System evolution for unknown context through multi-action evaluation

Asanga Nimalasena
Vladimir Getov

School of Electronics and Computer Science, University of Westminster

This is a copy of the author's accepted version of a paper subsequently published in the proceedings of 2013 IEEE 37th Annual Computer Software and Applications Conference Workshops (COMPSACW). IEEE, pp. 271-276. ISBN 9780769549873. It is available online at:

http://dx.doi.org/10.1109/COMPSACW.2013.43

System Evolution for Unknown Context through Multi-action Evaluation

Asanga Nimalasena and Vladimir Getov
School of Electronics and Computer Science
University of Westminster
London, United Kingdom
Asanga.Nimalasena@my.westminster.ac.uk, V.S.Getov@westminster.ac.uk

***Abstract* — Context-aware computing has been attracting growing attention in recent years. Generally, there are several ways for a context-aware system to select a course of action for a particular context change. One way is for the system developers to encompass all possible context changes in the domain knowledge. Then, the system matches a context change to that in the domain knowledge and chooses the corresponding action. Other methods include system inferences and adaptive learning whereby the system executes one action and evaluates the outcome and self-adapts/self-learns based on that. However, there are situations where a system encounters unknown contexts. In such cases, instead of one action being implemented and evaluated, multiple actions could be implemented concurrently. This parallel evaluation of actions could quicken the evolution time taken to select the best action suited to unknown context compared to the iterative approach. This paper proposes a framework for context-aware systems that finds the best action for unknown context through multi-action evaluation and self-adaptation. In a case study, we show how our multi-action evaluation system can be implemented for a hypothetical hotelier who uses the name-your-own-price mechanism to sell his perishable inventory.**

Keywords - context-aware systems; self-adaptation; multi-action evaluation

I. INTRODUCTION

Context awareness is a fundamental concept in pervasive computing. There are many definitions to what is a context. Context could be defined by location [13], location combined with behaviour [3] or encompassing multitude of factors such as the definition given by Dey [5]: "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves". This definition gives the context-aware system developers much more freedom to decide what constitutes a context.

Actions in context-aware systems are determined by the context and any changes to the context. A context-aware application does context inference on the basis of the so-called 5W1H (Where, When, What, Who, Why, How) factors [8]. This means that context-aware applications look at the who's, where's, when's and what's (that is, what the user is doing) of entities and use this information to determine why the situation is occurring [10]. An application doesn't actually determine why a situation is occurring, but the designer of the application does. This means the designer has to capture the domain knowledge and input it to the system. If there is a context which the designer did not foresee, then the context inference would fail.

Self-adapting or self-learning methods are employed to address the issues arising when a system encounters an unknown context. These methods use an iterative approach to finding the best possible action. In the cases when there is a large action space to evaluate, the time to find the best action would take longer and may not correctly identify the nature of the change due to the unknown context.

In order to overcome the problems in the iterative approach to unknown context, this paper proposes a framework which evolves itself when an unknown context is encountered. This is achieved by concurrently executing and evaluating multiple sets of possible actions and then determining the best course of action.

The rest of this paper is organized as follows. Section II reviews related work on context and self-adapting context-aware models. Section III gives a brief description of the proposed framework and details description of the multi-action evaluation system within the proposed framework. Section IV describes the evaluation of the concurrent action execution and evaluation system of the proposed framework. Section V presents experimental results of the evaluation. Finally, the paper concludes with Section VI which summarizes the findings from the evaluation and outlines directions for future work.

II. RELATED WORK

A. *Single Context – Single Action*

This category includes relatively simple context-aware systems that are specifically developed for smart environments and suffer from the earlier mentioned problem – the designer has to capture and input the domain knowledge into the system. Therefore, this limitation prevents such systems from being used in multi-context environments. One common theme of these smart environments is that, based on sensory data of one or more devices, other device(s) state(s) is/are changed to bring the environment (in another word context) to an optimal level that is most beneficial to the entity in that context. He et al [6] provide an example of such a smart plant-watering context-aware system. Three main areas could be identified in these types of context-aware frameworks – they are sensory data acquisition, context inference/management and action.

