Towards self-optimization of a pervasive computing task

Manolo Dulva Hina a,*, Amar Ramdane-Cherif b, Marietta S. Sicat c, Rodolfo S. Vargas a

a College of Computer Studies, AMA International University Bahrain, Salmabad, Kingdom of Bahrain
b LISV Laboratory, Université de Versailles-Saint-Quentin-en-Yvelines, France
c AMA International School, Budaiya, Kingdom of Bahrain

Abstract

In pervasive computing, our aim for the end user is continuous computing anytime, anywhere whenever possible, in a way that the computing system adapts to the user’s context. An integral component of pervasive computing is pervasive task, a computing task that follows wherever the user is, adapting to the computing needs of the user and self-optimizing on user’s behalf. In this paper, a task is represented by a set of computing files which then are mapped to their corresponding applications and suppliers, and optimal quality of service dimensions for each application. Here, we present how pervasive task can be realized, implementing self-configuration and self-optimization to adapt to the user’s preferences. This work is a contribution in the ongoing research towards adaptation of computing system to suit the user’s context and preferences in a pervasive computing system.

Keywords: pervasive computing; adaptive system; pervasive task; self-optimizing system; self-configuring system

1. Introduction

The essence of a pervasive computing system is to provide an end user with an infrastructure that will realize anytime, anywhere computing, allowing the user to continue working on his interrupted computing task when he wants, where he wants. This is Weiser’s vision of ubiquitous computing [1] [2]. A component of Weiser’s vision of calm technology [3] is the realization of pervasive user task. A task refers to a collection of computing job wherein a user needs to use a computer and computing resources to do the job. Furthermore, such task is realized by using one or more computer applications. There are, however, multiple suppliers for each of these applications. For example, a task of the user’s search for an affordable second-hand car according to his budget can be accomplished by using a series of computer applications such as web browser, word processor, spreadsheet, email, etc. Then the word processing application, for instance, can be realized by using a certain supplier such as MSWord, WordPad, Emacs, etc. For all these permutations and possibilities, there must be some optimal configuration as per user context and preferences. Indeed, the selection of a supplier needs to be based on the supplier’s suitability to the given context and to the user’s preferences. Given that the user may use one or more suppliers for a single application, we adapt priority scheme in which the highest priority is assigned to the supplier that the user prefers the most and the lowest priority to the supplier that the user prefers the least. The quality-of-service (QoS) dimensions of an application are parameters that consume computing resources and its values are synonymous to the quality of an application. The values that will be assigned to the QoS dimensions are also priority based and such priority is per user’s preference. To realize the

* Corresponding author. Tel.: (+973) 3888-4269; Fax: (+973) 1787-9380.
E-mail address: mhina@amaiu.edu.bh

© 2011 Published by Elsevier Ltd. Selection and/or peer-review under responsibility of [name organizer]
ubiquity of documents associated with the user’s task, the ubiquitous documents need to be stored in a server and is replicated on every member of a server group, making it accessible to the user whenever and wherever he wishes. This research work is a result of our desire that computing documents do follow where the user is, enabling him to do his computing task when he wants and where he wants. This research work is aligned with task-driven computing [4] and is a continuation of our earlier paper on pervasive task [5, 6].

This paper presents the concepts and principles used in the infrastructure to realize pervasive task. Data validation is done through case studies and formal specifications. This proposed solution is functional under these assumed conditions: (i) that wired and wireless communication facilities are available to support the networking needs of our system; (ii) that there are available application suppliers and media devices to support the computing needs of the user.

Apart from this introductory section, the rest of this paper is structured as follows: Section 2 provides a brief review of the state of the art and Section 3 presents our contribution. In Section 4, we present the infrastructure that realizes pervasive computing task. Section 5 presents some situational simulations. The paper is concluded in Section 6.

2. Review of the state of the art

Today’s computing has helped significantly in improving computer user’s productivity and efficiency. It would elevate user’s productivity even further if the user’s task and data could be transported from one environment to another and make the user able to continue working on an interrupted task anytime and anywhere he wants. Ubiquitous computing (aka pervasive computing) [7] is the answer. It is an important innovation that enhances user’s productivity. Ubiquity of user’s task can be achieved by making the user’s task, profile, data and task registry transportable from one environment to another. In order to realize pervasive computing, a network system that supports wired and wireless computing must exist.

