Internet of Things when combined with novel approaches such as the microservice architecture. To this end, this article proposes a novel, reputation oriented, trust model, called StoRM, for the Internet of Things that combines social dimensions and microservice architecture with agent technology. StoRM is based on well-established estimation parameters while it provides a reputation estimation mechanism based on social principles. Additionally, it proposes the use of microservices combined with learning and adoption properties facilitating the implementation of the agent-based system and the trust establishment among its members. Furthermore, it adopts a distributed locating mechanism based on social graphs and peer-to-peer networks. StoRM combining a set of features is able to address many of the challenges of trust management in the Internet of Things while it is one of the first approaches that involve the microservice architecture in a trust management model. Finally, a multi-agent simulation is presented that illustrates the viability of the proposed approach.

Keywords: Multi-agent Systems, Internet of Things, Agent-Based Model, Trustworthiness, Social Criteria, Microservice Architecture.

1. Introduction

The Internet of Things (IoT) is more than a scientifically popular trend, it actually transforms the way people live, work and communicate. There are already plenty of existing industrial applications that benefit from this large scaled environment. Smart environment, living and healthcare are just a few cases. Actually, as ambient intelligence is becoming more widespread, sensors and smart devices are included more and more often in new buildings and vehicles [44]. The IoT attempts to create a world where everyone and everything, called Things, will be connected, possibly cooperating to solve complex cases while knowledge will be diffused effortlessly in every direction. However, connecting such a large amount of heterogeneous entities and devices in an open and distributed network provides not only benefits and possibilities but also rises important challenges, leaving space for improvement. Intelligence and trustworthiness are some of the most important of them since entities have to trust each other in order to collaborate exchanging data or offering services while the heterogeneity makes it difficult to standardize the interaction and communication in the network.

The open and distributed environment allows the rapidly increasing Things to enter into the environment and reproduce themselves or create and delete other Things in the network. Malicious participants could pose a serious threat to the proper functioning of the network, harming its credibility through fake services, denial of cooperation or other malicious behaviors. Given this, involved parties are likely to be faced with a large amount of possible partners with a different degree of efficiency and/or effectiveness. Hence, Things acting in such an open and risky environment will have to make the appropriate decisions about the degree of trust that can be invested in a certain partner, a vital but still challenging task [1][43][45][47][50][61][54].

In this context, Intelligent Agents (IAs) are considered as an appropriate and promising technology that can deal with these challenges since they form an alternative to traditional interactions among people and objects [10][22][6][62][23][3]. Intelligent agents are increasingly used among others in networking and mobile technologies in order to achieve automatic and dynamic behavior, high scalability and self- healing networking, promoting flexibility and trustworthiness [3][23]. Their capability of autonomously representing people, devices or even services allows them to be applied in many real world applications, including health care and medical diagnostics, crisis management and green growth. Hence, an IoT-enabled environment

where Things, such as sensors and devices, will be able to communicate with each other can be considered as a multi-agent system, which in its turn is subject to the principles and challenges of agent technology [57][43][41][19].

At the same time, multi-agent systems are involved in a rich research effort regarding trust management. A wide range of trust and reputation models have already been proposed by the research community, even though many of them refer to the Semantic Web, the predecessor of the Internet of Things [42][46][60][12][56][63][27][49]. In general, reputation is the opinion of the public towards a party or agent in particular. Reputation allows parties to build trust, or the degree to which a party has confidence in another party, helping them to establish relationships that achieve mutual benefits. Hence, reputation (trust) models help parties to decide who to trust, encouraging trustworthy behavior and deterring dishonest participation by providing the means through which reputation and ultimately trust can be quantified [60][27][49].

Although, neither the IoT nor multi-agent systems (MASs) are considered primary social networks, examining the potential societal impacts and relationships among Things, objects and/or people, in the IoT is absolutely essential. In fact, research on the IoT is expected to shift from intelligent objects to objects with a real social consciousness. The background motivation is obvious since misbehaving owners with discriminatory behavior based on their social relationships will possess misbehaving devices and services for personal gain. Trustworthiness in such an environment, where objects and even people will try to preserve their unique characteristics, is complex and crucial. Hence, the social dimension of the IoT is currently a new open research area [62][5][29][63][11][27][58].

Additionally, intelligent agents seem able to deal not only with social dimensions but also with another important issue of this heterogeneity in the IoT when combined with novel approaches such as the microservice architecture. Actually, over the last years, microservices attracted much attention in software development, mainly due to the fact that it is hard to maintain vast amounts of code, oriented to centralized approaches. Hence, modular coding gained ground. Microservices have a lot of similarities with both the IoT and intelligent agents due to their distributed nature. Moreover, like agents and Things, a microservice can be considered as an independent individual that provides a particular service, collaborating with others in order to accomplish a goal and provide the requested final result

[13][9][24][18][55][20]. Yet, there is still a lack of approaches that combine agent technology with microservices in the Internet of Things regarding reputation (trust) issues.

To this end, this article proposes a novel, reputation oriented, trust model, called StoRM, for the IoT that combines social dimensions and microservice architecture with agent technology. The aim of this model is to allow Things, objects and people, to establish and maintain social relationships based on their experiences, preferences and requirements without complex underlying network protocols. StoRM is based on well-established estimation parameters [14][26][28][53] while it provides a reputation mechanism based on social principles. Additionally, StoRM proposes the use of microservices combined with adoption properties facilitating the implementation of the agent-based system and the trust establishment among its members. Hence, Things are able to dynamically adjust to the environment encouraging trustworthiness behavior. Furthermore, it adopts a distributed locating mechanism based on social graphs and peer-to-peer networks, in order to deal with the common challenge of locating ratings for open, large-scale distributed systems, such as multi-agent systems and IoT. As a result, StoRM combining a set of features is able to address many of the challenges of trust management in the Internet of Things while it is one of the first approaches that involve the microservice architecture in a trust management model. Finally, a multi-agent use case simulation is presented that illustrates the viability of the proposed approach.

The rest of the article is organized as follows: The next section presents the evaluation criteria along with a discussion on trust, reputation and risk. Section 3 presents a review of microservices, their advantages and their similarities with multi-agent systems. Section 4 presents StoRM and its contribution while Section 5 presents its evaluation simulation, demonstrating the added value of the approach. Section 6 discusses related work, and Section 7 concludes with final remarks and directions for future work.

2. Trust, Risk and Evaluation Criteria

Trust and reputation have been widely studied in the literature since they are considered key elements in the design and implementation of modern (multi-agent) systems. However, the research community that focuses on the IoT, does not have yet thoroughly studied trust management. As a result, although there are plenty of trust definitions, reputation and trust are sometimes confused and used even as synonyms. Trust is generally defined as the expectation of

competence and willingness to perform a given task and it is used as the basis for decision making in many contexts [16][32]. Trust, however, is much more than that; it is a very complicated concept that can be influenced by many properties. The uncertainties found in the modern MASs and the IoT present a number of new challenges. In open, distributed and large-scaled systems, such as the IoT, agents represent different stakeholders that are likely to be self-interested and might not always complete requested tasks. Moreover, given that the system is open, usually no central authority can control all agents, which means that agents can join and leave at any time. This allows agents to change their identity and re-enter, avoiding punishment for any past wrong doing. One, more, risky feature of open systems is that when an agent first enters the system has no information about the other agents in that environment. Given this, the agent is likely to be faced with a large amount of possible partners with a different degree of efficiency and/or effectiveness.

Hence, since agents such as individuals may be dishonest, or just fail even if they are not being dishonest, reputation ended up as a core element at trust establishment, in the sense that a better reputation can lead to greater trust. In general, reputation is best viewed as a history-based estimation of the likelihood of certain behavior which is an approach that can be used to directly calculate expectations. Reputation allows agents to build trust, or the degree to which one agent has confidence in another agent, helping them to establish relationships that achieve mutual benefits. In other words, reputation is an estimated opinion of a party for another party. Hence, usually reputation is a personal and subjective quantity, referring not to what behavior a party has but rather what behavior others think that party has [42][46][60][12][56][63][27][49]. Risk, on the other hand, is often undertaken in the hope of some gain or benefit. Risk is actually a situation that involves exposure to danger or loss, since although the outcome of a transaction is important to a party, the probability of loss is non-zero. Risk can also be defined as the intentional interaction with uncertainty, in the sense of a potential, unpredictable outcome [16]. Hence, the amount of risk that a party may be willing to tolerate is directly proportional to the amount of trust that the party has in the other party [33]. As a result, the main aim of reputation models is to support the establishment of trust between unfamiliar parties, equilibrating the risk.

At this point, for purposes of better understanding we have to define the involved parties. This study considers all parties as intelligent agents and the environment of the Internet of Things as a multi-agent system. In other words, each involved Thing, object, service or even

human, is considered as an agent (Figure 1). In this context, an agent A that interacts with another agent X can evaluate X's performance and thus affect its reputation. The evaluating agent (A) is called *truster* (here TR) whereas the evaluated agent (X) is called *trustee* (here TE). Furthermore, there are two more potential roles with a slight difference, namely *Recommenders* (here RR) and *Witnesses* (here WS). A witness provides reports based on personal previous experience with the agent under evaluation whereas a Recommender may have no previous interaction history with that party. A Recommender usually propagates reports based on others' experience or observation.

