Hindawi
Wireless Communications and Mobile Computing
Volume 2017, Article ID 7254181, 9 pages
https://doi.org/10.1155/2017/7254181

Research Article

ABS-SmartPriority: An Agent-Based Simulator of Strategies for Managing Self-Reported Priorities in Smart Cities

Iván García-Magariño[1,2] and Raquel Lacuesta[1,2]

[1]*Department of Computer Science and Engineering of Systems, University of Zaragoza, Escuela Universitaria Politécnica de Teruel, c/Atarazana 2, 44003 Teruel, Spain*
[2]*Instituto de Investigación Sanitaria Aragón, University of Zaragoza, Zaragoza, Spain*

Correspondence should be addressed to Iván García-Magariño; ivangmg@unizar.es

Received 29 August 2017; Accepted 25 October 2017; Published 10 December 2017

Academic Editor: Syed Hassan Ahmed

Smart cities still need the appropriate tools for allowing researchers to contribute in this growing field that will change the comfort and qualities of live of citizens. Smart cities can provide services such as informing of the less overcrowded tourist routes, path-finding for avoiding traffic jams, search services for parking, collecting tree branches from streets, and the finding of the nearest available public electric bicycles or cars. The levels of urgencies differ from some situations to others. Examples of urgent matters are the fast transportation critical patients and the removal of obstacles in main roads. Since smart cities are meant to manage huge amounts of requests, priorities should be automated and managed in a flexible way for new scenarios. Smart cities may need citizens to report the service priorities. However, citizens may have different criteria and could abuse the highest priority levels hindering the service performance for really urgent matters. A novel agent-based simulation open-source framework is proposed for testing different policies for normalizing and controlling self-reported priorities, with its simulator called ABS-SmartPriority. This approach is illustrated by simulating two different policies, in which the smart policy outperformed the one used as the control mechanism.

1. Introduction

One of the main goals of smart cities is to improve the quality of service (QoS) of citizens [1]. In smart cities, information communication technologies (ICT) allow citizens to use more efficiently the existing resources. In particular, network is the main necessity for the communication of the devices with Internet of Things (IoT) and the different kind of sensors [2]. Among others, some examples of sensors are waste collection sensors [3], conductivity sensors for monitoring groundwater resources [4], and radio frequency identification (RFID) sensors for identifying users of smart cities [5].

The wireless sensor networks (WSNs) can be appropriate for supporting communications in smart cities thanks to their low-cost and their easily adaptive configuration to different city circumstances. However, maintaining QoS with WSNs is challenging due to the power constraints, radio frequency interferences, noise, and multipath finding. Yigit et al. [6] proposed a priority mechanism for addressing these challenges.

In the management of city services, a priority mechanism can make it possible that urgent matters are attended faster. For example, Anagnostopoulos et al. [7] used priorities for determining how to collect waste in smart cities, prioritizing the areas that can have potentially risky waste such as hospital areas. Another example is the way patients are commonly attended every day in a hospital. When a patient gets to the hospital for the emergency department, a healthcare professional almost immediately performs a really quick examination to determine the actual urgency of the matter to establish a priority level [8]. This saves lives, as life-critical matters are attended with the highest priority. This priority policy works properly thanks to the existence of a common individual that examines objectively all the cases and assigns priorities in a fair way.

Nevertheless, the smart cities show a steep exponential growth of data and communications with imprecise information [2]. This amount of information does not make it possible to have independent individuals to fairly assign the priorities for every service. Then, the assignment of priorities may need to be indicated by the citizens of the smart city or its mobile applications. However, the self-assigned priorities would require a great level of responsibility from behalf of citizens. The Prisoner's Dilemma reveals that the self-assigned priority in an anonymous way (in the sense that nobody notices the behavior of each citizen) may be useless. Most citizens could request the highest level of priority (priority zero) for all the requests of services regardless their actual urgency. Martínez-Cánovas et al. [9] showed that cooperation is promoted when users repeat the decisions knowing that these are tracked and can be rewarded (or penalized).

The current work belongs to the trend of works about providing testbeds for analyzing and simulating different scenarios of smart cities. For example, Sanchez et al. [10] provided a testbed for simulating and analyzing different strategies for managing the information from IoT devices in smart cities. Although their testbed was initially inspired by Santander city, their testbed is applicable to many situations of smart cities in general. In addition, the current work is also in the line of research about scheduling services in smart cities. For instance, Anagnostopoulos et al. [11] proposed a scheduling mechanism for the waste collection in smart cities. They analyzed and tested their approach using data from the St. Petersburg city, and they presented an Android app for evaluating the feasibility of their approach.

In this scenario, multiagent systems (MASs) allow developers to implement systems with several autonomous entities for testing behaviors of societies composed of several individuals. For example, MASs have been shown to be useful for studying (a) the coordination of citizens in urban crisis situations [12] and (b) the coordination a group of experts for reaching a consensus with the Delphi process [13]. More concretely, agent-based simulators (ABSs) are a specific kind of MASs that is aimed at analyzing different circumstances by simulations. For instance, the ABS of Hassan et al. [14] simulated the friendship and partner relationships taking proximity and similarity into account with fuzzy techniques. In addition, the ABS for Tourist Urban Routes (ABSTUR) [15] showed its utility for analyzing the emergent behavior of tourists visiting a city based on the recommendations of certain routes. Furthermore, Garcia-Sabater et al. [16] proposed an ABS to anticipate the demands of manufacturing networks.