Ubiquitous computing is a promising domain. Work relevant to ours includes that of Garlan et al [4] who worked on Carnegie Mellon Project Aura; their Prism work was a motivating literature for us when we started our research. Our work has some similarities with theirs because of the nature of pervasive computing where user’s task must be omnipresent. Ours is equally different we intend to implement various computing domains to realize our goal, including the adaptation of a multi-agent system that uses incremental learning [8] and the self-adaptation and self-optimization concepts of autonomic computing [9, 10]. The element important to pervasive computing is the network system that supports wired and wireless computing must exist.

3. Novelty and contribution

In the conceptualization of an adaptive system that selects an optimal configuration for a user’s task and in the design of a system that provides configuration suitable to the user’s satisfaction, we raised these related questions: How do we associate a document to an optimal application supplier? What would be the basis of such association? If this optimal supplier is not available, how will we find a relevant replacement? How do we represent and quantify user’s satisfaction? What parameters are used to measure the user’s satisfaction with regards to system configuration?

Our proposed solutions follow: We formulated function and relations that associates a user task to different applications, to different suppliers, and to different QoS dimensions, taking into account user’s preferences. Since different suppliers are apt for different scenarios, part of our proposed solution is the ranking of application suppliers. In general, it is possible that in a given scenario, two or more suppliers may be found suitable for invocation, with the top-ranked supplier being activated by default. When the chosen supplier is not available, then the next-ranked supplier is taken as its replacement. We use the utility measure [0,1] to denote user satisfaction wherein a value of 0 means the condition is inappropriate while a value of 1 means that the user is happy with the condition. In general, the closer the configuration setting is to the user’s pre-defined preferences, the higher is the user satisfaction. In this work, we consider the application supplier and its QoS dimensions as parameters for system configuration.

4. Infrastructure to realize pervasive computing task

The network file system that supports pervasive computing must be functional via wired/wireless communication, must be scalable and reliable. Fig. 1(Left) shows a network composed of distributed servers, each of which belongs to the server group. A user’s file (also called volume) is stored and replicated in every member of the server group, hence enhancing the user’s task ubiquity. A workstation group may at one time be disconnected from the server. A ubiquitous system must be able to recover from disconnection and still have a reliable data. This is done by storing
user’s files in a cache/local memory while workstation is disconnected. Upon reconnection, the cache communicates with a member of the server group to update user file. This concept is inspired by Venus system of Coda [11].

The standard adopted by the IEEE Society since 1997 for wireless local area network is the IEEE 802.11. It defines the media access control (MAC) and physical (PHY) layers for a LAN with wireless connectivity. It addresses local area networking where the connected computers communicate over the air to other computers that are within close proximity to one another. It has two topologies: the ad-hoc network (where almost every computer node could communicate with other nodes, and is not suited for our framework) and the infrastructure network (the configuration that fits our needs). In the wireless network diagram of Fig. 1(Left), some access points (AP’s) linked to the server are available with which mobile laptops can communicate. The cards in the laptop (PCMCIA) and the AP’s support the MAC and PHY layers of the IEEE 802.11. The rest of the AP device acts as a bridge to convert the 802.11 protocol to MAC and PHY layers of the backbone distribution service (DS) which is typically an IEEE 802.3 Ethernet LAN [10].

