Senior Citizen Health Self-Care Through Adaptive Multiple Computation Cycles

Shi-Kuo Chang
Department of Computer Science
School of Computing and Information
University of Pittsburgh
Pittsburgh, PA 15260 USA
schang@pitt.edu

Abstract: A methodology is proposed for senior citizen health self-care through adaptive multiple computation cycles according to the principles of Slow Intelligence. An experimental system, called the TDR system, is being implemented on Android smart phones and PCs to serve as the test bed for the proposed approach. In this paper we describe the experimental TDR system, its basic components and the initial experimental results. Further experimental as well as theoretical investigations of the proposed approach are discussed.

Keywords: Senior citizen health self-care, patient monitoring, mobile computing, multiple computation cycles, slow intelligence, slow intelligence.

1. Introduction

Health care for senior citizens undoubtedly has become a central concern for any society on the Planet Earth because of the rapidly aging populations of both industrialized and developing countries. It is a central concern for private citizens and governments alike. For the government, its senior citizens are demanding better health care. For private citizens, it is also becoming clear that one cannot totally rely on the government, and self-help is increasingly necessary and attractive to supplement or even replace what the government can or cannot do.

Can the private citizens help themselves? It is not just a moral imperative or a philosophical principle. As people live longer and the demands for senior citizen health care increase steadily, there will not be enough resources to provide all senior citizens with quality health care even for the industrialized nations, not to mention the developing ones. Private citizens must be able and willing to help themselves. Thankfully with the advances in technologies, there are opportunities to bring self-administered health care and patient monitoring to reality, thus lowering the resources needed and keeping the costs down in providing senior citizen health self-care. We believe this issue is so important that all private citizens should be concerned and actively involved. It cannot be left to physicians, medical societies or pharmaceutical companies.

In this paper, an approach is presented for senior citizen health self-care through

adaptive multiple computation cycles according to the principles of Slow Intelligence. An experimental system, called the TDR system, is being implemented on Android smart phones and PCs to serve as the test bed for the proposed methodology. We will describe the experimental TDR system, its basic components, and the initial experimental results. Further experimental and theoretical investigations of the proposed approach are discussed.

This paper is organized as follows. Section 2 describes the multiple computation cycles for the TDR system. Section 3 presents the TDR interface. Section 4 illustrates the Chronobot database and the basic computation cycle using BrainWave headset. Section 5 discusses two other basic computation cycles using ImageEmoDetect and VoiceEmoDetect. Section 6 introduces patterns and explains how to extract events from data. In Section 7 the integration of EventGraphs for visualization by the end user (senior citizens, patients, physicians and health service providers) will be described. Section 8 is the only section discussing the theoretical formal model. The related works, and discussions of further experimental and theoretical investigations, are presented in Sections 9 and 10, respectively.

2. Computation Cycles for The TDR System

The TDR system is the experimental system for senior citizen health self-care. The three letters T, D and R stands for Heaven (Tian in Chinese), Earth (Di in Chinese) and Man (Ran in Chinese). The main characteristics of TDR system is that it has multiple computation cycles, which is typical of many Slow Intelligence Systems (SIS). The basic tenet of the Slow Intelligence System is that the performance of an SIS is gradually improved through multiple computation cycles. For the TDR systems, it implies the constant and cyclic interactions among Heaven, Earth and Man.

For the TDR system, the Level I computation cycle is to gather input data and store both raw data and easy-to-recognize attributes into the database. As the cost of sensors and interfaces to smart phones and PCs is decreasing following Moore's Law, the Level I computation cycle can be implemented using such low cost devices and processors. For our experimental system, we will use BrainWave headset that can be purchased for less than $100 US dollars. The image and voice capturing devices have already been integrated into the smart phone.

The Level II computation cycle is to extract certain basic events from the database

containing raw input data and simple attributes. Such basic events can be visually displayed as colorful icons in the carousel on the screen of the smart phone.

The Level III computation cycle is to combine the basic events into more complex events. At the same time, such complex events can also be visually displayed in the carousel on the phone screen by coloring the icons corresponding to the complex events.

Level III computation cycle may be repeated multiple times, so that the more complex events are formed and visually displayed by changing their color and presentation characteristics.