Due to the nature of its application a context-aware framework developed for these smart environments has an output action that is of two mutually exclusive states for a given context. In other words, the sensor data and action have a one-to-one relationship. For example, a context-aware application developed to control the ambient temperature in a room would have actions to turn on or off a fan or set the temperatures of the air conditioning to one specific value *Z* when the ambient temperature is *X* degrees or between *X* and *Y* degrees. It is not possible to have and neither does it make sense to incorporate into the context-aware framework actions such as: when the ambient temperature is *X* degrees turn on and off the fan or set the air conditioning temperature to *P* and *Q* degrees.

B. Single Context – Multiple Actions

One way to overcome this problem is to include system inferences and adaptive learning whereby the system executes one action and evaluates the outcome and self-adapts/self-learns based on the outcome. This method is used when an unknown context is encountered. For example, a context-aware system can be based on the context triple model – RAP – where R is a set of context resources, A is a set of actors which interact with context resources, and P is a set of real context related policies. A self-adapting algorithm which implements the RAP model is presented in [4]. This algorithm uses a closed feedback loop with four phases – monitoring, analysis, planning, and execution.

Another approach [9] proposes a formal method for incremental context awareness based on two monotonic extension models. The breadth-monotonic model extends the system so that it recognizes more situations (context) than before while the depth-monotonic model is applied for the cases when there is uncertainty and the system extends itself through estimation.

A third approach [12] defines a self-adapting context with the use of context edges (a context edge is the border between two contexts) and context spaces where the model is based on Q-Learning with a feedback loop which finds the optimal action for each state by the reward it receives from the environment for actions taken in that state.

Other proposed models include a model using case base reasoning to address domain specific problems and incomplete data sets [11]. The models mentioned above try to address the lack of domain knowledge through self-adapting whereas [8] proposes a model where both ontological and Bayesian network probabilistic reasoning are used for context reasoning and the context is modelled using ontology. Similarly, the approach described in [2] uses fuzzy sets to allow imperfection in context that is being sensed.

These models could be used to implement single context – multi-action systems. When an unknown context is encountered in such systems, it is possible to execute multiple actions iteratively and then evaluate the outcome of each of these actions to arrive at the best action for the given unknown context. In these systems it is also possible to have associated a single unknown context value with two or more actions before deciding on a best course of action. However, due to the iterative approach where each action in the action space is executed one at a time and then evaluated to find the best action, when the number of actions to execute and evaluate increases, the amount of time taken to find the best possible action becomes unacceptably long.

C. Multiple Context – Multiple Actions

In real world environments multiple contexts are considered and the output action correlates to a set of context values rather than to a single context, making the system multi-context – multi-action one. How such a multi-action context-aware system comes into use could be illustrated with the name-your-own-price (NYOP) application [15]. NYOP is a strategy where the buyer suggests the price which he/she is willing to pay for goods or services without knowing the minimum threshold price *T* which is acceptable to the seller. Imagine a hotelier who sells his inventory through such a NYOP channel. If the decision to accept or reject a bid is solely based on the bid value, then the hotelier is not going to have the fluidity to react to the demand uncertainty that occurs due to the change in context.

For example, consider a new event has been planned near the vicinity of the hotel and there is no historical data or knowledge to rely on which means that this is an unknown context.

In this case, instead of having one threshold price *T* the context-aware NYOP system could be set up with multiple threshold values *T1* and *T2*. Type of event and duration of the event could be considered as context values as these will affect the bid values, action space would consists of number of threshold values evaluated concurrently against the bids. Domain knowledge would provide the starting set of threshold values eliminating threshold values that are not worth evaluating against thus reducing number of actions in the action space.

The business model of the hotelier does not suffer from having multiple threshold values. Indeed, because the items are either perishable or time-dated the retailer wants to obtain as much revenue as possible and is ready to accept even lower bids as long as the bids that are willing to pay more also materialize [15].