4.1. User’s ubiquitous file system structure

Fig. 1(Right) shows the file structure stored on a local machine’s cache. The local machine may be used by many users; hence, each user has his own storage space in the Documents and Settings directory. In Fig. 1(Right), we showed the contents of the file directory of a user, named “Amar”. Here, Amar’s directory contains the usual file structure present in Windows 7 operating system plus the user’s profile and application registry. The user profile directory is composed of the user profile, the user QoS preference, and the user supplier preference. For each application, its registry would contain the supplier of the service, the last file opened in the application, and the features of application. In Fig. 1(Right), the registry used for every application is shown in generic format. This is because every registry for every application is distinct. The settings, for example, of a text editor application are obviously different from the setting of, say, web browser. In implementation, each application registry is developed to accommodate all features common to every possible supplier of such application. The following would be a possible registry entry for a text editor application:

supplier="MSWord",file="abc.doc",windowPane_height=<value>,windowPane_width=<value>, spelling_enable=yes, font="TimesNewRoman",font_size=11, cursor_position=<line_y, column_x>, zoom=100%, language="American English".

Fig. 1. (Left): Network configuration supporting pervasive/ubiquitous computing. (Right): Structure of ubiquitous file and application registry.

The concept of registry is important in transporting a user’s task from one environment to another. As the user starts working in a new environment, he would find that the application he left behind earlier is available right away with the file and application settings being exactly the same. Hence, he would simply continue working on his task.

4.2. The user profile (UP)and functions for the optimization of user’s task

The UP directory would contain three files, each one related to the others. The UP (Fig. 2 (Left)-a) contains username, password, descriptions of computer1, computer2, … computern. The user schedules are important in tracking down the movement of a mobile user, and to ensure that user’s task could be transported to the user’s location at a specific time. Fig. 2 (Left)-b illustrates the user QoS preference file which contains the values of individual QoS dimension of every application. For each QoS dimension, the user is given the range of values (minimum and maximum) and the user enters his desired value. The default value is the average (minimum + maximum/2). In the given figure, the video player has three QoS dimensions, namely: the frame rate, the frame quality, and the audio quality. Each of these dimensions has a user preferred value which could be modified as the user wishes. In the figure,
we present a partial list of multimedia applications and their QoS dimensions. In general, a QoS dimension is present in an application no matter which supplier is chosen, and such dimension consumes computing resources. The higher/better the setup of the QoS dimension, the more computing resources are consumed. In Fig. 2 (Left)-c, we have a list of user’s preferred supplier for different applications. Each supplier has its preference value as entered by the user. Also, it has a value on the user’s notion of replacing it with another supplier. These values are used by the system in setting up the parameters of the user’s applications.

Fig. 2 (Right) illustrates the server-workstation communication. The interaction begins when the user logs into the system, in which case the server transports the UP onto the local machine’s memory (called cache). The elected server in this transfer is the default server for a wired network and is the one that is located nearest to the mobile computer. In case of a server group, it is the server member that has the latest copy of the user files. Usually, every member of the server group has the same copy of this data. However, in case that there is discrepancy of the versions or timestamp of the data involved, an election is made and the elected server, having the latest copy of the files, sends the files involved onto the workstation, and then updates the copies of the members of the server group. This task is accomplished using the modified bully election algorithm [12].

Fig. 2 (Left): A generic user profile structure. (Right) Work-station interaction.

The communication between the workstation and the server repeats when the user exits the system or upon reconnection after the workstation gets disconnected from the server. Without experiencing disconnection, the workstation will transfer its cache’s data (user files) to the server upon the user’s exit. If disconnected, the local machine stores the user’s file in its cache while disconnected, and sends them to the server upon reconnection. In either case, the server sends a copy of the updated data onto each member of the server group. The timestamp of the files are used to determine which files in the server need to be updated.

For the user to work on his computing task, applications need to be instantiated, preferably using the suppliers that are preferred by the user. These suppliers themselves need to be configured so as the QoS dimensions preferred by the user are satisfied, if possible. The following functions are used in the self-optimization configuration of user’s task:

- $f_1$: Data format $\rightarrow$ Application
  This function maps data format (i.e. of form filename.extension, such as filename1.doc, www.somesite.html, etc.) with its corresponding application. Every data pair belonging to this function is given a relevance score. For this work, relevance scores are $H$, $L$ and $I$ denoting high, low and inappropriate relevance, respectively. Examples of data format-application mappings and their score are as follows: (filename.doc, text editor) has a score of $H$, (www.website.html, video player) has a score of $I$, and (filename.txt, web browser) has a score of $L$.