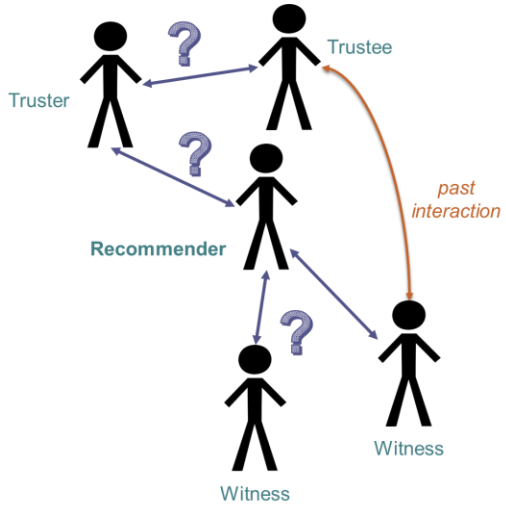

Figure 1. Roles of Involved Parties.

For some interactions an agent can be both truster and trustee, since it can evaluate its partner while it is evaluated by that partner. After each interaction in the environment, the truster has to evaluate the abilities of the trustee according to the evaluation criteria proposed by the (StoRM) model. These evaluation criteria vary from model to model. Yet, there are some that are usually referred either explicitly or implicitly in reputation models and metrics [42][46][60][63][56][49][27]. The most common metrics involve six properties; namely *response time*, *validity*, *correctness*, *cooperation*, *quality of service* (QoS) and *availability*.

Response time refers to the time that an agent needs in order to complete the tasks that it is responsible for. Time is the only parameter that is always taken into account in the literature. *Correctness*, no matter how it is called, is, actually, the second most used parameter after time. An agent is correct if its provided service or task is correct with respect to a specification. Sometimes, it is called *delivery*, especially in cases that involve services or devices. *Validity*,

sometimes referred as honesty, describes the degree that an agent is sincere and credible. An agent is sincere when it believes what it says, whereas it is credible when what it believes is true in the world. Hence, an agent is valid if it is both. Validity is not always such called, yet in most cases there are parameters that attempt to indicate how sincere and/or credible an agent is. *Cooperation* is not usually handled as separate parameter; however, it is an important feature in distributed social environments, referring to the willingness of an agent to do what it is asked for. Sometimes, it is even called *flexibility*. Quality of service (QoS) refers to the overall performance of a service, we adopt this criterion mainly referring to the case of services. To quantitatively measure quality of service, a number of characteristics are taken into account such as packet loss, bit rate, transmission delay, availability, failure probability and so on. Finally, *availability* refers mainly to the involved devices, or services, indicating the degree to which the device is in operable and committable state.

However, although these six parameters are, usually, taken into account in one way or another, they are not necessarily binding. Some of them could be replaced by other more domain-specific parameters depending on the domain of use. For instance, in a smart environment case, energy consumption related to each involved device could be possibly one of the most important evaluation criteria while in e-Commerce transactions price and payment variety would be more important. Yet, since StoRM is a general purpose trust management model, our intention is to adopt the aforementioned six criteria, four generic, one service-oriented and one device-oriented, that reflect the common critical characteristics of each involved party in an extensible list. More details will be provided below at the model section.

3. Microservices and Intelligent Agents

IAs are a well-studied technology that offers plenty of possibilities, especially autonomy. Lately, they were considered as a potential evolution or addition to microservices and, as a result, to the development of the IoT [55][13][18][20]. The motivation behind this is the possibility of adding autonomy, context awareness, and intelligence as natural behaviors to IoT and microservice architecture. Due to the distributed architecture and the dynamic environment, MASs, IoT and Microservices give an impression of similarity. Actually, microservices have almost every known agent property (Table 1), sharing many similar architectural characteristics (Table 2) [51].

Table 1. Agent and microservice properties

	Property	**Intelligent Agents**	**Microservices**
1.	**Autonomy**	X	X
2.	**Adaptability**	X	■
3.	**Mobility**	X	X
4.	**Migration**	X	X
5.	**Learning**	X	X
6.	**Reactivity**	X	■
7.	**Social ability (Collaboration/ Coordination/Interaction)**	X	X
8.	**Persistence (execution)**	Continuously	On demand
9.	**Proactivity**	X	■

Both microservices and intelligent agents are autonomous since they are able to accomplish their tasks without human intervention and independently of other services or agents. Mobility is another common feature since it is achievable by microservices as they can move and run independently in the network. Furthermore, microservices have the ability to migrate, yet the property of migration, not specific to agents, can be also contrasted to mobility. In general, as far as it concerns migration, it suspends the execution of an agent or service until it reaches its destination while mobility usually creates a clone at a remote destination. Reactivity could be considered as a microservice property in the sense that each service is responsible to run and return the appropriate data under demand. This behavior can be viewed as reactivity. Both technologies have also social abilities; interaction, collaboration and coordination are not only agent properties but also three of the basic properties of microservices. Learning and adaptation are two common agent properties but they are difficult to be adopted by microservices. Usually, microservices neither learn nor easily adapt new behavior. However, when combining with technologies such as agent technology it could be considered achievable. Of course, despite the similarities, microservices and agents differ in persistence; agent run continuously and decides for themselves when to perform some activity whereas microservices run on demand, executing a specific part of code. Hence, agent are able to act proactively opposed to microservices that run upon demand.

Actually, the difference between agent technology and microservices is in how they fulfill the aforementioned characteristics in order to address challenges. Microservices modify their behavior due to code changes whereas agents modify their behavior due to contextual

changes since they collaborate and perceive or even influence their dynamic environment. Table 2 presents architecture similarities and differences. A main difference between microservices and agents is the contextual behavior that can be demonstrated by an agent as opposed to microservices. Agents like microservices are goal-driven but they try to fulfil these goals by using knowledge of their own capabilities, the environment or other agents in order to make appropriate decisions. On the other hand, microservices allow tenderers to control devices even if they are physically hard to reach. Hence, combining these two technologies, considering each microservice as a single agent, could lead to novel, more efficient approaches for the IoT.

Table 2. Agent and microservice architecture similarities and differences

	Characteristic	**Intelligent Agents**	**Microservices**
1.	**Distributed**	distributed *code* along with distributed *knowledge* and *intelligence*	distributed (non-monolith) *code*
2.	**Modular**	Agents using services, metadata etc	services
3.	**Flexibility**	flexible to *code* and *contextual* changes	flexible to *code* changes
4.	**Behavioral**	*reactive*, *proactive*, *cognitive* (contextual)	*reactive*
5.	**Decoupled**	agent to *agent/service/environment*/etc	service to *service*
6.	**Inter-process communications**	event-driven/coordination-communication abilities/ *common and semantic* data structures	event-driven/data exchange/*common* data structures

4. StoRM

The proposed model is called StoRM and it is a distributed, hybrid reputation model. It is one of the first approaches that uses social principles and microservice architecture in order to combine in a practical way all available ratings, both those based on the agent's personal experience and those provided by known and/or unknown third parties in the IoT. StoRM aims at providing a distributed mechanism that would be able to model the way entities, services and devices communicate and interact in the IoT, overcoming their heterogeneity.

4.1. Microservices in StoRM

First of all, as already discussed, we model three main types of (IoT) Things; entities (human or virtual), services and devices. We assign an extendable list of characteristics C and preferences P to each entity (LC_x^k & LC_x^m | $k,m \in [1, N]$, $x \equiv entity$), where k and m represent the number of

characteristics and preferences, respectively. The C list contains much more than the entity's type, such as registration date or time period it is or will be active in the environment, assigned roles, energy consumption and so on. On the other hand, preferences include information such as the desirable temperature degree or the desirable charging level.

For computational and priority purposes, each characteristic and preference is assigned with a value of importance (weight) at the range [0, 1] (W_c^k & W_p^m | $k,m \in [1, N]$, $c \equiv$ *characteristic*, $p \equiv$ *preference*), defining how much attention will be paid to each characteristic or preference. For instance, an electric emergency vehicle should be always charged. Hence, its type, role and time that it will be active in the environment characteristics are important and, thus, they weigh more. For instance, in an environment with limited availability in energy, this vehicle would have priority among the rest vehicles.

All types of Things are represented as agents while microservice architecture was used for the implementation of services and devices, achieving the necessary functionalities and reducing the common issue of device handling in the IoT, especially for the trust models. More specifically, since services and especially devices are usually small with limited resources while microservices should be small and autonomous but at the same time collaborative [48], as far as it concerns devices we propose the implementation and use of two types of microservices; namely *device_microservice* and *gateway_microservice*, represented in the environment as agents (Figure 2). In case of service an implemented *service_microservice* is used.

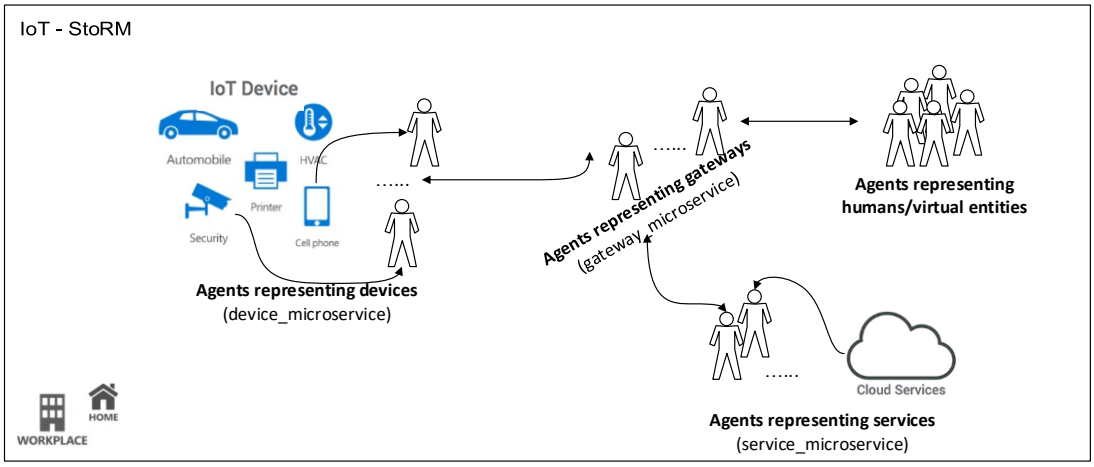

Figure 2. StoRM overview.