The hotelier's NYOP channel decisions could be influenced by many contexts which may not relate to one another. Other examples of such contexts that the hotelier's system could be part of are specific weather conditions (ice, snow, hurricane), current occupancy rate (high, low) and even the legislation (increase/decrease of taxes). Each of these contexts would have multiple values which will have a varying degree of influence on the hotelier's decision making process. For example, considering the earlier contexts, events could be conferences, official gathering and weddings. Each of these events has different characteristics which must be considered in the decision making process. The weather could be defined with concrete parameters such as temperature 27^{o} C, humidity 60%, as well as using vague or fuzzy terms such as "sunny day" or "mildly chilly day".

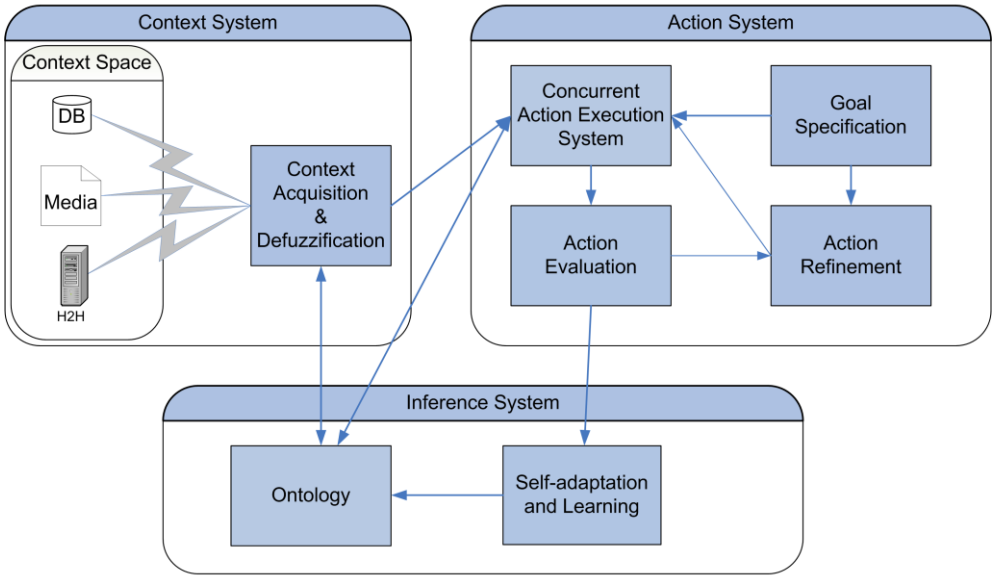

Figure 1. High-level system diagram of proposed framework

Permutations of these multiple contexts could be defined as below.

{Conference, Sunny day}→ action space {A1, A2, ..,An}

{Wedding, {27° C, 50% Humidity}, 20% Occupancy} → action space {Aa,Ab,..,Az}

Each permutation of these contexts result in multi-action evaluation as the hotelier is unlikely to have encountered all the permutations and has to decide on the best course of action whenever a new permutation of the context values occurs. As the number of permutations for different context value combinations increases exponentially with an addition of each new context, the environment that has to employ multi-context – multi-action systems faces several challenges. The main challenge is the high number of actions or action values to evaluate which results in much longer time needed to find the best action for current unknown context.

The multi-action execution and evaluation framework proposed in this paper is somewhat similar to the agent's action evaluation in subsumption architecture [14] where an agent executes multiple actions to evaluate the best one.

However, the subsumption architecture arranges the modules into a hierarchy where actions are divided into low priority and high priority as one of its characteristics. In contrast, the proposed solution in this paper considers all actions to be equal and to have the same level of priority until the outcome is evaluated against the goals.

III. PROPOSED FRAMEWORK

The proposed framework consists of three systems. They are the context system, the inference system and the action system. Fig. 1 shows a high-level diagram of the framework and the interactions between each system.

The primary objective of the context system is context acquisition and defuzzification. It is assumed that context space is a heterogeneous where context values are acquired from various heterogeneous sources. Taking the earlier example of the hotelier, the occupancy rates would come from the hotelier's own database, weather information from a host-to-host (H2H) weather service and information about upcoming events through various media sources (electronic paper and social media).