- $f_2$: Application $\rightarrow$ (Preferred supplier, Priority)
  This function maps an application to a user’s preferred supplier and such supplier’s priority ranking. In general, an application can be implemented using more than one supplier. For example, Text Editing can be implemented using Microsoft Word, Emans, NotePad, WordPad, etc. Web Browsing, on the other hand, can be done using Internet Explorer, Google Chrome, Mozilla Firefox, Opera, etc. The default supplier to implement the application is the one with the highest priority. The priority ranking of these suppliers is based solely on user’s preference. Sample elements of function $f_2$ can be: $f_2 = (\text{Text Editor, (MSWord, 1)})$, (Text Editor, (WordPad, 2)), etc.

- $f_3$: Application $i$ $\rightarrow$ (QoS dimension $j$, Priority)
  This function maps a specific application $i$ to its QoS dimension parameter $j$ where $1 \leq i \leq \text{app\_max}$ (maximum number of applications) and application $i$ $\in$ User task. Recall that in this work, task is a computing work the user needs to do; to accomplish this task, the user runs one or more computing applications. For example, a user wishing to shop for a second-hand car on the Internet may use a web browser, a text editor, a video player, and
other applications to do his search and selection of suitable second-hand car. Also, \(1 \leq j \leq \text{qos\_max} \) (maximum number of QoS dimensions) and QoS dimension \( j \in \text{application} \ i \). Priority is of type \( \mathbb{N} \) (i.e. integer greater than zero). Since there are many possible values for each QoS dimension, the user arranges these values by their priority ranking. Possible elements that may be associated with this function may be:

\[
f_j = \{(\text{Text Editor}, \{40 \text{ characters per line, 1}\}), \ (\text{Web Browser}, \{\text{low page loading, 3}\}), \ (\text{Audio/Video Player}, \{\text{medium volume, 1}\}), \ (\text{Audio/Video Player}, \{\text{high volume, 2}\}), \ etc.\}
\]

Given a user task, one or more applications may be instantiated. In instantiating an application, however, one or more suppliers may be available and QoS dimensions of an application may or may not be available in some suppliers. Indeed, respecting the user’s preferences is the way to instantiate an application. If a user’s preference cannot be supported, the dynamic reconfiguration mechanism of our system must look upon various configuration spaces and determine the configuration that is most appropriate to the given context and user’s needs. Given a user’s task that can be implemented using various applications, we have:

- **A task’s QoS dimension space is given by:**

\[
\text{QoS dimension space} = \bigotimes_{i=1}^{\text{qos\_max}} D_i
\]

(1)

Given two applications \( s \) and \( t \), the applications’ QoS dimension space is \( D_i(s) \otimes D_i(t) \).

- **An application \( i \) has its own set of suppliers, called supplier space:**

\[
\text{Supplier space} = \bigotimes_{i=1}^{\text{app\_max}} \text{Supp}_i
\]

(2)

A feasible configuration is a set-up that satisfies the user’s preferences given the context and constraints. When the configuration is feasible, it is said that the user’s satisfaction is achieved. Let the user’s satisfaction to a configuration outcome be within the bound of satisfaction space. The satisfaction space is within the interval \([0, 1]\) wherein a value of 0 means the outcome is totally unacceptable whereas the value of 1 corresponds to the user’s satisfaction. Whenever possible, the system strives to attain an outcome of 1 or something close to it.

- **Given an application, the user’s satisfaction is enhanced if his preferences are enforced. The supplier preference in instantiating an application are given by:**

\[
\text{Supplier preference} = h_s x^s \cdot f_s c_s
\]

(3)

where \( s \in \text{Supplier space} \) is an application supplier.