In this context, we present a novel open-source ABS and its underlying framework, both of them called ABS-SmartPriority, as a solution to assess and compare different self-reported priority strategies in smart city societies. This approach allows researchers to explore new ways of fairly managing the resources and networks of smart cities considering self-reported priorities.

2. Materials and Methods

2.1. ABS-SmartPriority. This work uses the novel ABS called ABS-SmartPriority for simulating the management of resources of smart cities based on self-reported priorities. This can simulate priorities reported by citizens, city workers, or mobile applications on behalf of their users. This ABS is developed with the Process for developing Efficient ABS (PEABS) [17]. In order to ensure the reproducibility of the experiments, the source code of the underlying framework has been made available as supplementary material of the current article [18]. This open-source code may allow other researches to extend the existing framework or understand better the underlying implementation. The framework was programmed with the C# language and the Unity engine. We selected Unity because (a) it is a cross-platform engine that allows deploying applications in many platforms such as Windows, Mac, Android, and iOS and (b) it facilitates the programming of visual and interactive applications. From the two programming languages supported by Unity, we chose C# since its object-oriented nature facilitated the straightforward application of PEABS for implementing an efficient ABS. In addition, agents are normally implemented as objects in object-oriented programming in the literature [19].

The user can assign priorities in a range (in our case 0 to 2), in which zero represents the highest priority and two the lowest one. However, the framework could be easily adapted to represent different priority ranges. In the real world, the user would be a person that assigns this level of priority for any request of service or resource. In fact, the user could set default priorities for certain activities through their mobile applications for not being annoyed in selecting the priority in most cases. However, since this work is intended to simulate different behaviors, each person is impersonated by an agent, which will be referred to as "user agent" from this point forward. In a wider contest, the user can be a citizen, a company with devices with IoT, an organization, a city worker, a mobile app on behalf of a person, or more generally any entity that has to share a service with other peers in a smart city. In the real world, users are supposed to be told to use always the lowest priority, unless there were really unusual and urgent situations. In this case, they could use several high priority levels, but they should only use the highest one in life-death situations or the ones that could imply serious health risks.

The framework allows researchers to define different kinds of user agents by implementing the operations about (a) deciding when to ask a service and (b) selecting the priority. We defined two kinds of users for illustrating the current approach. The selfish user agents selected the highest priority. The default honest user agents mostly select the lowest priority except in a few cases that represent actual urgent matters. Both user agent kinds applied the mechanism for taking nondeterministic decisions proposed by TABSAOND (a technique for developing ABS apps and online tools with nondeterministic decisions) [20].

In each service request, an agent transforms the priority self-reported by the user into a priority that can be fairly compared with the priorities of other users. This agent would be actually software in the real world, and this approach simulates this with software agents called "priority normalizer agents" (also referred to as "priority agents"). Each priority

agent will be assigned to each citizen or any other kind of entity that is represented by the user agent. The priority normalizer agent is the one that would be extended to test different self-reported policies, and this can register all the previous priorities requested by its corresponding user.

The "resource manager agent" (also denoted as "manager agent") is in charge of assigning the services faster to the ones with the highest priorities. In order to avoid starvation of some users, the manager agent has to continuously attend all the requests, even if the higher priority ones are attended faster. For this purpose, this agent has several queues. Here, we illustrate the approach with three queues due to the selected number of priority levels, but it could be generalized for a different number of queues.

The next subsections further describe the most relevant aspects of respectively each kind of agents. In addition, Section 2.1.4 introduces the user interface (UI) of the simulator.

2.1.1. Behaviors of User Agents. In order to facilitate the definition of different user agents, firstly we defined a basic user agent called "stochastic user agent" that takes nondeterministic decisions about when to ask the services and which priority to report. This agent takes these decisions based on probabilities for the different cases, by applying TABSAOND. This agent was implemented to serve as a common basis for defining usual user behaviors in a simple way by just changing certain probabilities.

Regarding the decisions about when asking services, we defined a threshold that determines a timeline spot in which the agent is considered to go from novel user to experienced user. It defines different probabilities for deciding whether to ask the service or not, regarding whether it is a novel or an experienced user. Regarding the probability, in each simulated iteration it takes the decision with the following formula:

$$d_a = \begin{cases} \text{yes}, & \text{if } r \leq p_{a,e} \\ \text{no}, & \text{otherwise}, \end{cases} \tag{1}$$

where d_a is the decision about whether to ask a service, r is a random number in the interval $[0, 1]$, generated by a generic random number generator library of Unity engine, and $p_{a,e}$ is the probability of asking a service regarding the e experience of use (which can be either beginner or experienced user).

The calculation of $p_{a,e}$ is done by firstly distinguishing whether the user is a beginner or not by comparing their time using the service with a certain threshold. Then, $p_{a,e}$ is set to a specific probability value if the user is a beginner or is set to a different value otherwise. The selection of these two probability values can be established in a calibration phase of the system, for example, following the technique for Automatically Training ABS (ATABS) [21].

The beginner user is normally assumed to ask for more services, while experienced users may need a less frequent use of the service. In addition, with this model, the system can use a high amount of requests filling the queues of the different priority levels. Then, the users ask for a normal amount of requests, but with the queues already filled. In this way, we can properly compare the different priority policies.