Each device_microservice is related to a specific IoT device and it is responsible for its function and/or data exchange. Complicated devices with more functionalities could possess

more than one device_microservice that will collaborate in order to represent and handle properly the device and its data. device_microservice tasks include storage of received data and propagation of generated data (e.g. sensor data), response on demand, mode handling (e.g. start/stop). In other words, device_microservice (or a set of such microservices) is an agent that represents a specific device in the environment.

On the other hand, devices and even services are usually non-smart, isolated entities. Hence, it is important for the rest of the community to reach and communicate with them. For this purpose, we propose the use of the so-called *gateway_microservice*. This microservice should handle several tasks since it is actually a middleware between devices and other IoT Things. Actually, the main task of a gateway_microservice is to periodically discover reachable services and devices in the environment, store the related information and inform, on demand, other Things. For instance, consider a printer device. A gateway agent (gateway_microservice) will store all information such as printer global name and its characteristics (e.g. ink level) and as soon as another Thing asks for printer, the gateway agent will inform about the available printers in the environment. Additionally, it could start or stop a specific device or even modify its transmission frequency if requested by the proper user, such as the owner of a printer or sensor.

```
ACLMessage msg = receive();
if (msg != null) {
    ACLMessage reply = msg.createReply();
    sender = msg.getSender();
    if (msg.getPerformative() == ACLMessage.INFORM) {
        //receives a reply (to its request message) by the service (service_microservice) in order to register it
        content = msg.getContent();
    } else {
        reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
        reply.setContent("(Unexpected-act " + ACLMessage.getPerformative(msg.getPerformative()) + ")" + "Try Again!!");
        send(reply);
    }
    //registration process....
    if (content != null) {
       registration = fileToString(content); //reads the message and stores its content to a string
       try {
            DFAgentDescription dfd = new DFAgentDescription();
            dfd.setName(getAID());
            //Register the service
            ServiceDescription sd = new ServiceDescription();
            sd.setType("Broker");
            sd.setName(getLocalName() + " Service");
            dfd.addServices(sd);
            DFService.register(this, dfd);
        } catch (Exception e) {
        System.out.println(e);
        }
        System.out.println(this.getLocalName() + " has registered as Broker service");
    }
}
```

Figure 3. Part of a gateway_microservice agent source code.

Part of the source code of a gateway_microservice agent is presented in Figure 3, where a new broker service registration is coded. This source code is written in Java and implemented in EMERALD, an interoperating knowledge-based framework [39], which is based on JADE [8] (more details about EMERALD can be found at section 5.1). More specifically, the aforementioned gateway_microservice agent had sent a request message to a broker service in order to invite it for registration. Figure 3 displays the code following that. In particular, as soon as the gateway_microservice agent receives a (nonempty) message, in ACL format [8][52], it checks its performative act. If this is "INFORM" then the agent confirms that it receives a reply by the service (service_microservice) in order to register it. Hence, next, the service is registered, assigned to a new DFAgentDescription. To this end, a new service description is created including the type of the service ("Broker") and its name, allowing other agents in the environment to locate it.

4.2. Rating Parameters and Evaluation Procedure

As soon as two agents interact, agent truster (TR) evaluates trustee (TE) and stores this rating to its private repository, here called *ERep*. For evaluation purposes, as already discussed StoRM adopts an extendable list ($EC^n \mid n \in [1, N]$, here $N = 6$) of six criteria, namely response time, validity, correctness, cooperation, quality of service (QoS) and availability. The first five apply to all IoT Thing categories but quality of service is used only for services whereas availability can be used for both services and devices. However, these are not enough. The truster (TR) has to indicate how confident it is for its rating, allowing later other agents to decide upon how much attention to pay on this specific rating. For instance, a rating provided by an unsure agent will be considered with skepticism or even ignored. Additionally, since time is an important aspect, especially in decision making processes, each rating is associated with a time stamp (t), indicating the time point of the transaction. In [21] authors discussed the problem of time management in distributed systems and proposed a promising solution. Based on that a reference to the universal time is needed. To this end, they propose the use of SNTP protocol to establish the universal time and offset between that time and the local clock. Our model can adopt this approach, although at this point we describe its architecture.

Hence, taking the above into account the truster's rating value (r) in StoRM is a tuple with eight, nine or ten elements, depending on the number of the evaluation criteria:

r := (TR, TE, t, response time, validity, correctness, cooperation, confidence) (1) or

r := (TR, TE, t, response time, validity, correctness, cooperation, availability, confidence) (1') in case of devices, or

r := (TR, TE, t, response time, validity, correctness, cooperation, QoS, availability, confidence) (1'') in case of services.

Although each truster agent stores its own ratings, the r tuple includes the variable TR (truster global name) since the truster may forward its ratings to other agents; hence, these agents should be able to identify the rating agent for each rating they receive. In StoRM, confidence as well as the rest rating values vary from 0.1 (min/terrible) to 10 (max/perfect); $r \in [0.1, 10]$.

Additionally, agents that comply with StoRM do not only rate their parties, storing these ratings to the aforementioned repository ERep but they use more repositories in order to identify friends and malicious partners. More specifically, there are two more repositories, one for storing promising partners (*WRep*, whitelist), partners that act responsibly and provide high quality services or products, and one for those partners that should be avoided (*BRep*, blacklist). Hence, truster has each time to decide if a partner should be added to one of these lists. To this end, each agent has a threshold that determines its degree of *tolerance* and a threshold that determines an indicator value about what it considers as *exceptional* behavior. The procedure is straightforward. First, the TR agent as soon as it stores its rating to ERep, compares the criteria values with its thresholds. More specifically, it computes the weighted average for the criteria that it is interested in, forming the $Wrating_r$ value:

$$Wrating_r = \frac{\sum_{i=1}^{n} w_i \times r^{EC^n}}{\sum_{i=1}^{n} w_i} \quad (2)$$

where w_i is a weight defined by the TR for each selected criterion (EC^n). Next, the TR agent compares this value with its tolerance and exceptional values; if $Wrating_r \leq tolerance$ the trustee (TE) agent is stored in TR's blacklist (BRep) whereas if $Wrating_r \geq exceptional$ the TE agent is stored in TR's whitelist (WRep), if it is not already there. Additionally, in order to avoid characterizing a TE agent as malicious (BRep list) due to temporary misbehavior (e.g. functional

error or misunderstanding) StoRM indicates that if RR and/or WS agents recommend a TE agent that is stored at TR's BRep more than two times, TR will remove that agent from its BRep list.

4.3. Adopting LOCATOR

A major challenge for open, distributed and large-scale systems, such as MASs and the IoT, is how to locate ratings among the rest of the community. To this end, StoRM adopts LOCATOR [36], a locating rating mechanism that uses features from social graphs [32] and peer-to-peer networks [4] in order to incorporate potential social dimensions and relationships within the IoT.

LOCATOR in order to convince parties to provide ratings, since entities are unwilling to sacrifice time and resources, uses a reward mechanism. According to this mechanism, each party will get a credit whenever it provides a recommendation. The credit could be positive or negative and since there is no central authority to monitor and store credits, each agent should store by itself its credit score. Moreover, since time is important, each credit is valid only for a specific time period, depending on the personal strategy of the TR agent that requests the recommendations. A high credit score provides an evidence about the activity and recommendation quality of an RR (or WS) agent. The higher the credit score is the more weighted is the recommendation (trustworthy partner).

As far as it concerns propagation mechanism, LOCATOR inspired by social graphs and P2P networks considers the environment as a social network of agents. Practically, this social network is represented by a social graph; a graph based on agent interactions. Hence, although the notion of neighbors does not exist in IoT and MASs, agents can use previously known partners (agents that have already interacted) in a similar point of view. Using the knowledge represented by the social graph, LOCATOR determines the relationships of interactions among agents and the proximity between parties in the environment. LOCATOR, based on trusted paths, identifies three categories of neighbors according to their social distance from the truster, called local neighbors, longer ties and longest ties, respectively. A trusted path is a path that consists of a truster (TR – the source), several recommenders (RR agents) or witnesses (WS), a trustee (TE – the target), and trust relations among them. In other words, it is a trusted path from the truster to the trustee. Hence, local neighbors are agents that have previously interacted with truster, longer ties are agents that can be connected to the truster with a path length less than five (<=5) nodes and longest ties are agents that can be connected with greater path length (>5).

Value 5 was chosen due to a number of simulation experiments conducted in the context of LOCATOR that revealed that agents tend to interact more with agents connected to them within such a path. This was probably due to more available recommendation/witness reports from already known (closely connected) agents.

Hence given a trusted path, propagation works in this way: if agent A_1 trusts agent A_2, and A_2 trusts agent A_3, then A_1 can derive some trust towards A_3. The challenge is to set a proper limitation of path length, since a smaller limitation may lead to fewer paths, while a larger one may cause inaccurate prediction. Usually, in P2P networks there is a maximum time-to-live (TTL) parameter assigned to each request message, which means that a message will be propagated for a specific time period. Adopting the notion of TTL, in LOCATOR, each request message is accompanied with a TTL value, yet it represents neither the time that the message is valid nor the maximum path length (hops in the graph) but rather the time period that the truster will wait for response. In other words, truster does not determine how far the message will be propagated in the network but specifies how fast it needs feedback. This way, truster is able to locate reports quite fast and make quick decisions. Of course, if more accuracy is needed, a longer time parameter should be assigned.