The inference system forms the key part of self-adaptation. When the best course of action is found for an unknown context this information is stored within the ontology for future use. The ontology's inherent inference capabilities can be used to check if an unknown context could be deduced to a known context before proceeding to concurrent multi-action evaluation.

The action system encompasses all sub-modules related to concurrent action execution and evaluation of action outcome based on a goal specification. It is only this action system that this paper focuses on and evaluates in Section IV.

The action systems come into play when an unknown context is encountered by the framework. This would result in multiple actions being executed for the same set of perceived context values.

The framework address the problem of high number of actions in the action space to evaluate with the use of goal specifications. Goals specify the acceptable range of outcomes and for an action to qualify to be considered, its outcome must fall within the goal specification. This reduces the number of actions to be considered in the action space. The actual values for goal specifications is determined by domain knowledge and dependent on the concrete implementation of the framework.

Taking the earlier mentioned hotelier's NYOP example, the goal specification of the hotelier could be a threshold prices between ($100, $200). This means that any action that evaluates the threshold price lower than $100 or higher than $200 is not considered. Though the hotelier's NYOP is used

here for the evaluation, the framework could be adopted into any context-aware system where the domain allows multiple actions to be executed concurrently and then evaluated to find the best possible option. Examples of such systems include cloud services that provide spot instances [1] to find the optimal market value under various demand and resource constraints scenarios.

A separate concurrent action execution system is used while the actions are executed before the evaluation. This is intended to provide a separate environment for the actions under evaluation. When an unknown context is encountered, the system could evaluate either all possible actions within the goal specification (brute force evaluation) or start with the action of a context which is the closest to the unknown one. The latter will further reduce the number of actions requiring evaluation.

The closest known context action is found by calculating the difference between the known context values and values in the unknown context set. If there are two sets with equal differences, then the priority of each context is considered, based on the degree of influence each context value has on the overall outcome. If two known contexts have the same set of difference values, the decision is based on the selected direction to start off – optimistically or pessimistically. We define starting off optimistically as assuming that the overall outcome of the system due to unknown context is higher than the outcome from the closest known context action. Similarly, starting off pessimistically is defined as assuming that the overall outcome of the system due to the unknown context is lower than the closest known context action outcome. Once the closest action is determined, then it is expanded both optimistically and pessimistically with actions whose outcome is within the goal specification range.

The expansion criteria are defined in the action refinement system which controls how many actions are in the action space. This is done by specifying how many actions to expand pessimistically and optimistically, as well as the distance between the outcomes of the two nearest actions. For an action *A*, the resulting outcome *O* is defined as a function of *O(A)*. Then, the total action space for evaluation under the goal specifications G_{lo} and G_{hi} with an optimistic and pessimistic expansion criteria n,m ($n,m > 0$), distance between the outcomes for the two nearest actions Δ ($\Delta > 0$) and starting off with closest known context's action $O(A_c)$ could be defined as follows.

$$G_{lo} < O(A_n) < O(A_{(n-1)}) < O(A_c) < O(A_{(m-1)}) < O(A_m) < G_{hi}$$

Here, the terminating conditions for the expansion are defined below.

$$O(A_{(n+1)}) < G_{lo} \quad (1)$$

$$O(A_{(m+1)}) > G_{hi} \quad (2)$$

$$O(A_{(n-1)}) - O(A_n) = O(A_m) - O(A_{(m-1)}) = \Delta \quad (3)$$

Equation (1) defines actions the outcome of which is outside the lower end of the goal specification, while (2) defines actions the outcome of which is outside the higher end of the goal specification. Equation (3) specifies that the difference between the outcomes for any two nearest actions in the action space is equal.

The action evaluation consists of the evaluation criteria used to determine the outcome of which action is the most beneficial. The evaluation criteria are subjective to the concrete implementation of the framework. For example, the hotelier's evaluation criteria would be the action that gives the highest yield. If the framework is implemented for an error correction system, then the evaluation criteria would be the action that results in the lowest error.