After this, if the user agent requests a service, it selects the priority based on a distribution of probabilities. As a particular case of the suggestions of TABSAOND, it applies the following formula for the three priority levels, although this could be extended for a higher number of levels:

$$d_p = \begin{cases} 0, & \text{if } r \leq p_{p,0} \\ 1, & \text{if } p_{p,0} < r \leq \left(p_{p,0} + p_{p,1}\right) \\ 2, & \text{otherwise}, \end{cases} \tag{2}$$

where d_p is the decision of which priority level is assigned, r has the same meaning as in the previous equation, and $p_{p,0}$ and $p_{p,1}$ are the probabilities of respectively selecting the zero and one levels of priority. The probability of selecting the two levels (i.e., the lowest one) is the subtraction of these probabilities from one. The $p_{p,0}$ and $p_{p,1}$ probability values are normally calibrated when conforming the desired behavior of a specific kind of user agents.

The difference between the honest and the selfish agent is that the former asks the high priority levels with frequencies when they actually needed and the selfish agent mainly asks the highest level. In particular, the honest user is represented with the default stochastic user agent in which the probabilities are $p_{p,0} = 10\%$ and $p_{p,1} = 10\%$. The selfish user agent is defined as an extension of the default stochastic user agent in which the probabilities are $p_{p,0} = 98\%$ and $p_{p,1} = 1\%$.

Different kinds of selfish agents could be implemented for testing new security attacks and misuses in the requests of services. For example, new kinds of selfish agents can be implemented by extending the user agent class and implementing the "Live" method. For example, the denial of service (DoS) or the distributed DoS attacks could be implemented by calling many times to the "Ask Service" method from the Live method with the corresponding loop.

2.1.2. Normalization of Priority Agents. In the current approach, the normalizer priority agents represent a logical mandatory layer of the communication for requesting services in smart cities with self-reported priorities. These priority agents are in charge of adapting self-reported priorities into normalized priorities that can be meaningfully compared between different users.

In the current framework, one can easily define new strategies for normalizing self-reported priorities, by extending the priority agent (i.e., by extending the corresponding object-oriented class), and overriding just the "Normalize" method.

In order to associate the priorities and the users that reported these, each priority agent is in charge of managing only the petitions of one user agent. In this way, the priority agent can record its activities in their fields (i.e., object-oriented attributes) to consider the history of each user agent. Since the framework is implemented with PEABS, the priority agents could even communicate each other by accessing all the agents of simulation of the corresponding type.

To illustrate the current approach, we have defined two strategies, denoted, respectively, as "smart" and "simple" from this point forward. The smart strategy is based on trusting the self-reported priorities of users at the beginning, but it also records the history of the priorities that each user has assigned to its requests. Each priority agent calculates the average of the priorities requested by the user. If the average surpasses or is equal to certain value (i.e., most of the requests had the lowest priority), then it trusts their self-reported priority and forwards the petition with this priority. If the average priority is below that threshold (i.e., the agent assigns high priorities frequently), then the priority agent interprets that the user is misusing the priority system, and it forwards its petition with the lowest priority level. This can be formalized with the following formula:

$$x_N = \begin{cases} x_{\max}, & \overline{X_h} >= x_t \\ x_{\text{in}}, & \text{otherwise,} \end{cases} \quad (3)$$

where x_N is the outputted normalized priority, x_t is the threshold for average priority in the history, X_h is the history of priorities, $x_{\max}$ is the maximum number that represents the lowest priority level, and x_{in} is the priority reported by the user agent.

Since the framework is prepared to support different numbers of self-reported priority levels, the threshold is calculated from another inverted threshold with the following formula:

$$x_t = x_{\max} - x_{t,i}, \quad (4)$$

where $x_{t,i}$ is the original inverted threshold and x_t and $x_{\max}$ have the same meaning as before.

It is worth mentioning that the proposed ABS could be easily adapted to support other ranges of priorities. In particular, the internal parameter "max priority" is represented as a property of the "Params" class in the source code and can be set to any value. For example, it could be set to 5 for representing a 5-star system. The range of normalized priorities can be established with the "max norm. priority" property of the same class. The inversion of the priorities for setting high priorities with high numbers (e.g., for being 5 stars the maximum priority) can be easily implemented by changing the order of the corresponding loop in the Live method of the manager agent.

2.1.3. Resource Manager Agent. The resource manager agent is in charge of managing the requests of users, for attending them considering both their chronological order and their priorities. It uses the priorities normalized by the corresponding strategy mechanism through the normalizer priority agents.

The resource manager agent has a queue for each priority level. Each request is added to the queue of its priority level. The request is also added to the queues of the lower priority levels, to guarantee that a request is not attended later than any subsequent request of a lower level. Later, when a request is attended, this is removed from all the queues in which it was added.

In order to avoid the starvation of the requests of the lower priorities, all the queues are attended in all the simulation iterations. However, to guarantee a lower waiting time in urgent requests, the number of attended requests is different for each priority level, and these amounts are configurable. For example, in the current experiments, in each loop, the manager attended three requests from the zero-level queue, two requests from the one-level queue, and just one from the two-level queue, if all the queues had enough requests.