LOCATOR works as follow (Figure 4). Firstly, an agent TR interested in a trustee agent TE, based on its preferences (LP_{TR}^{m}) decides upon the characteristics (the aforementioned LC_{TE}^{k} list) it considers important, e.g. a club membership. Next, it assigns proper weights (the aforementioned W_c^k) to each of them and searches its database for previously known agents (local neighbors) in order to find those that fulfil its requirements. Characteristics that weight more are more important in the sense that TR believes that partners with these characteristics will be more reliable (social influence). As a result, their recommendation is expected to be more valuable. In this context, TR depending on its personal strategy sends a rating request firstly to local neighbors with one or two high-weighted characteristics. For instance, partners that provide the same service or had a previous successful transaction with TR. If the feedback is not satisfying, TR may sent a request message to partners with lower-weighted characteristics.

After choosing the local neighbors that will be the direct receivers of the request, TR assigns two time thresholds to its request message, a TTL value and a requested credit time period, and sends it to them. They acting as recommenders (RR agents), on their turn, propagate

the message to their own local neighbors following the same procedure as long as they have time (t < TTL). Finally, these RR agents send the feedback (recommendation and credit score) to TR. Feedback is, actually, trusted paths from TR to TE through RR agents.

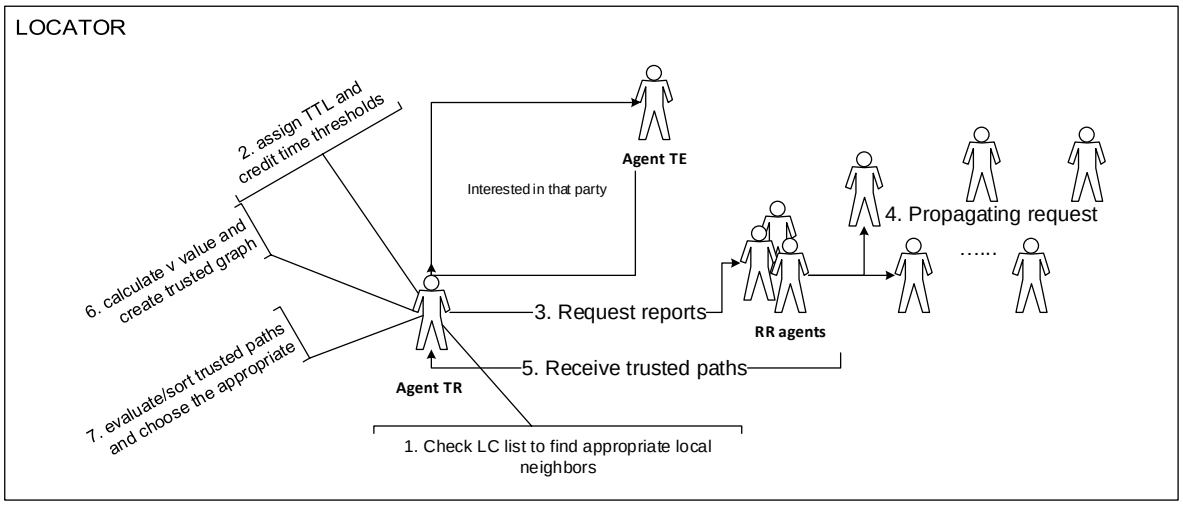

Figure 4. Brief functionality overview of LOCATOR.

At the next step, TR assigns a value V, an indication of relevance, to each received trusted path. This value is calculated as follows:

$$V = \left(pl - 0.25 \times hp \right) \times C_{RR,\forall} \, pl \le 5 \text{ or } V = \left(pl - 0.5 \times hp \right) \times C_{RR,\forall} \, pl \ge 6 \quad (3)$$

where pl stands for the length of the trusted path, hp stands for the number of network nodes while C_{RR} is the credit score of the local neighbor (RR agent) that returned that path. C_{RR} is based on RR agent's credits with a time stamp that fits in TR requested time period. Using this time period, TR has a clue about RR's latest behavior. The V value attempts to discard feedback, taking into account the concept of risk. More specifically, longest ties are more possible to be completely strangers even for TE's local neighbors. Hence, they can be considered as less trusted, which means that TR will take more risk. On the other hand, longer ties are more possible to be previously known partners of the TE's local neighbors and probably they are more valuable recommendation sources.

4.4. Discarding Ratings

At this point, TR received feedback and created a trusted graph by combining all available trust paths. For multiple trusted paths in a trusted graph such as this, the main challenge is how to

combine the available evidence. The V value, an indicator value estimating the risk and social proximity, provided by LOCATOR will be used for this purpose combined to TR's preferences. To this end, StoRM proposes the following discarding algorithm that allows TR to choose the best among the available ratings according to its personal strategy:

Input data: $\left\{trPath_x^V\right\}$*, the set of the returned trusted paths along with the V value*

Step 1: Sort in descending order all return trusted paths according to the V value.

Step 2: Choose X (according to TR's strategy) of them; those with higher V value.

Step 3: Extract ratings from these paths.

Step 4: Remove from the above ratings those with low confidence value

(confidence < (TR's) confidence_threshold).

Step 5: Remove from the remaining ratings those provided by agents stored in TR's BRep list.

Step 6: Sort remaining ratings in descending order according the time (t value).

Step 7: Choose X of the above ratings; those that fit in TR's desired time period or the newest.

Output data: $\left\{r^{EC^x}\right\}$, the set of the finally chosen X ratings

Hence, TR, using the above discarding algorithm, will be able to proceed with the most promising (possible trustworthiness) and more recent ratings. Notice that it is included a step (step 5) that allows TR to avoid ratings from partners that it is unwilling to interact with or had a previous bad experience.

4.5. Estimation Mechanism

In order to better analyze rating data, crossing out extremely positive or extremely negative values, rating values in StoRM are logarithmically transformed. Actually, the most important feature of the logarithm is that, relatively, it moves big values closer together while it moves small values farther apart. And this is useful in analyzing data, because many statistical techniques work better with data that are single-peaked and symmetric. Furthermore, it is easier to describe the relationship between variables when it is approximately linear. Hence, each rating is normalized ($r \in [-1,1]$ | -1 ≡ terrible, 1 ≡ perfect), by using 10 as base. As a result, the final reputation value ranges from -1 to +1, where -1, +1, 0 stand for absolutely negative, absolutely positive and neutral (also used for newcomers), respectively, which means that an agent's reputation could be either negative or positive.

Hence, the final reputation value (R_x, x ≡ entity) of an agent, at a specific time t, is based on the weighted average of the relevant reports (normalized ratings) and is calculated as follows:

$$T(t) = \frac{w_{per}}{w_{per} + w_{rec}} \times \frac{\sum_{\forall t_{start} < t_i < t_{end}} \frac{w_{pr}\ x\ log\left(r^{EC^n}\right) \times t_i}{\sum_{pr=1}^{N} w_{pr}}}{\sum_{\forall t_{start} < t_i < t_{end}} t_i} + \frac{w_{rec}}{w_{per} + w_{rec}} \times \frac{\sum_{\forall t_{start} < t_j < t_{end}} \frac{w_{rc}\ x\ log\left(r^{EC^m}\right) \times t_j}{\sum_{rc=1}^{M} w_{rc}}}{\sum_{\forall t_{start} < t_j < t_{end}} t_j} \quad (4)$$

There are some important aspects in equation (4). The first is that the normalized ratings are divided to two groups, one referring to direct experience (transaction between TR and TE), with W_{pr} weigh values, and one referring to those third-party ratings, with W_{rc} weigh values, that were obtained as described above. The second is that TR is able to balance between personal experience (W_{per}) and witness reports (W_{rec}), which is actually an opinion provided by strangers. Finally, since time, as already discussed, is important it is involved in the final value in the sense that recent ratings weigh more.

5. Evaluation Simulation

In order to use and evaluate StoRM, the proposed model, we conducted a number of simulation experiments that model potential IoT environments. The aim of these simulations were not only to evaluate the added-value of the approach but also to consider about possible composition of an IoT environment. For instance, a smart living case would include a higher percentage of device entities whereas an eCommerce case would include more service entities.

5.1. Simulation environment

For implementation purposes, we adopt the use of EMERALD [38], a framework for interoperating knowledge-based intelligent agents (Figure 5). It is built on top of JADE [8], a reliable and widely used multi-agent framework, and it is fully FIPA-compliant. EMERALD was chosen since it provides a safe, generic, and reusable framework for modeling and monitoring agent communication and agreements. Moreover, EMERALD was involved in cross-community interoperations such as in [37]. It proposes, among others, a reusable prototype for knowledge-customizable agents (called KC-Agents) and the use of Reasoners [39].

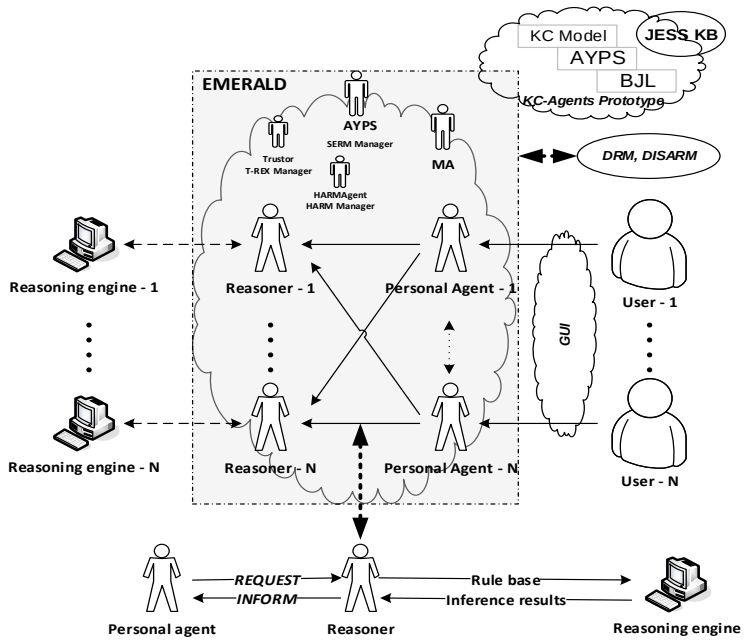

Figure 5. EMERALD overview.