- **The term \( c_s \in [0, 1] \) reflects how the user cares about supplier \( s \). Given an application, if it has \( n \) suppliers, arranged in order of user’s preference, then the value of \( c_s \) for these suppliers are \( c_{\text{supplier1}} = 1; c_{\text{supplier2}} = 1 - 1/n; c_{\text{supplier3}} = 1 - 1/n - 1/n, \) and so on. The last supplier, therefore, has a value of \( c_s \) close to 0, which means that the user cares not to have it if given a choice. In general, for each application, the \( c_s \) assigned to supplier \( i \), where \( 1 \leq i \leq n \) and \( n \) is the maximum number of suppliers, is given by:**

\[
c_{\text{supplier \_i}} = 1 - \sum_{1}^{(i-1)/n}
\]

(4)

- **The term \( f_s: \text{dom}(s) \rightarrow [0, 1] \) denotes the expected features present in supplier \( s \). The supplier features are those features, other than the QoS dimensions, that are important to the user. For example, for a text editing application, the user might prefer a supplier that provides a spelling and grammar checking, an equation editor or feature to build a table, etc. For example, if the user listed \( n = 3 \) preferred features for an application, and the selected supplier supports them, then \( f_s = 1 \). If, however, one of these features is missing (either because the feature is not installed or the supplier does not have such feature), then the number of missing feature \( m = 1 \) and the value of \( f_s \) becomes \( f_s = 1 - m/(n + 1) = 1 - \dfrac{1}{4} = 0.75 \). In general, the user satisfaction with respect to application features is given by:**

\[
f_s = 1 - \dfrac{m}{n+1}
\]

(5)

- **The term \( h_s x^s \) expresses the user’s satisfaction with respect to the change of the supplier and is specified as follows: \( h_s \in (0, 1] \) is the user’s tolerance for change in the supplier. If the value is close to 1 then the user is fine with the change whereas a value that is close to 0 means that the user is not happy with the change. The optimized value of \( h_s \) is given by:**

\[
h_s = \arg \max \left[ \left( \dfrac{c_s + c_{\text{ref}}}{2} \right) c_s \right]
\]

(6)

where \( c_{\text{ref}} \) is a value obtained from equation 4 for replacement supplier. \( x^s \) indicates if change penalty must be considered. \( x^s = 1 \) if the supplier exchange is due to the dynamic change of environment whereas \( x^s = 0 \) if the exchange is instigated by the user.
Similarly, a user’s preferences for QoS dimensions of his applications as given by:

\[ \text{QoS preference} = h_q \cdot c_q \]  

where \( q \in \text{QoS dimension space} \) is a QoS dimension of an application. Note that equations (3) and (7) are almost identical except for the differences in the subscripts and the absence of feature in QoS dimensions.

A feasible configuration is achieved if the user’s task can be realized by appropriate applications that are instantiated using the user’s preferred suppliers and QoS dimensions. The feasible configuration is given by:

\[ \arg \max = \prod_{a \in 1} \text{Supplier preference} (a) \cdot \text{QoS preference} (a) \]  

5. Scenario simulations

Here, we show how the equations derived in the previous section are used in self-optimization of a user task. Consider for example a user on the go (i.e. in a park) working on a text document (work1.txt) and another document MathML (work2.xml) as part of his homework. To do so, our user needs instantiation of these files using appropriate suppliers. Using function \( f_1 \), we can deduce that work1.txt is associated to text processing while work2.xml is associated to a mark-up language software application. Hence, our \( f_1 \) yields:

\[ \text{f1: application associated to a mark-up language software application.} \]

Finding relevant suppliers can be done using function \( f_2 \) which yields the following values:

\[ f_2 = \{ \{ \text{word processing, (MSWord, 1)}, \{ \text{word processing, (NotePad, 2)}, \{ \text{word processing, (WordPad, 3)}, \{ \text{mark-up language application, (Lambda, 1)} \} \} \} \]

Given that supplier priority is involved in \( f_2 \) then the most-preferred supplier is sought. With reference to equation 3, the numerical value associated with user’s preferred suppliers are as follows: (i) priority = 1 (high), user satisfaction = 1, (ii) priority = 2 (medium), user satisfaction = 2/3, and (iii) priority = 3 (Low), user satisfaction = 1/3. In usual case, the user’s preferred suppliers are used to instantiate the applications. Now, consider that one of the user’s preferred supplier, MSWord, is absent as it is not available in the user’s laptop. The method by which the system finds the optimal supplier configuration is shown below:

Case 1: (Lambda, MSWord) is not possible. Case 2: (Lambda, NotePad) is alternative 1. Case 3: (Lambda, WordPad) is alternative 2. Then the replacement selection is based on user satisfaction score: User Satisfaction (Case 2) = \((1+1+\frac{1}{2})/3 = 8/9 = 0.89\). User Satisfaction (Case 3) = \((1+1+\frac{1}{2})/3 = 7/9 = 0.78\).