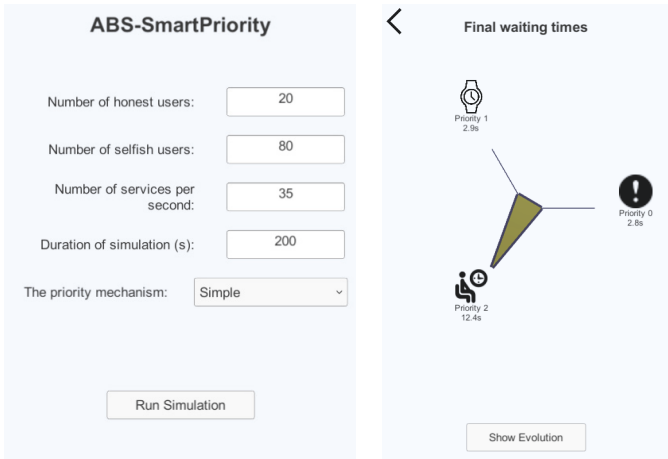
The resource manager agent knows the number of services that it can assign in each simulation iteration, due to the existing limited resources for attending the corresponding service in a smart city. Thus, in each iteration, it continues looping attending requests in the mentioned order, until the number of available assignments is reached or all queues are empty. Notice that if the queue of a priority level is empty, then the possible number of assignments is distributed among the other different priority levels.

2.1.4. User Interface. The UI of ABS-SmartPriority allows users to configure simulations for testing the different techniques for managing self-reported priorities in smart cities. Figure 1(a) shows the screen of the UI for entering the input parameters of the simulations. First, the user can introduce the numbers of respectively the honest users and the selfish users. The input screen also allows one to set the speed in which requests can be attended. This is set as the number of service requests that can be attended per second (each second is simulated with an iteration). The users of this app can also set the duration of the time that is simulated. Finally, the users can select a priority mechanism from a dropdown list in the UI. The user can press the "Run Simulation" button to start the simulation.

Once the simulation is executed, the application shows the final results with a starplot like the one of the execution example of Figure 1(b). Each end of the starplot is associated with a priority level that is shown with a label. An image is also displayed for each priority level to facilitate the understanding of the final results at a glance. Each end represents the average time that a fair observer user agent has waited when requesting services with each priority level.

The user can also press on the "Show Evolution" button to observe the evolution of average waiting times. The chart of this screen uses a different color for marking the waiting time of each priority level. The abscissa axis represents the simulated time, while the ordinates axis represents the waiting time. Section 3 will present examples of simulation evolutions.

2.2. Experimental Method. One of the main goals of the experimentation was to simulate different scenarios and strategies with the presented ABS and its framework, to determine whether this approach is useful for testing different strategies for fairly managing self-reported priorities. Another goal was to compare different ways of managing self-reported priorities in smart cities, determining whether the current approach can find statistically significant differences in their outcomes.



(a) Screen for the input parameters

(b) Screen with final simulation results

Figure 1: UI of ABS-SmartPriority.

Firstly, we defined two different kinds of citizens with ABS-SmartPriority. The first kind of citizens used the priority service properly assigning the lowest level of priority to the common operations and only used the high levels of priorities for urgent matters. The second kind of citizens was programmed to behave as selfish and tried to unfairly take advantage of the priority system by assigning a high priority in most requests including the ones related to common and daily situations.

Two strategies were defined for managing the self-reported priorities. One always trusted the self-reported priorities and forwarded them to the system. This strategy was used as the control mechanism of this experiment. The second strategy kept track of the users, controlling and normalizing the self-reported priorities by assigning the lowest priority level to the users that had abused the system. This abuse was detected from the excessive use of the high priority levels in their past history.

An observer agent was included to monitor the simulation. This agent impersonated an honest citizen who commonly asked low-priority requests except for infrequent urgent matters. This agent recorded the times that it waited for getting the different services for each priority level.

The ABS was executed respectively with the different strategies for managing the self-reported priorities. The results were compared to determine which policy provides a faster response for attending honest users.

The system was executed 100 times for each strategy to avoid bias due to the nondeterministic behaviors. The results were compared with a statistical test and the difference was measured with the corresponding effect size.

3. Results and Discussions

In the experimentation, we compared the different mechanisms for managing self-reported priorities, which are referred to respectively as the simple and smart strategies.

Table 1: Input parameters used in the experimentation.

Parameter	Value
Number of honest users	20
Number of selfish users	80
Number of services attended per second	35
Duration of the simulation (s)	200

ABS-SmartPriority was executed with same input parameters for each priority mechanism, and Table 1 shows these common input parameters.

In the system, we introduced an observer user agent that asked services with different priorities using a common honest priority probability distribution. This agent recorded all the times in which their service requests were attended. The simulator uses the average response times of these requests as the indicators of the QoS considering different priorities.

As mentioned in the experimental method, the simulator was executed 100 times for each priority mechanism. Table 2 presents the means of these simulations with the standard deviations (SD) between parentheses. This table also includes the differences of means and the percentage reductions. The latter ones are the percentages that the differences represent with respect to the value of the control mechanism (i.e., the simple strategy). One can observe that the smart technique reduces considerably the waiting time in the highest priority levels (i.e., levels zero and one) with reductions over 93%. In the lowest priority level, it achieves also reductions of waiting times with a reduction about 55%. Thus, this technique improves mostly the reduction of urgent requests, but it also reduces the waiting time of the daily requests, from the perspective of honest users.