The raw data and simple attributes are simple entities consisting of (attribute name, attribute pairs). The extraction of events from such simple entities are based upon certain user-defined patterns. The extracted events can be represented as directed graphs (di-graphs) so that such di-graphs can be combined with other di-graphs to form more complex di-graphs. Therefore it is called EventGraphs. We can then apply the techniques of graph querying to query the knowledge base.

3. The TDR Interface

In this section we explain how the end user (senior citizens, patients, physicians and health service providers) can interact with the TDR system through the TDR interface, which is a web service supported by most browsers and can run on any smart phone or any PC. The TDR interface allows a user to log into the TDR system, display previously recorded data, input new data from various sensors such as the brainwave headset (see Figure 1) or the voice and image input data captured by a smart phone (see Section 5), and last but not least extract significant events (see Section 6).

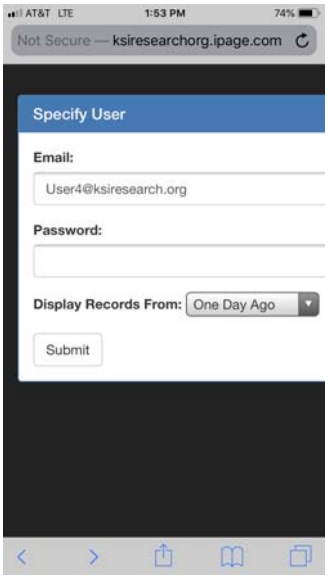
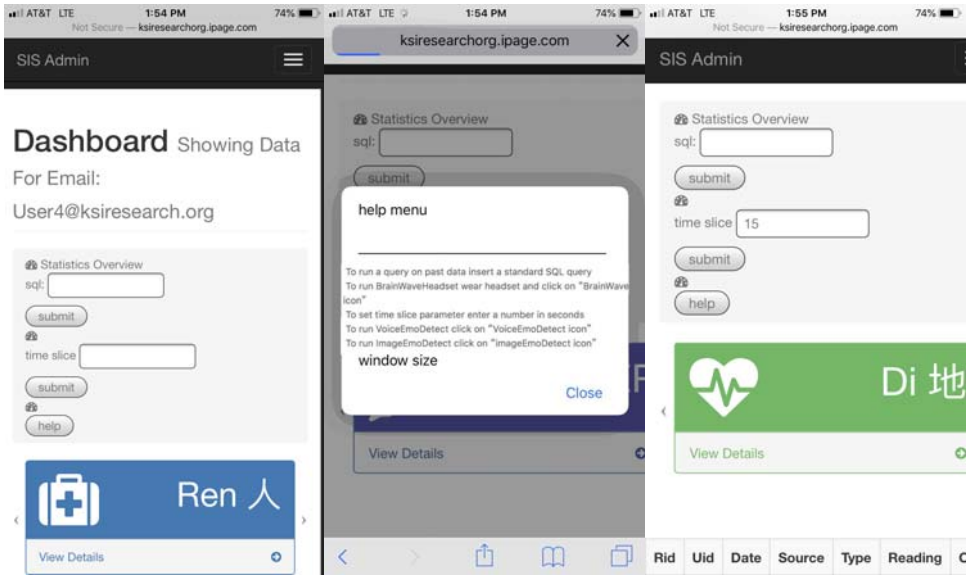


Figure 1. Initial screen of the TDR interface. Its primary purpose is to display records in a certain time period to the end user. It has been expanded to support the input of various sensor data and the extraction of significant events.