IV. METHODOLOGY

For the evaluation test case we have envisaged a hypothetical scenario where an hotelier sells rooms through the NYOP channel. As discussed earlier, the NYOP operates by allowing buyers to bid for an item on a perceived value rather than based on the actual value set by the seller of the item. The seller has an internal threshold value hidden from the buyers which he or she considers to be the minimum value for a bid in order to successfully complete the transaction. There is a variety of NYOP strategies, such as allowing multiple bids, restricting subsequent bids to happen after a time laps [7, 15] but for our experiments we do not employ any such NYOP strategies. Instead, each value is considered as an individual bid and not as a subsequent bid part of a bidding transaction. We also assume that there are no restrictions to using multiple threshold values to evaluate the bids.

For the evaluation we implemented standalone proof-of-concept module only for the action system. The evaluation is only concerned with the concurrent multi-action evaluation and it's assumed that the system is working under the condition that it has already encountered an unknown context. We have employed the strategy of start off with the closest known context.

We have used two test cases and a control test to go along with our experiments. One of the test cases simulates an unknown context in which the mean value of the bids is lower than the threshold value of the closest known context. This case we refer as the pessimistic case where if the hotelier does not adjust the threshold value by lowering it to capture the bids, he or she will lose out under the current context.

The second test case simulates an unknown context under which the mean bid values are considerably higher than the threshold value of the closest known context which we refer to as the optimistic case. Under this context the hotelier has to increase the threshold value to be just below the mean value of bids to prevent the bids between the mean value and the current threshold value being succeeding. This is a NYOP strategy that encourages higher bidding values. Though we make no assumption about the bidding strategies we include this test case for the completeness of the evaluation, by showing that the framework works for both the optimistic and the pessimistic cases.

For the above two test cases each bid value will be evaluated concurrently against all the actions in multi-action space. It is possible some bids would be successful in more

than one threshold. In such cases the bid would be considered successful only in the highest threshold it exceeds and successful bid count will be calculated accordingly.

We generated bid values using a normal distribution and adjusted the mean value to make the generated values fit to either pessimistic case or the optimistic case. We set the standard variation to the same value as the expansion criteria thus ensuring that there are bid values for each range.

Though we use the normal distribution to generate input bid values it makes no difference to the outcome of the test case even if they are of different distribution. This is because the best course of action is chosen after evaluating all the bid values with all the actions in the action space.

For the control test we employ a self-adaptive context-aware model that iteratively executes the all possible actions when an unknown context is encountered and finally evaluates the outcomes to find the best fit. We selected a sample size of (total inputs bids / action space size) for each action before the next iteration begins. With this sample size each action in the action space will evaluate the same number of bids and the total number of test cases would be the same across all test cases. Under control test cases each bid is only evaluated by one action and bid is not repeated and also it will evaluate all the bids in the same order as they appeared in the two main test cases.

The action evaluation is done based on the number of successful bids for each threshold value. The best course of action would be the one with the highest number of successful bids. The system evolution would result in associating threshold value used by this action being associated with the unknown context.

For the test case we had a goal specification $(G_{lo}, G_{hi}) = (190, 300)$. We set up equal numbers of pessimistic and optimistic expansion – $(n, m) = (2, 2)$ – making the total actions in the action space (including the closest known action) to be 5 and the distance between each two nearest actions outcome to be 15. This value is derived from the hotelier's domain knowledge.

For the unknown context encountered, the system found the closest known context action threshold value to be 225. We had to know the closest known context action threshold value beforehand in order to explicitly generate bid values to match the pessimistic and the optimistic cases.

We have generated for the pessimistic case 1000 bid values using a normal distribution function with a mean value of 212.50. For the optimistic case we have generated another 1000 bid values with a mean value of 243.50.

We ran two control tests per each of the cases above making four control tests altogether. The two control tests differ from each other depending on the direction of the iteration, whether it traverses in the optimistic direction or in the pessimistic direction. We start off with the closest known context action threshold value and evaluate the first 200 (derived from the total inputs bids / action space size) bid values in the same order as the previous test cases. In the subsequent iteration we select the nearest pessimistic action for one test and when all pessimistic actions are exhausted we move to the optimistic actions. In the other test we start off from the closest action and move in the direction of the optimistic actions before evaluating the pessimistic action. This is to check if the order (optimistic action first or pessimistic action first) and the nature of the action chosen has any effect on the overall outcome compared with the concurrent multi-action execution and evaluation.