The agent prototype promotes customizable agents, providing the necessary infrastructure for equipping them with a rule engine and a knowledge base (KB) that contains agent's knowledge and personal strategy. Complying with this prototype, agents have their ratings expressed as facts in RDF format (Figure 6). Reasoners, on the other hand, are agents that offer reasoning services to the rest of the agent community. A Reasoner can launch an associated reasoning engine, in order to perform inference and provide results. EMERALD supports a number of Reasoners, among others deductive and defeasible cases.

```
<rdf:RDF>
  <storm:rating rdf:about="&storm_ex">
    <storm:id rdf:datatype="&xsd;integer">1</storm:id>
    <storm:truster>tr1</ storm:truster>
    <storm:trustee>te1</ storm:trustee>
    <storm:t>175</storm:t>
    <storm:response_time>9</storm:response_time>
    <storm:validity >7</storm:validity >
    <storm:correctness>6</storm:correctness>
    <storm:cooperation>9</storm:cooperation>
    <storm:availability>7</storm: availability >
    <storm:confidence>9</storm:confidence>
  </ storm:rating>
</rdf:RDF>
```

Figure 6. A rating example in RDF format regarding a device agent.

Additionally, EMERALD provides an advanced yellow pages service, called AYPS, that is responsible for recording and representing information related to registered in the environment agents, namely their name, type, registration time and activity. This information is dynamically stored in the AYPS agent's database. Hence, the service is able to retrieve up-to-date information at any time. Hence, even if StoRM (or any other distributed model) is a distributed reputation model, agents that use it are able to send requests to AYPS in order to get first a list of potential partners, which is the case for newcomers. Next, they can use the StoRM model in order to estimate reputation for one or more of them in order to find the most appropriate partner (higher reputation value). Of course, it is not necessary to use such services; it is up to each agent's personal strategy how it will locate potential partners. The more an agent knows the environment, the better it can choose partners.

Finally, EMERALD is an agent platform that supports trust and reputation mechanisms in order to support trustworthiness and efficient decision making in the multi-agent system. It supports both centralized and distributed reputation models. Actually, it has been used so far in studying how agents act on behalf of their users in cases such as trading. In this context, agents in the environment are free to ask others for their opinion (ratings), hence each agent requests a service/functiom from the most trustworthy and reliable provider according to it.

5.2. Testbed

For simulation purposes, as far as it concerns the testbed, we adopted and updated a combination of two, quite popular, testbed environments [30][31] previously used in [35] and [34]. The initial testbed as well as its slight variation that is adopted in this article were developed by a well-known research team for evaluation purposes regarding reputation models. This testbed without loss of generality reduces the complexity of the environment. Furthermore, it allows quickly obtainable and easily reproducible results. The foremost advantage of the testbed is the fact that it provides a realistic view of a multi-agent system's performance under commonly appeared conditions, such as realistic network latency, congestion, user behavior and so forth. In this context, we preserved the testbed design but slightly changed the evaluation settings, taking into account the data provided in previous works. The main differences in our proposed variation is that we include all types of Things (virtual entities, services and devices) but since they are represented by agents there is no functional differentiation.

More specifically, the testbed environment is a multi-agent system, representing IoT environments, consisting of agents seeking for something, e.g. a service, and agents that provide it, namely Seekers and Tenderers. We assume that the performance of a tenderer and effectively its trustworthiness, in a specific service/function is independent of other services/functions that it might offer. Hence, in order to reduce the complexity of the testbed's environment, without loss of generality, we assume that the performance of a tenderer is independent from the service/function/etc that is provided. In this context, it is assumed that there is only one type of service/function/etc per agent type (entities, services, devices) in the testbed and, as a result, all the tenderers offer the same service/function/etc per agent type. Nevertheless, the performance of the tenderers, in terms of correctness, response time, etc., differs and determines the utility that a seeker gains from each interaction (called UG ≡ utility gain). This UG was designed as part of the testbed in order to have a comparison measure. It is not part of the model itself or the framework. A number of evaluation simulations were conducted in EMERALD by other previous works that used also a utility gain value. Yet, that utility was not defined as it is here nor it was part of EMERALD. Here, we consider as utility gain value the average value of all rated criteria in the range UG ∈ [0.1, 10], depending on the level of performance of the tenderer in that interaction. More specifically, UG is calculated as follows based on the tenderer's type:

Entities: $UG = AVG(r^{responsetime}, r^{validity}, r^{correctness}, r^{cooperation})$ (5)

Services: $UG = AVG(r^{responsetime}, r^{validity}, r^{correctness}, r^{cooperation}, r^{QoS}, r^{availability})$ (5')

Devices: $UG = AVG(r^{responsetime}, r^{validity}, r^{correctness}, r^{cooperation}, r^{availability})$ (5'')

A tenderer agent can serve many users at a time. After an interaction, the seeker agent rates the tenderer based on the proposed evaluation criteria. Each agent interaction is a simulation round. Events that take place in the same round are considered simultaneous and, thus, the round number is used as the timestamp for events and ratings.

To this end, we run a large amount of simulation rounds where the environment was populated with agents, seekers and tenderers. Of course, we assume that seekers select always the tenderer with the highest reputation value. Whenever there are no available ratings for a tenderer, its reputation value is zero. Table 3 presents the four types of service tenderers included in the testbed environment. More specifically, the testbed includes good, ordinary, bad and intermittent tenderers, namely honest and malicious agents. The first three provide services/functions according to the assigned mean value of quality with a small range of

deviation. In other words, good, ordinary and bad tenderers tend to have a mean level of performance, hence, their activity (actual performance) follows a normal distribution around this mean, in terms of UG value. Intermittent agents, on the other hand, cover all possible outcomes randomly.

Table 3. Testbed: types of tenderers and performance distribution

Tenderers	***Population Density***	***Tending mean performance (UG value)***	***Performance distribution (UG value)***
Good tenderers	15%	9	[8, 10]
Ordinary tenderers	30%	7	[6, 8]
Bad tenderers	40%	3	[0, 6]
Intermittent tenderers	15%	5	[0.1, 10]

More particularly, regarding their strategy, good tenderers act always honestly, providing immediately seekers with accurate and right services/functions. Ordinary tenderers, on the other hand, are usually honest but they have sometimes a significant delay in response. Hence, ordinary tenderer agents respond always to a call, providing usually the right service/function but most of the times with a delay. The above, good and ordinary tenderers, form the two honest cases. On the other hand, the testbed includes two malicious cases, intermittent and bad tenderers. Intermittent tenderers respond usually without delay but most of the times they do not act as expected but in a wrong way. Bad tenderers respond always with a delay, providing wrong services/functions. Bad tenderers are an obvious bad case, in the sense that they are absolutely malicious agents that act dishonest. As a result, they can be located quite easily with a well-formed reputation model. Yet, intermittent tenderers are a more complicated and dreaded case of malicious agents. They act immediately, providing either good or bad services/functions, without a specific behavior pattern. Hence, it is difficult for the rest agents to detect and reveal their malicious behavior.

In this context, as far as, it concerns the allocation of tenderers, we use a quite common case where just half of the tenderers lead to profit (satisfying UG value), since it is impossible to explore exhaustively allocation possibilities. Furthermore, in real life, good and intermittent tenderers are not so common, hence, these categories get a low percentage (15%) in the population. Additionally to tenderer allocation, we acknowledge the fact that multi-agent systems, just like IoT, are open systems, allowing agents to join or leave the system at any time.

In this context, in order to simulate this dynamic behavior, we remove a percentage of the testbed agents while we add new ones into it, at each simulation round. Yet, our intention is to maintain the different groups of categories and their relevant proportions. Hence, at each round just a 10% to 20% is renewed, by actually replacing agents in the system.

Finally, we take into account another discipline since we study MASs in the context of the IoT. In this context, agents, whether they are seekers or tenderers, they could represent different types of Things, namely entities, services and devices. Hence, taking into account the above testbed settings, we additionally determined three evaluation cases (Table 4). The first case includes an approximately equal distribution among entities, services and devices. The rest two cases assume that the environment has more service or device agents (Things), respectively. Additionally, we provide more storage space to entities and less to services and devices, in order to reflect their usual capabilities in IoT. Furthermore, device agents are equipped with a delay parameter allowing us to model better limitations to device speed.

Table 4. Testbed: distribution of agent (Thing) types

Agents	***1st case: approximately equal***	***2nd case: more services***	***3rd case: more devices***
Entities	34%	20%	20%
Services	33%	60%	20%
Devices	33%	20%	60%

5.3. StoRM evaluation

The first set of simulations was conducted just for the proposed model. For this purpose, all agents complied with the StoRM model, the previously discussed settings were used while each time point (taking integer values) is associated with the respective simulation round. Figure 7 displays the first experimental findings. We conducted three simulation sets, each of them complied with one the three aforementioned cases regarding agent type distribution among entities, services and devices. Figure 7 displays a combination of these results, namely the mean UG value per round taking into account all three simulation sets. The model has, in general, a promising upward trend which indicates that it is able to provide good estimations, allowing agents to reach quite fast possible well behaved partners. However, it needs some time to reach good utility values. This is not surprising since agents need time to interact and create relationships (known and whitelisted agents) in the network.