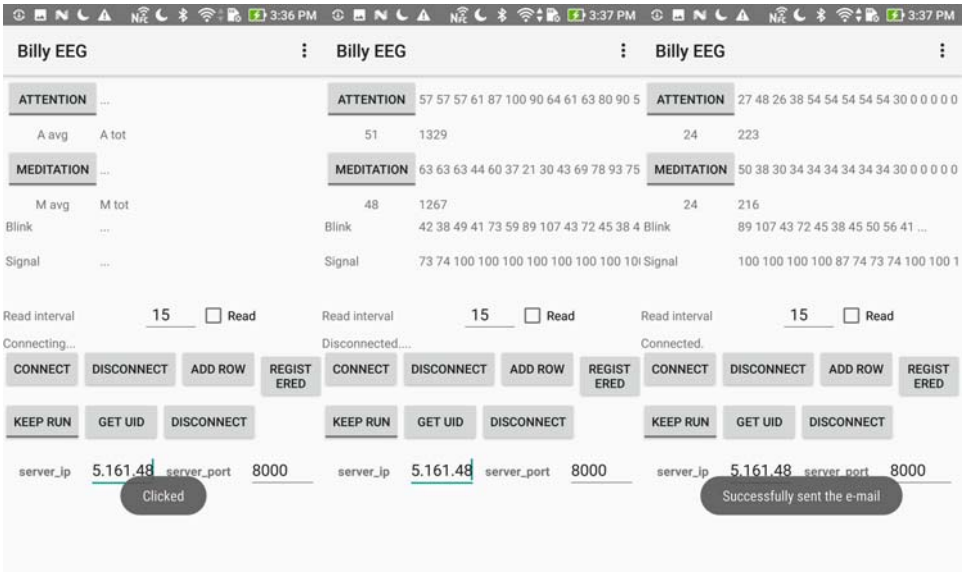
With the help of the TDR interface, a user can specify certain control parameters such as the time slice interval, or to run an Input Processor component (see Figure 2).



(a) (b) (c)

Figure 2. The Dashboard allows the user to specify a latent SQL query or a time slice parameter (a). For the novice user a help page explains the various parameters (b). As an example the user may enter 15 seconds as the duration of the time slice and leave the latent query blank (c).

On his/her smart phone the user can click on the BrainWave icon to run the brainwave Input Processor to access the brainwave sensor to continuously input data in 15 seconds time slices (see Figure 3).



(a) (b) (c)

Figure 3. The user clicks on 'connect' to connect the Input Processor to the brainwave sensor (a). The input data are processed into 'Attention' and 'Meditation' attributes (b). The Uploader then sends these attributes and raw data by e-mail to the SIS Server of the TDR system (c).

The Input Processor on the smart phone would input data continuously until the user clicks on 'disconnect' to stop it. The TDR system saves the uploaded brainwave

data in the ‘records’ relation of the ‘chronobot’ database (see Figure 4).

1292	666666	2019-06-27 15:36:52	BrainWave	Meditation	16	headset
1288	666666	2019-06-27 14:09:37	BrainWave	Attention	61	headset
1290	666666	2019-06-27 14:09:37	BrainWave	BrainWaveData	63|51|50|50|69|64|74|63|51|63|40|56|56|63|90||53|4...	headset
1289	666666	2019-06-27 14:09:37	BrainWave	Meditation	64	headset
1285	666666	2019-06-27 14:09:22	BrainWave	Attention	63	headset
1286	666666	2019-06-27 14:09:22	BrainWave	Meditation	69	headset
1287	666666	2019-06-27 14:09:22	BrainWave	BrainWaveData	63|53|64|77|77|80|66|57|53|63|77|80|90|78|74||56|4...	headset
1283	666666	2019-06-27 14:09:06	BrainWave	Meditation	53	headset
1284	666666	2019-06-27 14:09:06	BrainWave	BrainWaveData	48|35|53|64|60|60|48|51|64|77|87|69|60|54|48|53||3...	headset
1282	666666	2019-06-27 14:09:06	BrainWave	Attention	46	headset
1281	666666	2019-06-27 14:08:51	BrainWave	BrainWaveData	100|88|83|84|74|84|70|50|51|40|54|63|81|64|44||75|...	headset
1279	666666	2019-06-27 14:08:51	BrainWave	Attention	50	headset
1280	666666	2019-06-27 14:08:51	BrainWave	Meditation	51	headset
1278	666666	2019-06-27 14:08:36	BrainWave	BrainWaveData	43|40|51|44|40|40|40|40|38|40|61|61|81|81|88||47|5...	headset
1277	666666	2019-06-27 14:08:36	BrainWave	Meditation	43	headset
1276	666666	2019-06-27 14:08:36	BrainWave	Attention	43	headset
1273	666666	2019-06-27 14:08:20	BrainWave	Attention	40	headset
1274	666666	2019-06-27 14:08:20	BrainWave	Meditation	38	headset
1275	666666	2019-06-27 14:08:20	BrainWave	BrainWaveData	53|53|53|56|54|43|43|43|43|47|47|41|41|41|41|41|43...	headset
1270	666666	2019-06-27 14:08:05	BrainWave	Attention	19	headset
1271	666666	2019-06-27 14:08:05	BrainWave	Meditation	21	headset

Figure 4. The BrainWave data from the brainwave headset consists of the Meditation (attribute, value) pair, the Attention (attribute value) pair, and the BrainWaveData (attribute value) pair where the value is a long string of numbers. For the same BrainWave instance these three (attribute, value) pairs must have the same time stamp.