V. RESULTS

The expansion from the closest known context action threshold value resulted in five actions that will evaluate bids with 5 different threshold values. The threshold values were 195, 210, 225, 240 and 255. These are denoted as A(195), A(210), A(225), A(240) and A(250) in the graphs below.

A. *Pessimistic Test Case*

The results of the pessimistic test case (Fig. 2) show that under the current unknown context the majority of successful bid values were evaluated by an action that had a threshold value of 210. In essence, our hypothetical hotelier could associate the current unknown context with the threshold value 210 for future evaluation thus effectively evolving the system to recognise the current unknown context in the future. We know this conclusion to be correct as we generated the bid values using a normal distribution with a mean value of 212.50.

Looking at the control test cases under the optimistic direction we only had 332 successful bid values while in the pessimistic direction we only had 342 bid values. The low success rate is due to employing only a single action which evaluates the bids by using only a single threshold value. Furthermore, the results of the test case erroneously show 195 to be the threshold value under which the majority of bids are successful. This proves that using a single action it is not possible to capture the nature of the change due to the context change evaluating a set number of bids (1000 bids in this case) as compared concurrent evaluation.

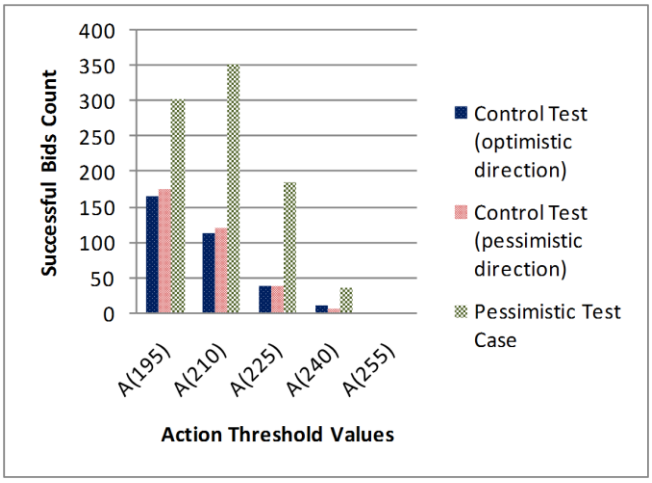

Figure 2. Successful bid count for test case where unknown context result in pessimistic bid values

B. *Optimsitic Test Case*

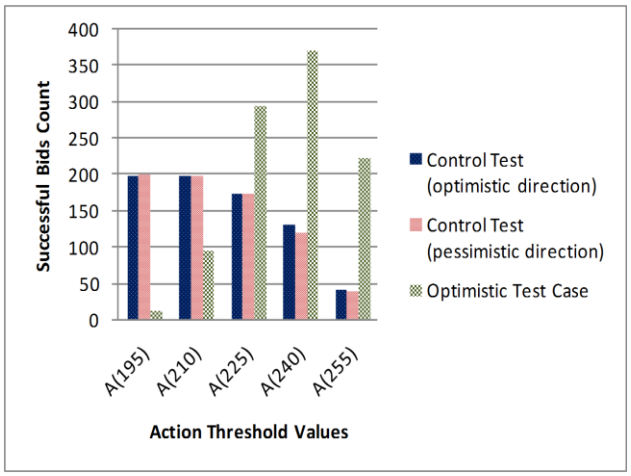

Figure 3. Successful bid count for the test case where unknown context results in optimistic bid values

The results of the optimistic test case (Fig. 3) show that under the current unknown context the majority of successful bids occur with an action that evaluated them with a threshold value of 240. Thus, the hypothetical hotelier could evolve the system to recognise the current context as an optimistic one and set 240 as the threshold for any future actions that evaluate the buyer bids under the current context set. We know this to be true as the bid values generated under the normal distribution had a mean value of 243.50.

In this case the control test cases, both in the pessimistic direction and in the optimistic direction, resulted in erroneously identifying that using a threshold value of 195 is the correct course of action. In fact, this indicates that the context change, or the unknown context encountered has a pessimistic effect on the bid values when we know for a fact the opposite to be the truth.