This analysis has been performed from the perception and measurement of the experience of a honest user (i.e., the observer agent). It is worth mentioning that the reduction

Table 2: Comparison of waiting time means for the different priority levels between the different priority techniques.

	Priority 0	Priority 1	Priority 2
Simple waiting time (s)	3.218 (0.742)	3.004 (0.653)	10.610 (1.772)
Smart waiting time (s)	0.213 (0.500)	0.161 (0.376)	4.739 (0.634)
Diff. means (s)	3.005	2.843	5.871
Percentage reduction (%)	93.37	94.65	55.34

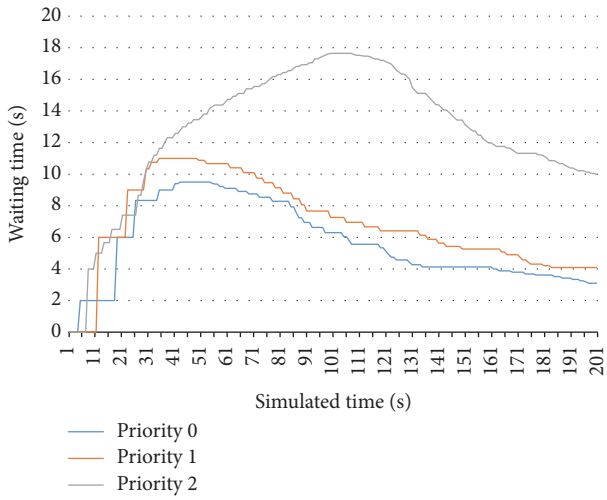


Figure 2: Results with the simple priority mechanism.

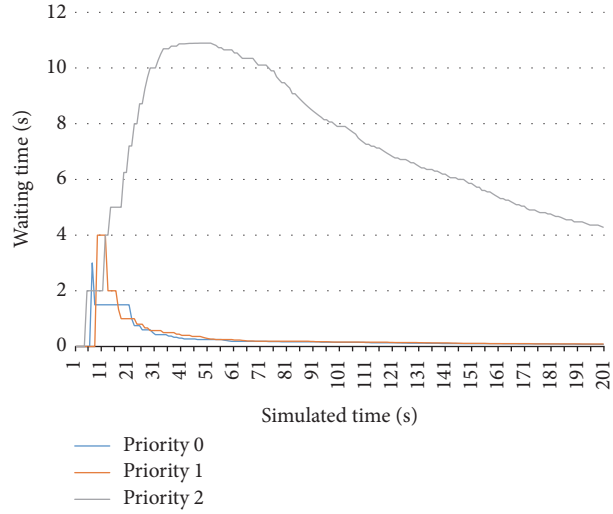


Figure 3: Results with the smart priority mechanism.

has been achieved by normalizing the self-reported priorities of selfish user agent, reassigning their priorities. Thus, the selfish users must wait as much as the other agents, letting the possibility of attending fast to actually urgent requests. From the selfish users' viewpoint, with the smart priority mechanism, they cannot take advantage of the priority system to be attended always fast with the highest priority. This is the reason why all the waiting times were reduced for honest users, although all the experiments used the same speed for providing smart city services.

Figure 2 shows the simulation outcomes of one of the executions with the simple priority mechanism, by showing its evolution. One can observe that the lowest priority level (i.e., priority 2) obtained higher response times (in average 12.8 s) than the other two priority levels (priorities 1 and 0 with, respectively, averages of 6.9 s and 5.7 s). This difference can also be noticed in the maximum waiting times. In the case of the lowest priority level the maximum was 17.7 s, while the higher levels spent at most 11.0 s and 9.5 s waiting for, respectively, priorities 1 and 0. In this way, one can observe that the resource manager agent worked properly, as even with the simple priority policy and with most of the petitions with the highest priority, the queues worked and maintained the highest priority levels with lower response times than the lowest priority.

Though, the highest priority level was still not so fast with an average of 5.7 s and a SD of 2.5 s, probably due to the misuse of the highest priority level on behalf of the selfish user agents, which was not controlled by the simple strategy.

In the evolution chart, one can also observe the intended effect of the user agents starting with high amounts of requests and then after a while lowering these amounts. This was reflected in the steep increase of waiting times, and then the later slow decrease of these.

Figure 3 shows an example of the results of a simulation of the smart strategy for managing self-reported priorities, by presenting the simulation evolution. The resource agent still maintained the highest priority levels with lower waiting times (in both cases with an average of 0.3 s) in comparison to the lowest priority (with an average of 7.1 s). This difference is also reflected in the maximum waiting times, which was 10.9 s for the lowest priority and 4.0 s and 3.0 s for, respectively, priorities 1 and 0. However, the difference was that the smart strategy detected the misuse of the highest priority levels on behalf of the selfish user agents and did not let them abuse these levels. In this manner, in the few times that the honest observer agent needed an urgent service, it got the service much faster. This is confirmed by the much lower response times of priority levels 1 and 0, which were 0.3 s in both cases in comparison to the simple strategy (previously shown in Figure 2) that obtained, respectively, 6.9 s and 5.7 s. At the end of the simulation, the queues were emptied and its use could be considered as the normal one. At the end of the simulation, one can observe that the two highest priorities were almost immediately attended, with waiting times of 0.08 s and 0.09 s, in comparison to the lowest priority level with a waiting time of 4.3 s.

Table 3: Robust tests of equality of means.