Hence, Case 2 is the preferred alternative. Formally, if \( f_2: \text{Application} \rightarrow (\text{Supplier}, \text{Priority}) \) where Priority: \( \mathbb{N}, \) then the chosen supplier is given by: \( \exists \text{y}: \text{Supplier} \land \exists \text{p}: \text{Priority} \land (\text{app} \cdot y \rightarrow (\text{y}, \text{p})) \in f_2 \).

5.1. Optimizing user task configuration

Consider a scenario where all word processing suppliers are available. For example, given the suppliers MSWord, NotePad, WordPad, then the corresponding user satisfaction with respect to these suppliers are as follows:

\[ c_{\text{MSWord}} = 1.0, c_{\text{NotePad}} = 2/3, c_{\text{WordPad}} = 1/3. \]

This indicates that the user is most happy with the top-ranked supplier (MSWord) and least happy with the bottom-ranked supplier (WordPad). Consider further that these suppliers have \( n = 3 \) preferred features (e.g. equation editor, spelling checker, bibliography editor). If in MSWord set-up, the equation editor is not installed, then the missing feature \( m = 1 \) and the user satisfaction becomes \( f_s = 1 - m/(n + 1) = 1 - \frac{1}{4} = 0.75 \). This also reduces the user’s satisfaction, as given by the relationship \( c_{\text{MSWord}} \cdot f_{\text{MSWord}} = (1.0)(0.75) = 0.75 \).

Now, consider a case of a dynamic reconfiguration wherein the default supplier is to be replaced by another. Not taking \( f_s \) into account (assumption: \( f_s = 1 \), if the current supplier is NotePad, then the user’s satisfaction is \( c_{\text{NotePad}} = 2/3 = 0.67 \). What would happen if it will be replaced by another supplier through dynamic reconfiguration (\( x_{\text{supplier}} = 1.0 \))? Using the relationship \( h_{\text{supplier}} = (c_{\text{supplier}} + c_{\text{replacement}})/2 \cdot c_{\text{supplier}} \) then the results of possible alternative configurations are as follows:

Replacing NotePad (supplier 2). Case 1: Replacement by MSWord (supplier 1): \((0.67)(1) \cdot [(0.67 + 1)/2*(0.67)] = 0.835 \). Case 2: Replacement by itself (supplier 2): \((0.67)(1) \cdot [(0.67 + 0.67)/2*(0.67)] = 0.67 \). Case 3: Replacement by WordPad (supplier 3): \((0.67)(1) \cdot [(0.67 + 0.33)/2*(0.67)] = 0.50 \). Hence, if the reconfiguration aims at satisfying the user, then the second-ranked supplier should be replaced by the top-ranked supplier.

In a similar fashion, QoS dimensions are given scores for their priority ranking. With characters per line as QoS parameter to a text editor, then \( C_{115 \text{ characters per line}} = 1.0, C_{100 \text{ characters per line}} = 0.67, C_{80 \text{ characters per line}} = 0.33 \). Indeed, the feasible configuration for a text editor application is given by:

\[ \arg \max_{\text{Text Editor}} = (f_{\text{MSWord}} C_{\text{MSWord}})(f_{115 \text{ characters per line}}) = 1.0 \]
5.2. Simulation results

Using user’s preferences, we simulated variations in user’s satisfaction as these preferences are modified through dynamic configuration. The results are presented through various graphs in Fig. 3 (Right). The first three graphs deal with application supplier, and the variation of user’s satisfaction as additional parameters (supplier features and alternative replacements) are taken into account. The last two graphs deal with QoS dimensions and their variations. In general, user is satisfied if the supplier and its desired features and QoS dimensions are provided. Whenever possible, in a dynamic configuration, the preferred setting is one where the parameters are those of user’s top preferences.