Examining the results from both test cases shows that concurrent multi-action execution and evaluation is able to identify the best course of action or an unknown context thus evolving the system to work in previously unknown contexts.

VI. CONCLUSION

In this paper, we proposed a framework for system evolution when an unknown context is encountered through concurrent multi-action evaluation. An implementation of the action system of the proposed framework was completed for the NYOP scenario. The experimental results from the tests have shown that the concurrent multi-action evaluation is capable of correctly identifying the best course of action for the unknown context and is able to evolve the system so the system could recognize the context in the future. These promising results complement very well the much faster evaluation time for our concurrent multi-action framework when compared to context-aware systems based on the iterative approach.

Though we implemented the action system of the framework for an hotelier selling with a NYOP channel, we believe the framework could be easily adopted for any domain that allows concurrent multi-action evaluation. For example, a cloud provider could use the proposed framework to adjust the market value of spot instances using wide variety of context values such as current resource availability, current demand and future demand for spot instances and even react quickly to dynamic changes in the competitors' spot instance market values. Another potential domain of high interest is the application performance tuning, where the proposed framework could be used to determine which set of application parameters give the greatest performance gains. Both the cloud computing and the performance tuning areas provide exciting opportunities and directions for future work based on our multi-action evaluation framework.

REFERENCES

[1] Amazon EC2 Spot Instances, http://aws.amazon.com/ec2/spot-instances/ , last visited on 19 Jan 2013.

[2] C. Anagnostopoulos and S. Hadjiefthymiades, Advanced Inference in Situation-Aware Computing. IEEE Trans. Systems, Man and Cybernetics, Part A: Systems and Humans, 39 (5), 1108-1115, 2009.

[3] P.J. Brown, J.D. Bovey and X. Chen, Context-aware applications: from the laboratory to the marketplace, IEEE Personal Communications, 4 (5), 58-64, 1997.

[4] T. Cioara et al., A self-adapting algorithm for context aware systems, Proc. 9th Roedunet Int. Conference (RoEduNet), pp. 374-379, 2010.

[5] A.K. Dey, Understanding and Using Context, J. Personal and Ubiquitous Computing. 5 (1), 4-7, 2001.

[6] J. He, Y. Zhang, G. Huang and J. Cao, A smart web service based on the context of things. ACM Trans. Internet Technology. 11 (3), 13:1-13:23, 2012.

[7] O. Hinz, I. Hann and M. Spann, Price discrimination in e- commerce? An examination of dynamic pricing in name-your-own price markets, Mis quarterly, 35 (1), 81-98, 2011.

[8] K. Kwang-Eun and S. Kwee-Bo, Development of context aware system based on Bayesian network driven context reasoning method and ontology context modeling, Proc. Int. Conf. Control, Automation and Systems (ICCAS), pp. 2309-2313, 2008.

[9] S.W. Loke, Incremental awareness and compositionality: A design philosophy for context-aware pervasive systems, Pervasive and Mobile Computing. 6 (2), 239-253, 2010.

[10] J. Madhusudanan, A. Selvakumar and R. Sudha, Frame work for context aware applications, Proc. Int. Conf. Computing Communication and Networking Technologies (ICCCNT), pp. 1-4, 2010.

[11] N. Nwiabu, I. Allison, P. Holt, P. Lowit and B. Oyeneyin, Situation awareness in context-aware case-based decision support, IEEE 1st Int. Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), pp. 9-16, 2011.

[12] N. O'Connor, R. Cunningham and V. Cahill, Self-Adapting Context Definition, Proc. 1st Int. Conf. Self-Adaptive and Self-Organizing Systems (SASO '07), pp. 336-339, 2007.

[13] B.N. Schilit and M.M Theimer, Disseminating active map information to mobile hosts, IEEE Network, 8 (5), 22-32, 1994.

[14] M. Wooldridge, Intelligent Agents. MIT Press, 1999.

[15] J.G. Wilson and G. Zhang, Optimal design of a name- your- own price channel, J. Revenue and Pricing Management, 7 (3), 281-290, 2008.