4. The Chronobot Database

The Chronobot Database consists of the following four relations. The ‘records’ relations stores the raw data and extracted attributes from various sensors (see Figure 5), where only one data instance is shown:

rid	uid	datetime	source	type	value	originator
233	666666	2019-05-09 17:49:47	BrainWave	Attention	66	headset
234	666666	2019-05-09 17:49:47	BrainWave	Meditation	52	headset
235	666666	2019-05-09 17:49:47	BrainWave	BrainWaveData	75|54|38|35|40|35|40|40|41|47|44|44|40|53|53|53|53...	headset

Figure 5. The ‘records’ relation.

The end users of the TDR system is identified by unique uid’s. The users are defined in the ‘users’ relation and managed by the administrator (see Figure 6).

444444		User4234	user4@ksiresearch.org	Victor	Zheng
555555	*NULL*	User5234	user5@example.com	Nannan	Wen
666666	*NULL*	User6234	user6@example.com	Victor	Zheng
777777	*NULL*	User7234	user7@example.com	Kyrie	Gu
888888	*NULL*	User8234	user8@example.com	*NULL*	*NULL*
999999	*NULL*	User9234	user9@example.com	*NULL*	*NULL*

Figure 6. The ‘users’ relation. The attributes are the user id uid, the group id gid, the password, the e-mail, the first name and the last name.

The patterns that can be applied to extract significant events from the ‘records’ relation are defined in the ‘patterns’ relation. Some examples are presented in Figure 7.

pattern_ID	expression	sql_clause	description
1	previous-instance < current-instance < next-instan...	*NULL*	The pattern is in descending order
2	previous-instance > current-instance > next-instan...	*NULL*	The pattern is in ascending order
101	0 <= well-being < 25	*NULL*	blue
102	25<= well-being < 50	*NULL*	green
103	50<= well-being < 75	*NULL*	yellow
104	75<= well-being < 100	*NULL*	red

Figure 7. The 'patterns' relation stores the patterns. In this example two groups of patterns are illustrated. Those with pattern_ID less than 100 are used to describe BrainWave patterns whose instances are either in descending order or in ascenting order. Those with pattern_ID greater than 100 are used to define the range of colors of icons that appear in the carousel of the TDR user interface, with 'blue' indicating a peaceful state, 'red' an agitated state, 'green' and 'yellow' intermediate states.

Finally, the 'events' relation stores the significant events extracted from the 'records' relation according to rules defined in the 'patterns' relation. Two examples of EventGraphs are illustrated in Figure 8.

EventGraph_ID	node_ID	node_value	previous_nodeID	pattern_ID	strength range from 0 to 1	temstamp
1	1	75	*NULL*	1	0	2019-05-09 17:49:47
1	2	54	1	1	0	2019-05-09 17:49:47
1	3	38	2	1	0	2019-05-09 17:49:47
1	4	35	3	1	0	2019-05-09 17:49:47
6	1	0	*NULL*	101	0	2019-06-24 19:52:37
6	2	0	1	101	0	2019-06-24 19:52:42

Figure 8. The 'events' relation stores significant events extracted from the 'records' relation.

The events relation stores EventGraphs. As illustrated in Figure 8 EventGraph 1 has four nodes 1, 2, 3 and 4, with node 1 connected to node 2, node 2 connected to node 3 and node 3 connected to node 4 (see Figure 12). This EventGraph_1 is detected from the input data at time stamp 2019-05-09 17:49:47 according to pattern 1.