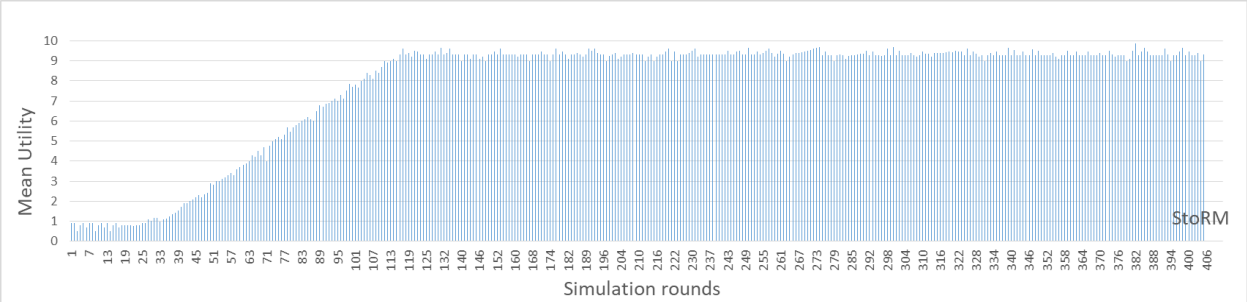

Figure 7. Mean Utility Gained by StoRM over time.

Next, we repeated the experiments more times in order to check the behavior of StoRM regarding the different populated environments as far as it concerns agent type distribution. This allowed us not only to check the model itself but also to study the potential differences among agents that represent different types of Things. This is useful since services and especially devices have limited resources which could lead to less decision or storage possibilities. Our intention was to simulate a worst case scenario where many of the available parties will have limitations. Figure 8 displays the results of this set of simulations. The test settings are the same except the agent type distribution (Table 4).

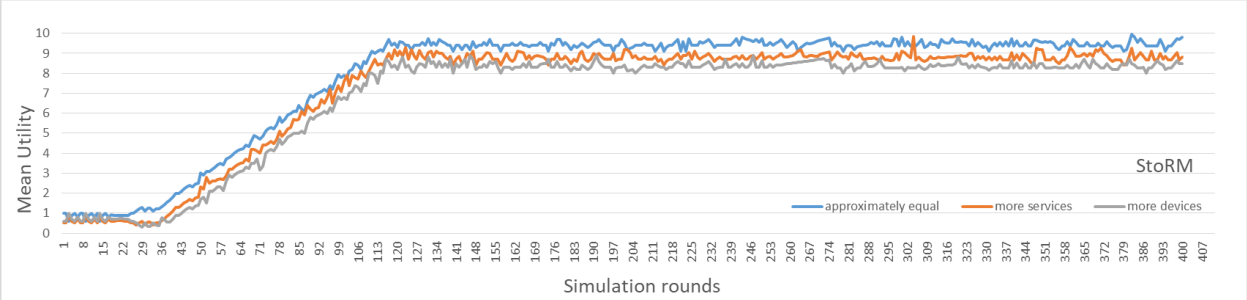

Figure 8. Mean Utility Gained by StoRM over time per agent type distribution.

These findings (Figure 8) support the previous observation; the model has a quite stable upward trend, providing good estimations. More interesting but, actually, not surprising is the difference between the three distribution cases due to the limitation we have modeled. An approximately equal distribution among agent types, entities are slight more (+1%), leads to better results (higher UG values). Agents representing entities have more storage space, allowing them to store more ratings which leads to more available data and better estimation. On the other hand, agents that represent devices have space and speed limitations which lead to data loss and slightly worse (but still good) estimations. Agents that represent services fall somewhere in the middle. Obviously, each case has peculiarities but an IoT application usually has and will have a

mixed distribution while technology evolution will allow devices to getting better and smarter. As a result, models such as StoRM will provide not only a stable behavior but also good estimations even if there are some limitations. The good estimations in all cases are also supported by a modeling decision, the use of microservice architecture. service_microservice, device_microservice and especially gateway_microservice provide autonomy and better behavior.

5.4. Model comparison

In order to further evaluate the proposed model, we checked its behavior in comparison to other known distributed reputation models. In this context, the only difference among seeker agents is the trust models that they use, so the utility gained by each agent through simulations will reflect the performance of its trust model in selecting reliable tenderers for interactions. As a result, the testbed records the UG of each interaction with each trust model used. For purposes of a fair comparison, each model is employed by a large and equal number of agents. Table 5 presents the four trust models and their related population density in the testbed environment. These models are the proposed model (StoRM), DISARM [35], CRM [34] and Certified Reputation [30].

Table 5. Testbed: trust models and population density

Trust model compliance	***Population Density***
StoRM	25%
DISARM	25%
CRM	25%
Certified Reputation	25%

DISARM is a social, distributed, hybrid, rule-based reputation model which uses defeasible logic. It uses rules, combines interaction trust and witness reputation, considering the agents acting in the environment as a social network. Yet, it does not use graphs but a history of personal previous interactions and a set of quite complicated rules in order to create a potential propagation network. Certified Reputation, on the other hand, is a well-known model that asks agents to give ratings of their performance after every transaction while the agents that receive these ratings have to keep them. Hence, each agent that needs ratings is able to ask any other agent for its stored references. However, Certified Reputation is designed to determine the access rights of agents, rather than to determine their expected performance. CRM (Comprehensive

Reputation Model) is a probability-based model that asks agents to keep ratings, both from their direct transactions and witnesses, calling the procedure online trust estimation. Later, the actual performance of an evaluated agent will be compared against the above related ratings, in order to judge the accuracy of the consulting agents in the previous on-line process. This procedure is called off-line.

Therefore, we compare the above-mentioned models using the discussed testbed settings. At this point, we have to mention that we adopted the first case for the agent type distribution (approximately equal) because only StoRM distinguishes the difference between the represented IoT Things. Figure 9 presents a ranking of all four models regarding the mean utility gained (UG value).

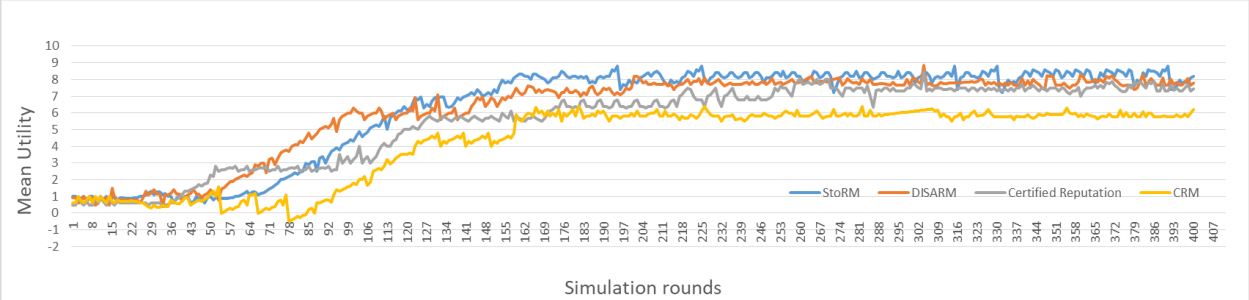

Figure 9. Mean Utility Gained.

This comparison reveals that the three of four, StoRM, DISARM and Certified Reputation, gain a quite high UG value. Yet, none of them reaches the higher values that achieved StoRM (neither itself), as shown in Figures 7 and 8, when it was the only adopted model in the environment. This can be explained by the fact that the percentage of agents that comply with each model is lower leading inevitably to less interactions under each specific model, less social relationships, and loss of rating data. Among the four models, StoRM, DISARM achieve slightly the higher performance, probably due to the fact that they consider the environment as a social network that allows messages to propagate. DISARM enables agents to get familiarized with the environment faster as opposed to the rest but StoRM has a higher upward performance, keeping it stabilized and higher from a time point. This performance reveals that using a combination of social graphs and P2P networks in order to locate partners reveal a dynamic potential. On the other hand, CRM needs time to get familiarized with the environment and remains with a stable but significant lower performance.

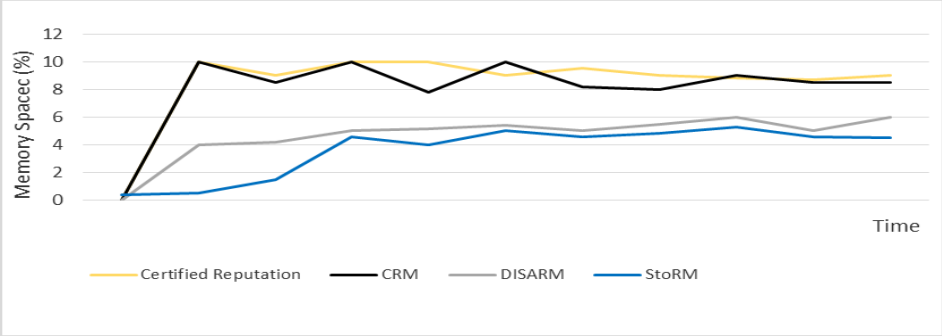

Figure 10. Storage space growth.

Furthermore, in order to study better the performance of the proposed model, another comparison direction was towards execution time and storage space. Distributed models store their own ratings and those obtained by others which is far less than the total number of available ratings in the system. Of course, they can erase extra witness ratings after use but still, they have sometimes significant memory space requirements. Figure 10 depicts the percentage usage of the available space each agent has. It reveals that models like StoRM and DISARM that take into account social aspects need less space. Actually, they collect less ratings whenever they need them, allowing them to use only a part of their space. Finally, StoRM has a stabilized downward usage which make it a good option of IoT modeling cases where there are many entities with space limitations.

6. Related Work

Trust and reputation represent an interesting and active research area. There are already plenty of trust models proposed even for the Internet of Things. Yet, to the best of our knowledge StoRM is one of the first, if not the first, model that combines social trust management in IoT with microservice architecture.