	Statistic[a]	df1	df2	Sig.
Priority 0				
Welch	1127.361	1	173.420	.000
Brown-Forsythe	1127.361	1	173.420	.000
Priority 1				
Welch	1423.749	1	158.261	.000
Brown-Forsythe	1423.749	1	158.261	.000
Priority 2				
Welch	973.528	1	123.955	.000
Brown-Forsythe	973.528	1	123.955	.000

[a] Asymptotically F distributed.

Table 4: Cohen's d effect size.

	Priority 0	Priority 1	Priority 2
Cohen's d	4.75	5.34	4.41

We applied Welch's t-test to determine whether the differences of means were statistically significant considering the 100 simulations of each strategy. We selected this test as it is robust for unequal variances. We also applied the Brown-Forsythe test for testing the equality of means, as this test is also robust for unequal variances. Table 3 shows the results of both tests. As one can observe in the results, both tests determined that the differences of means were very significant (considering a .001 significance value) for all the priority levels.

In order to meaningfully measure the differences between the waiting time means, we calculated Cohen's d effect sizes between the simple and smart strategies for each priority level. Table 4 shows the resulting Cohen's d effect sizes. According to Cohen's guidelines [22] and the later interpretations proposed by Rosenthal [23], all the effect sizes can be considered very large, as they all surpassed the threshold of 1.3.

Figure 4 visually shows the differences of waiting times in the highest priority level (i.e., the level zero) with a boxplot. One can observe that the smart priority mechanism substantially decreased the means, considering that these differences were much higher than the variances. Though, the smart priority mechanism also presented some outliers, probably due to the actual coincidences of urgent requests from different honest user agents. On the contrary, the simple mechanism strategy did not present outliers but the mean was quite higher. The reason probably was that the services were overloaded with all the high priority requests of selfish user agents and the users waited a regular high amount of time.

Figure 5 graphically presents the differences of waiting times in the one-level priority. In this case, the smart priority strategy also greatly reduced the means considering the variances. The existence of outliers only in the smart strategy may be due to a similar reason as in the previous case.

Finally, Figure 6 shows the boxplot for the lowest priority level. It is worth mentioning that in the viewpoint of honest users the waiting time reduced also in this level in comparing

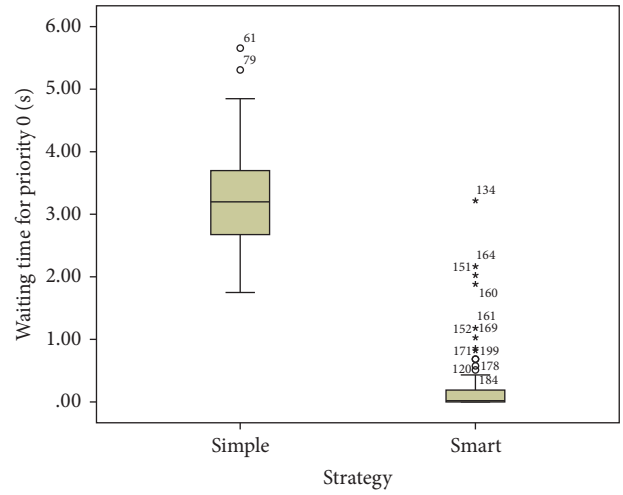


Figure 4: Comparison of waiting times of priority 0 between the simple and smart strategies.

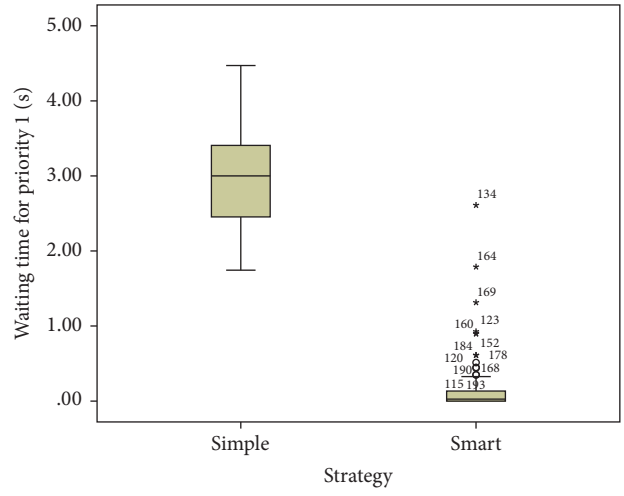


Figure 5: Comparison of waiting times of priority 1 between the simple and smart strategies.

when using the simple strategy. The reason is that the smart strategy stops the abuse of the high priority level. In the simple strategy, three selfish agents were normally attended every time an honest user was attended, since the former ones commonly used the highest priority level. This hindered the fair waiting time in the lowest level. In this boxplot, one can observe that the smart priority did not present outliers, probably since the requests of services of low-priority were more regular. Remember that honest agents mostly asked low-priority services and that the requests of selfish user agents were normalized as low-priority ones by the smart priority agents.

On the whole, ABS-SmartPriority has allowed us to properly simulate different self-reported priority policies that presented statistically significant differences in the outcomes.