Fig. 3. (Left) The training process for ML knowledge acquisition: (Top) the mapping of data type to an application, (Middle) building a user’s preferred supplier list, and (Bottom) building a user’s preferred QoS dimensions for application. (Right): Various graphs showing variations of user’s satisfaction with respect to its preferred supplier and QoS dimension and their replacements.

6. Conclusion

Pervasive computing allows anytime, anywhere computing allowing an end user to continue working on an interrupted task whenever and wherever he wishes. In this regard, we advocate pervasive task, one that follows the user. Access to wired and wireless connection is essential to the realization of pervasive computing. A pervasive task can only be meaningful if the end user can sense that he is actually continuing on an interrupted computing task using a computing setting that looks almost exactly the same with that of what he left behind earlier. To realize this, the task which is comprised of various applications should be mapped to the supplier that the user prefers the most and set-up with QoS dimensions that suit his needs. This paper intends to realize that. Through machine learning training (detail not presented in this paper), we illustrated the acquisition of positive examples to form user’s preferred suppliers and QoS dimensions for selected applications. In a rich computing environment, alternative configuration spaces are possible which give the user some choices for configuring the set-up of his application. We have illustrated that configuration could be dynamic or user-invoked, and the consequences, with respect to user’s satisfaction, of these possible configurations. Optimization is achieved if the system is able to configure set-up based on user’s preferences.

References

[1] M. Weiser, “The computer for the twenty-first century,” Scientific American, vol. 265, pp. 94 - 104, September 1991.
[2] M. Weiser, “Some computer science issues in ubiquitous computing,” Communications of the ACM, vol. 36, pp. 74-84, 1993.
[3] M. Weiser and J. S. Brown, “Designing Calm Technology,” Powergrid Journal, vol. 1, pp. 94 - 110, 1996.
[4] Z. Wang and D. Garlan, “Task-Driven Computing,” Carnegie Mellon University, Pittsburg, PA, 2000.
[5] M. D. Hina, A. Ramdane-Cherif, and C. Tadj, “A Context-Sensitive Incremental Learning Paradigm of a Ubiquitous Multimodal Multimedia Computing System,” WiMob 2005, IEEE Intl. Conf. on Wireless and Mobile Computing, Networking and Communications, Montreal, Canada, 2005, pp. 104-111.
[6] A. Awde, M. D. Hina, Y. Bellik, A. Ramdane-Cherif, and C. Tadj, “Task Migration in a Pervasive Multimodal Multimedia Computing System for Visually-Impaired Users,” in GPC 2007 - 2nd Int. Conf. on Grid and Pervasive Computing, Paris, France, 2007, pp. 459 - 471.
[7] M. Satyanarayanan, “Pervasive Computing: Vision and Challenges,” IEEE Personal Communications, vol. 8, pp. 10-17, August 2001.
[8] O. Buffet, A. Dutech, and F. Charpillet, “Incremental reinforcement learning for designing multi-agent systems,” Montreal, Que., Canada, 2001, pp. 31-32.
[9] P. Horn, “Autonomic Computing: IBM’s Perspective on the State of Information Technology,” IBM Research, 2001.
[10] M. D. Hina, C. Tadj, A. Ramdane-Cherif, and N. Levy, “Autonomic Communication in Pervasive Multimodal Multimedia Computing System,” in Autonomic Communication, A. Vasilakos, M. Parashar, S. Karnouskos, and W. Pedrycz, Eds., ed Berlin, Heidelberg, New York, Tokyo: Springer, 2009.
[11] M. Satyanarayanan, “Accessing Information on Demand at any Location, Mobile Information Access,” IEEE Personal Communications, vol. 3, Feb. 1996.
[12] A. Arghavani, E. Ahmadi, and A. T. Haghighat, “Improved bully election algorithm in distributed systems,” IEEE Intl. Conf. on Information Technology and Multimedia, Kuala Lumpur, Malaysia, 2011.