Bao and Chen [7] proposed one of the first trust management protocols considering both social trust and QoS trust metrics and using both direct observations and indirect recommendations to update trust. They used only three trust evaluation parameters, namely honesty, cooperativeness and community-interest opposed to StoRM that proposes a well-studied set available for all kind of Things. On the other, just like StoRM this approach acknowledges the need to take into consideration social relationships in trust management for IoT. Yet, it does not reflect the actual social relations among agents, like StoRM, but rather attempts to focus on

authentication and security issues. Chen et al. [15] proposed a trust management model based on fuzzy reputation for IoT. However, their trust management model considers a specific IoT environment consisting of only wireless sensors with QoS trust metrics only such as packet forwarding/delivery ratio and energy consumption, and does not take into account the social relationship which is important in social IoT systems. Furthermore, StoRM proposed a novel management system that can handle different types of Thing by adopting the architecture of microservices.

CRM (Comprehensive Reputation Model) [34] is a typical and well known distributed reputation model. In CRM the ratings used to assess the trustworthiness of a particular agent can either be obtained from an agent's interaction history or collected from other agents that can provide their suggestions in the form of ratings; namely interaction trust and witness reputation, respectively. CRM is a probabilistic-based model, taking into account the number of interactions between agents, the timely relevance of provided information and the confidence of reporting agents on the provided data. More specifically, CRM, first, takes into account direct interactions among agents, calling the procedure online trust estimation. After a variable interval of time, the actual performance of the evaluated agent is compared against the information provided by other agents in a procedure called off-line. Off-line procedure considers the communicated information to judge the accuracy of the consulting agents in the previous on-line trust assessment process. In other words, in CRM the trust assessment procedure is composed of on-line and off-line evaluation processes. Both CRM and StoRM acknowledge the need for hybrid reputation models taking into account time issues, yet they propose a starkly opposite approach. Additionally, both models use a confidence parameter in order to weight ratings more accurately. However, StoRM takes into account a variety of additional parameters, allowing users to define weights about them. As a result, more accurate and personalized estimations are provided. Furthermore, only StoRM considers the social relations among agents providing an approach that let them establish and maintain trust relationships, locating quite easily reliable ratings.

Finally, DISARM [35], a previous work of us, is a hybrid distributed model that uses defeasible logic. It adopts social aspects, being a nonmonotonic model. DISARM yet uses a large and quite complicated set of rules in order to locate ratings and to combine interaction trust and witness reputation. Both DISARM and StoRM consider the environment as a social network although they use different approaches. StoRM uses social graphs and P2P principles in order to

allow agents to identify how far a party is, revealing a hint about the degree of trust that they can invest in it. Furthermore, it proposes a simpler and lightweight approach for the estimation mechanism while it pays much attention to the locating mechanism. On the other hand, DISARM is mainly based on interaction made by already known agents without acknowledging exactly how far is the interaction distance between a known agent and its recommendation for another unknown. Yet, the both are based on similar, well-established estimation parameters. Hence, DISARM can be adopted in any multi-agent system in the Semantic Web but StoRM is more suitable for modeling IoT cases.

7. Conclusions

This article presented StoRM, a social, distributed and hybrid reputation model that adopts the architecture of microservices for the IoT. It limits the common disadvantages of the existing distributed trust approaches, such as locating ratings, by considering the agents acting in the environment as a social network. Hence, each agent is able to propagate its requests to the rest of the agent community, locating quite fast ratings from previously known and well-rated agents. It is based on well-established estimation parameters, such as information correctness and validity. StoRM can be adopted in any multi-agent system that will represent an IoT environment, including different type of Things (entities, services, devices). Finally, we provided an evaluation simulation that illustrates the usability of the proposed model.

As for future directions, our priority is to study the scalability the complexity of the model as well as the scalability of the multi-agent system. Hence, we will further improve the proposed model, attempting to reduce among others its complexity. To this end, our intention is to further study the performance of our model by comparing it to reputation models from the literature and use it in real-world applications such as smart living. Our aim is to enrich it with a powerful mechanisms that will extract the relationships between potential partners as well as their past and future behavior. Hence, another direction is towards further improving StoRM by adopting more technologies, such as ontologies, machine learning techniques and user identity recognition and management being some of them. For instance, ORDAIN [37], an ontology for trust management in the IoT could be used for this purpose. An ontology is a formal naming and definition of the types, properties, and interrelationships of the entities that really or fundamentally exist for a particular domain of discourse. Over the years, ontologies have become

common in the Web. This is not surprising since ontologies make domain assumptions explicit and clear, enabling information reuse and common understanding of the structure of information. Furthermore, ontologies have the ability to integrate existing ontologies describing portions of a large domain. Hence, formalizing reputation and, as a result trust, into ontologies has several advantages such as creating a common understanding for reputation and enabling mapping between reputation concepts.

Acknowledgment

The Postdoctoral Research was implemented through an IKY scholarship funded by the "Strengthening Post-Academic Researchers / Researchers" Act from the resources of the OP "Human Resources Development, Education and Lifelong Learning" priority axis 6,8,9 and co-funded by The European Social Fund - the ESF and the Greek government.

References

[1] Aggarwal, C. C., Ashish, N., & Sheth, A. (2012). The Internet of Things: A Survey from the Data-Centric Perspective. *Managing and Mining Sensor Data*, 383-428. doi:10.1007/978-1-4614-6309-2_12.

[2] Alioto, M. (2017). *Enabling the Internet of Things: From Integrated Circuits to Integrated Systems*. Cham: Springer International Publishing.

[3] Amin, E., Abouelela, M., & Soliman, A. (2018). The Role of Heterogeneity and the Dynamics of Voluntary Contributions to Public Goods: An Experimental and Agent-Based Simulation Analysis. *Journal of Artificial Societies and Social Simulation*, 21(1). doi:10.18564/jasss.3585

[4] Androutsellis-Theotokis, S., & Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4), 335–371.

[5] Atzori, L., Iera, A., & Morabito, G. (2014). From "smart objects" to "social objects": The next evolutionary step of the internet of things. *IEEE Communications Magazine*, 52(1), 97-105. doi:10.1109/mcom.2014.6710070

[6] Banisch, S., & Olbrich, E. (2017). The Coconut Model with Heterogeneous Strategies and Learning. *Journal of Artificial Societies and Social Simulation*, 20(1). doi:10.18564/jasss.3142

[7] Bao, F., & Chen, I. (2012) Trust management for the Internet of Things and its application to service composition. In: Proceedings of the IEEE International Symposium on World of Wireless, Mobile and Multimedia Networks (WoWMoM), 1–6.

[8] Bellifemine, F., Caire, G., Poggi, A. & Rimassa, G. (2003). JADE: A white Paper. *EXP in search of innovation*, 3(3), 6-19.

[9] Bhowmik, A. K., Khendek, F., Hormati, M., & Glitho, R. (2015). An architecture for M2M enabled social networks. 2015 *14th Annual Mediterranean Ad Hoc Networking Workshop* (MED-HOC-NET). doi:10.1109/medhocnet.2015.7173161

[10] Bordini, R. H. (2014). *Multi-agent programming*. Springer.

[11] Brittes, M. P., Jr., B. S., & Wille, E. C. (2017). Trustworthiness Management Through Social Relationships in Internet of Medical Things. *Journal of Communication and Information Systems*, 32(1), 1-7. doi:10.14209/jcis.2017.1

[12] Burete, R., Badica, A., Badica, C., & Moraru, F. (2011). Enhanced Reputation Model with Forgiveness for E-Business Agents. *Theoretical and Practical Frameworks for Agent-Based Systems*, 147-163. doi:10.4018/978-1-4666-1565-6.ch010

[13] Butzin, B., Golatowski, F., & Timmermann, D. (2016). Microservices approach for the internet of things. 2016 *IEEE 21st International Conference on Emerging Technologies and Factory Automation* (ETFA). doi:10.1109/etfa.2016.7733707

[14] Castelfranchi, C., & Falcone, R. (2010). *Trust Theory: A Socio-Cognitive and Computational Model*, 1st edition, Wiley Series in Agent Technology. Wiley ISBN:13:978-0470028759 .

[15] Chen, C., Helal, S. (2011). A Device-Centric Approach to a Safer Internet of Things. *2011 Int. Workshop on Networking and Object Memories for the Internet of Things*, pp. 1-6.

[16] Cline, P. B. (2015). The Merging of Risk Analysis and Adventure Education. *Wilderness Risk Management*. 5 (1): 43–45.

[17] Dasgupta, P. (2000).Trust as a commodity. Gambetta D. (Ed.). *Trust: Making and Breaking Cooperative Relations*, Blackwell, 49-72.

[18] Fetzer, C. (2016). Building Critical Applications Using Microservices. *IEEE Security & Privacy*, 14(6), 86-89. doi:10.1109/msp.2016.129

[19] Gao, D., Deng, X., Zhao, Q., Zhou, H., & Bai, B. (2015). Multi-Agent Based Simulation of Organizational Routines on Complex Networks. *Journal of Artificial Societies and Social Simulation*, 18(3). doi:10.18564/jasss.2817

[20] Garriga, M. (2018). Towards a Taxonomy of Microservices Architectures. *Software Engineering and Formal Methods Lecture Notes in Computer Science*, 203-218. doi:10.1007/978-3-319-74781-1_15.