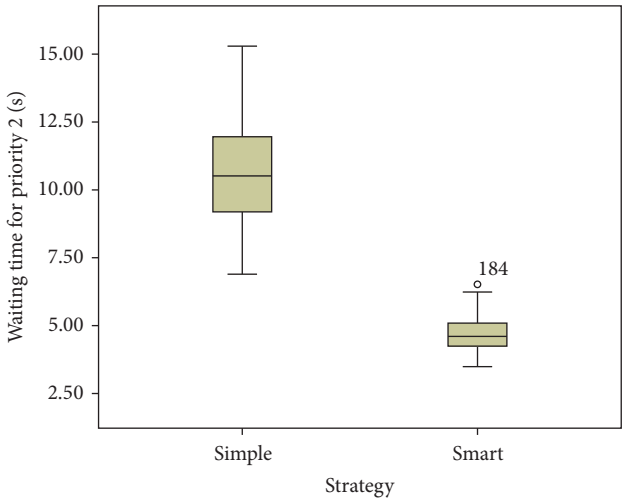


Figure 6: Comparison of waiting times of priority 2 when using a smart technique for controlling self-reported priorities.

The main contribution of the current work over works about priority mechanisms like the ones about waste collection [7] and hospital management [8] is that the proposed approach explores the management of self-report priorities by people with heterogeneous criteria and agendas in a nearly fair way. In addition, the current work improves the testbeds about smart priorities such as the one of Sanchez et al. [10] by incorporating the concept of self-reported priorities for increasing the performance of the use of smart city resources and services. Finally, the current work improves the previous works about respectively measuring the communication of MASs [24] and the integration of web services in agent-oriented methodologies [25] by presenting a MAS that manages the service requests considering priorities normalized from freely self-reported priorities of users.

4. Conclusions

The current work has presented an ABS for analyzing and comparing the repercussions of different strategies for managing self-reported priorities in smart cities. This simulator was illustrated by comparing two different strategies. The first one was based on tracking the history of each citizen to prevent them from unfairly overusing the high priority levels. The second one was a control mechanism that just used the priorities self-reported by the users. The experiments showed that the presented ABS-SmartPriority simulator and its underlying framework were useful to compare these strategies and detect statistically significant differences in the outcomes between these strategies. Therefore, the presented ABS can assist smart city engineers in preparing adequate priority protocols for managing the priorities self-reported by citizens through their apps and devices with IoT in order to fairly manage the smart city services.

In the future, this work is planned to be extended by experiencing a larger number of strategies for managing self-reported priorities. It will also be extended for defining strategies that manage several kinds of smart city services in different ways. We also plan to enhance this work by incorporating mechanisms that use trust and reputation techniques to interchange the reliabilities of the users that self-report their priorities. Furthermore, in the future the current approach will incorporate a feedback system in which the veracity of the reported emergencies may be checked after being attended.

The UI of the ABS is planned to incorporate more parameters about the experience of smart city users and their behaviors. In addition, the simulator will consider more features of users for determining their frequencies in requesting certain services. For example, more user agent types will be implemented with parameters such as the age of users, which is normally related to the need of healthcare services. The UI will be extended to configure these new user agent types with their parameters for the simulations.

Conflicts of Interest

The authors state that this work has not any conflicts of interest.

Acknowledgments

This work has been supported by the program "Estancias de movilidad en el extranjero José Castillejo para jóvenes doctores" funded by the Spanish Ministry of Education, Culture and Sport with Reference CAS17/00005. The authors also acknowledge support from "Universidad de Zaragoza," "Fundación Bancaria Ibercaja," and "Fundación CAI" in the "Programa Ibercaja-CAI de Estancias de Investigación" with Reference IT24/16. They acknowledge the research project "Construcción de un framework para agilizar el desarrollo de aplicaciones móviles en el ámbito de la salud" funded by University of Zaragoza and Foundation Ibercaja with Grant Reference JIUZ-2017-TEC-03. This work also acknowledges the research project "Desarrollo Colaborativo de Soluciones AAL" with Reference TIN2014-57028-R funded by the Spanish Ministry of Economy and Competitiveness. It has also been supported by "Organismo Autónomo Programas Educativos Europeos" with Reference 2013-1-CZ1-GRU06-14277. Finally, they acknowledge support from project "Sensores vestibles y tecnología móvil como apoyo en la formación y práctica de mindfulness: prototipo previo aplicado a bienestar" funded by University of Zaragoza with Grant no. UZ2017-TEC-02.

Supplementary Materials

This supplementary material contains the source code of the agent-based simulator of strategies for managing self-reported priorities in smart cities called ABS-SmartPriority. In this material, the file "SmartPriority.zip" of the project with the source code can be unzipped and opened with the Unity Editor 5.5.1. The material also contains a brief web-based documentation for presenting the simulator and indicating how to define new strategies. *(Supplementary Materials)*

References

[1] H. Yeh, "The effects of successful ICT-based smart city services: from citizens' perspectives," *Government Information Quarterly*, vol. 34, no. 3, pp. 556–565, 2017.

[2] H. B. Sta, "Quality and the efficiency of data in "Smart-Cities"," *Future Generation Computer Systems*, vol. 74, pp. 409–416, 2017.

[3] F. Vicentini, A. Giusti, A. Rovetta et al., "Sensorized waste collection container for content estimation and collection optimization," *Waste Management*, vol. 29, no. 5, pp. 1467–1472, 2009.

[4] L. Parra, S. Sendra, J. Lloret, and I. Bosch, "Development of a conductivity sensor for monitoring groundwater resources to optimize water management in smart city environments," *Sensors*, vol. 15, no. 9, pp. 20990–21015, 2015.