[21] Gawinecki, M., Ganzha, M., Kobzdej, P., Paprzycki, M., Badica, C., Scafes, M., Popa, G.G. (2006) Managing Information and Time Flow in an Agent-Based E-Commerce System. *2006 Fifth International Symposium on Parallel and Distributed Computing*, Timisoara, pp. 352-359. doi: 10.1109/ISPDC.2006.32

[22] Gelfond, M., & Kahl, Y. (2014). *Knowledge representation, reasoning, and the design of intelligent agents: the answer-set programming approach*. New York (N.Y.): Cambridge University Press.

[23] Gore, R., Lemos, C., Shults, F. L., & Wildman, W. J. (2018). Forecasting Changes in Religiosity and Existential Security with an Agent-Based Model. *Journal of Artificial Societies and Social Simulation*, 21(1). doi:10.18564/jasss.3596

[24] Gregoire, J., & Nguyen, S. N. (2017). Tell me again, why should i talk to strangers? *2017 20th Conference on Innovations in Clouds, Internet and Networks* (ICIN). doi:10.1109/icin.2017.7899431

[25] Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7), 1645-1660. doi:10.1016/j.future.2013.01.010

[26] Gutowska, A., & Buckley, K. (2008). Computing reputation metric in multi-agent e-commerce reputation system. In: *Proceedings of the 28th International Conference on Distributed Computing Systems*, pp. 255–260.

[27] Harwood, T., & Garry, T. (2017). Internet of Things: understanding trust in techno-service systems. *Journal of Service Management*, 28(3), 442-475. doi:10.1108/josm-11-2016-0299

[28] Hendrikx, F., Bubendorfer, K., & Chard, R. (2015). Reputation systems: A survey and taxonomy. *Journal of Parallel and Distributed Computing*, 75, 184-197. doi:10.1016/j.jpdc.2014.08.004.

[29] Hobbs, R. L., & Dron, W. (2015). Using Intelligent Agents for Social Sensing across Disadvantaged Networks. *2015 IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems*. doi:10.1109/mass.2015.96

[30] Huynh, D, Jennings, N R., & Shadbolt, N R. (2006a). Certified reputation: How an agent can trust a stranger. In AAMAS '06: *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, Hokkaido, Japan.

[31] Huynh, D., Jennings, N R., & Shadbolt, N. R. (2006b). An integrated trust and reputation model for open multi-agent systems. *Journal of Autonomous Agents and Multi-Agent Systems* (AAMAS), 13(2):119-154.

[32] Jiang, W., Wang, G., Bhuiyan, M.Z.A., & Wu, J. (2016). Understanding graph-based trust evaluation in online social networks: methodologies and challenges. *ACM Computing Surveys*, 49(1), 10.

[33] Jøsang, A., Ismail, R., & Boyd, C. (2007). A survey of trust and reputation systems for online service provision. Decision Support Systems, 43(2), 618-644. doi:10.1016/j.dss.2005.05.019.

[34] Khosravifar, B., Bentahar, J., Gomrokchi, M., & Alam, R. (2012). CRM: An efficient trust and reputation model for agent computing. *Kowledge-Based Systems*, 30:1-16.

[35] Kravari, K. & Bassiliades N. (2016) DISARM: A Social Distributed Agent Reputation Model based on Defeasible Logic. *Journal of Systems and Software*, 117: 130-152.

[36] Kravari, K. & Bassiliades, N. (2017). Social principles in agent-based trust management for the Internet of Things. Presented at *14th Workshop on Agents for Complex Systems (ACSys 2017) in the framework of 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing* (SYNASC'17).

[37] Kravari, K. & Bassiliades, N. (2017b). ORDAIN: An ontology for trust management in the internet of things: (Short paper). *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10574 LNCS, 216-223.

[38] Kravari, K., Bassiliades, N. & Boley, H. (2012). Cross-Community Interoperation Between Knowledge-Based Multi-Agent Systems: A Study on EMERALD and Rule Responder. *Journal of Expert Systems With Applications*, 39(10), 9571-9587.

[39] Kravari, K., Kontopoulos, E. & Bassiliades, N. (2010a). EMERALD: A Multi-Agent System for Knowledge-based Reasoning Interoperability in the Semantic Web. *6th Hellenic Conference on Artificial Intelligence (SETN 2010)*, Springer Berlin / Heidelberg, LNCS, 6040/2010, 173-182.

[40] Kravari, K., Kontopoulos, E., & Bassiliades, N. (2010b). Trusted reasoning services for semantic web agents. *Informatica: International journal of computing and informatics*, 34(4), 429-440.

[41] Kubera, Y., Mathieu, P. & Picault, S. (2010), Everything can be Agent!. *Proceedings of the ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems* (AAMAS'2010), 1547–1548.

[42] Kuo, C., & Chang, S. E. (2016). Web services-based trust framework design and applications: A case study. *2016 Eighth International Conference on Ubiquitous and Future Networks* (ICUFN). doi:10.1109/icufn.2016.7537157

[43] Lee, I., & Lee, K. (2015). The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Business Horizons*, 58(4), 431-440. doi:10.1016/j.bushor.2015.03.008.

[44] Li, S., Da Xu, L., Zhao, S. (2015). The internet of things: a survey. *Information Systems Frontiers*, 17(2), 243-259.

[45] Ma, M., Wang, P., & Chu, C. (2013). Data Management for Internet of Things: Challenges, Approaches and Opportunities. *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, 1144-1151. doi:10.1109/greencom-ithings-cpscom.2013.199.

[46] Majd, E., & Balakrishnan, V. (2016). A reputation-oriented trust model for multi-agent environments. *Industrial Management & Data Systems*, 116(7), 1380-1396. doi:10.1108/imds-06-2015-0256

[47] Mukhopadhyay, S. C., & Suryadevara, N. K. (2014). Internet of Things: Challenges and Opportunities. *Internet of Things Smart Sensors, Measurement and Instrumentation*, 1-17. doi:10.1007/978-3-319-04223-7_1.

[48] Nadareishvili, I., Mitra, R., McLarty, M., & Amundsen, M. (2016). Microservice Architecture. O'Reilly Media, Inc., 1st edn.

[49] Nguyen, T. D., & Bai, Q. (2017). Enhance Trust Management in Composite Services with Indirect Ratings. *The Computer Journal*, 60(11), 1619-1632. doi:10.1093/comjnl/bxx026

[50] Niu, J., Jin, Y., Lee, A. J., Sandhu, R., Xu, W., & Zhang, X. (2016). Panel Security and Privacy in the Age of Internet of Things. *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies - SACMAT 16*, 49-50. doi:10.1145/2914642.2927920

[51] Pandey, A.K., Vasishtha, A.K., & Saxena, A.S. (2016) Properties and interaction of object oriented software agent with system. In: *2016 3rd International Conference on Computing for Sustainable Global Development* (INDIACom). pp. 1141-1143

[52] Poslad, S. (2007). Specifying Protocols for Multi-agent System Interaction. ACM *Transactions on Autonomous and Adaptive Systems*, 4 (4): 15–es. doi:10.1145/1293731.1293735.

[53] Rajasree, S., & Elizabeth, B. (2016). Trust Based Cloud Service Tenderer Selection. *International Journal of Engineering and Computer Science*. doi:10.18535/ijecs/v5i5.63.

[54] Ruan, Y., Durresi, A., & Alfantoukh, L. (2016). Trust Management Framework for Internet of Things. 2016 *IEEE 30th International Conference on Advanced Information Networking and Applications* (AINA). doi:10.1109/aina.2016.136

[55] Salah, T., Zemerly, M. J., Yeun, C. Y., Al-Qutayri, M., & Al-Hammadi, Y. (2016). The evolution of distributed systems towards microservices architecture. 2016 *11th International Conference for Internet Technology and Secured Transactions* (ICITST). doi:10.1109/icitst.2016.7856721.

[56] Son, H., Kang, N., Gwak, B., & Lee, D. (2017). An adaptive IoT trust estimation scheme combining interaction history and stereotypical reputation. *2017 14th IEEE Annual Seeker Communications & Networking Conference* (CCNC). doi:10.1109/ccnc.2017.7983132

[57] Talia, D. (2014). Towards Internet Intelligent Services Based on Cloud Computing and Multi-Agents. *Advances in Intelligent Systems and Computing Advances onto the Internet of Things*, 271-283. doi:10.1007/978-3-319-03992-3_19

[58] Truong, N. B., Lee, H., Askwith, B., & Lee, G. M. (2017). Toward a Trust Evaluation Mechanism in the Social Internet of Things. *Sensors*, 17(12), 1346. doi:10.3390/s17061346

[59] Vermesan, O., & Friess, P. (2015). *Internet of things - from research and innovation to market deployment*. Aalborg: River .

[60] Wahab, O. A., Bentahar, J., Otrok, H., & Mourad, A. (2015). A survey on trust and reputation models for Web services: Single, composite, and communities. *Decision Support Systems*, 74, 121-134. doi:10.1016/j.dss.2015.04.009

[61] Whitmore, A., Agarwal, A., & Xu, L. D. (2014). The Internet of Things—A survey of topics and trends. *Information Systems Frontiers*, 17(2), 261-274. doi:10.1007/s10796-014-9489-2.

[62] Xianyu, B. (2010). Social Preference, Incomplete Information, and the Evolution of Ultimatum Game in the Small World Networks: An Agent-Based Approach. *Journal of Artificial Societies and Social Simulation*, 13(2). doi:10.18564/jasss.1534

[63] Xiao, H., Sidhu, N., & Christianson, B. (2015). Guarantor and reputation based trust model for Social Internet of Things. *2015 International Wireless Communications and Mobile Computing Conference* (IWCMC). doi:10.1109/iwcmc.2015.7289151

[64] Xu, L. D., He, W., & Li, S. (2014). Internet of Things in Industries: A Survey. *IEEE Transactions on Industrial Informatics*, 10(4), 2233-2243. doi:10.1109/tii.2014.2300753.