[5] T.-h. Kim, C. Ramos, and S. Mohammed, "Smart city and IoT," *Future Generation Computer Systems*, vol. 76, pp. 159–162, 2017.

[6] M. Yigit, V. C. Gungor, E. Fadel, L. Nassef, N. Akkari, and I. F. Akyildiz, "Channel-aware routing and priority-aware multichannel scheduling for WSN-based smart grid applications," *Journal of Network and Computer Applications*, vol. 71, pp. 50–58, 2016.

[7] T. Anagnostopoulos, K. Kolomvatsos, C. Anagnostopoulos, A. Zaslavsky, and S. Hadjiefthymiades, "Assessing dynamic models for high priority waste collection in smart cities," *The Journal of Systems and Software*, vol. 110, pp. 178–192, 2015.

[8] Y. de Harlez and R. Malagueño, "Examining the joint effects of strategic priorities, use of management control systems, and personal background on hospital performance," *Management Accounting Research*, vol. 30, pp. 2–17, 2016.

[9] G. Martínez-Cánovas, E. Del Val, V. Botti, P. Hernández, and M. Rebollo, "A formal model based on Game Theory for the analysis of cooperation in distributed service discovery," *Information Sciences*, vol. 326, pp. 59–70, 2016.

[10] L. Sanchez, L. Muñoz, J. A. Galache et al., "SmartSantander: IoT experimentation over a smart city testbed," *Computer Networks*, vol. 61, pp. 217–238, 2014.

[11] T. Anagnostopoulos, A. Zaslavsy, A. Medvedev, and S. Khoruzhnicov, "Top—k Query based dynamic scheduling for IoT-enabled smart city waste collection," in *Proceedings of the 16th IEEE International Conference on Mobile Data Management (MDM '15)*, pp. 50–55, Pittsburgh, Pa, USA, June 2015.

[12] I. García-Magariño, C. Gutiérrez, and R. Fuentes-Fernández, "The INGENIAS development kit: a practical application for crisis-management," *Bio-Inspired Systems: Computational and Ambient Intelligence*, vol. 5517, pp. 537–544, 2009.

[13] I. García-Magariño, J. J. Gómez-Sanz, and J. R. Pérez-Agüera, "A multi-agent based implementation of a Delphi process," in *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '08)*, vol. 3, pp. 1543–1546, International Foundation for Autonomous Agents and Multiagent Systems, Estoril, Portugal, May 2008.

[14] S. Hassan, M. Salgado, and J. Pavón, "Friendship dynamics: modelling social relationships through a fuzzy agent-based simulation," *Discrete Dynamics in Nature and Society*, vol. 2011, Article ID 765640, pp. 1–19, 2011.

[15] I. García-Magariño, "ABSTUR: An Agent-based Simulator for Tourist Urban Routes," *Expert Systems with Applications*, vol. 42, no. 12, pp. 5287–5302, 2015.

[16] J. P. Garcia-Sabater, J. Lloret, J. A. Marin-Garcia, and X. Puig-Bernabeu, "Coordinating a cooperative automotive manufacturing network - An agent-based model," in *Cooperative Design, Visualization, and Engineering*, vol. 6240 of *Lecture Notes in Computer Science*, pp. 231–238, 2010.

[17] I. García-Magariño, A. Gómez-Rodríguez, J. C. González-Moreno, and G. Palacios-Navarro, "PEABS: a process for developing efficient agent-based simulators," *Engineering Applications of Artificial Intelligence*, vol. 46, pp. 104–112, 2015.

[18] I. Garca-Magari, R. Lacuesta, and I. Garca-Magariño, "Source code of ABS-SmartPriority," 2017, Published as supplementary material of the current article.

[19] I. García-Magariño, M. Cossentino, and V. Seidita, "A metrics suite for evaluating agent-oriented architectures," in *Proceedings of the 25th Annual ACM Symposium on Applied Computing (SAC '10)*, pp. 912–919, Sierre, Switzerland, March 2010.

[20] I. García-Magariño, G. Palacios-Navarro, and R. Lacuesta, "TABSAOND: A technique for developing agent-based simulation apps and online tools with nondeterministic decisions," *Simulation Modelling Practice and Theory*, vol. 77, pp. 84–107, 2017.

[21] I. García-Magariño and G. Palacios-Navarro, "ATABS: a technique for automatically training agent-based simulators," *Simulation Modelling Practice and Theory*, vol. 66, pp. 174–192, 2016.

[22] J. Cohen, *Statistical Power Analysis for The Behavioral Sciences*, Lawrence Earlbaum Associates, Hillsdale, NJ, USA, 2nd edition, 1988.

[23] J. A. Rosenthal, "Qualitative descriptors of strength of association and effect size," *Journal of Social Service Research*, vol. 21, no. 4, pp. 37–59, 1996.

[24] C. Gutiérrez Cosio and I. García Magariño, "A metrics suite for the communication of multi-agent systems," *Journal of Physical Agents*, vol. 3, no. 2, pp. 7–14, 2009.

[25] R. Fuentes-Fernández, I. Garca-Magariño, J. J. Gómez-Sanz, and J. Pavón, "Integration of web services in an agent-oriented methodology," *International Transactions on Systems Science and Applications*, vol. 3, pp. 145–161, 2007.