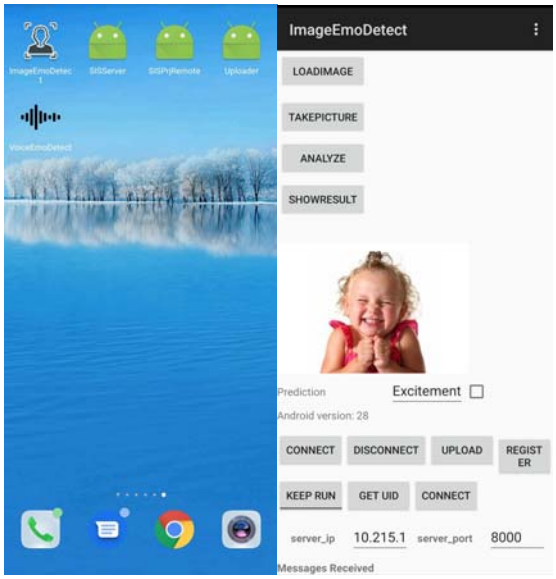
In the TDR system, each extracted event is defined by an event graph. Different event graphs can be combined, and at the same time the visual icons in the carousel of the TDR interface can also be combined at a higher level, thus resulting in a different color. This is the concept of computation cycle, which will be discussed in Section 7.

5. Image and Voice Input Processors

In addition to the BrainWave Input Processor, there can be other input processors such as the ImageEmoDetect and the VoiceEmoDetect input processors. The user clicks on  or  to run ImageEmoDetect or VoiceEmoDetect, respectively.

The ImageEmoDetect Input Processor is capable of taking photo input and sending it

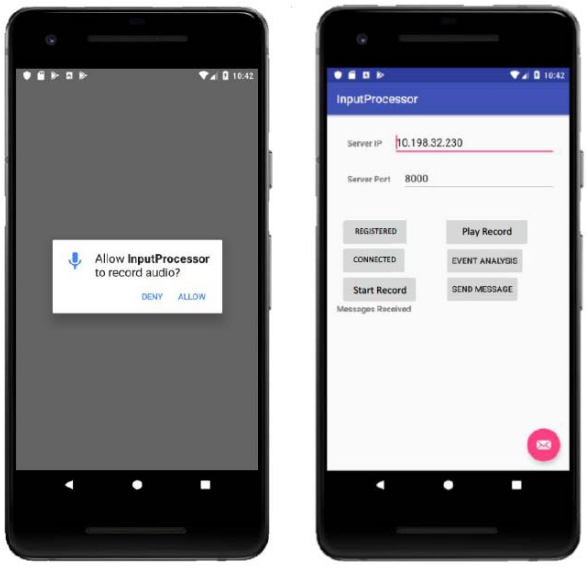
to the Uploader to e-mail the image input to the SIS server to be stored in the Records relation (see Figure 9 and Figure 11).



(a) (b)

Figure 9. The user clicks on ImageEmoDetect icon (upper left corner in Figure 9a) to run this app, which takes live photos or access stored images and analyzes the input data to extract certain attributes such as Excitement, Amusement, Sadness. These attributes may have a certainty value between 0 and 1 (Figure 9b).

The VoiceEmoDetect Input Processor is capable of recording voice input and sending it to the Uploader to send the voice input to the SIS server to be stored in the Records relation (see Figure 10 and Figure 11).



(a) (b)

Figure 10. The user clicks on VoiceEmoDetect icon (middle left corner of Figure 9a) to run this app, VoiceEmoDetect records voice input (Figure 10a) and analyzes the voice input data to extract certain attributes such as VoiceEmoReg, which may have the value ‘Happy’, ‘Angry’ and so on (Figure 10b).

rid	uid	datetime	source	type	value
1372	888888	2019-06-28 18:26:18	Image	Amusement	0
1373	888888	2019-06-28 18:26:18	Image	Sadness	0
1374	888888	2019-06-28 18:26:18	Image	ImageData	iVBORw0KGgoAAAANSUhEUgAAAPoAAAEHCAIAAAAI9cwZAAAAA3...
1375	888888	2019-06-28 18:27:13	Image	Amusement	0
1376	888888	2019-06-28 18:27:13	Image	Sadness	0
1377	888888	2019-06-28 18:27:13	Image	ImageData	iVBORw0KGgoAAAANSUhEUgAAAIYAAAAFvCAIAAADxE5coAAAAA3...
1436	666666	2019-07-01 04:25:16	Audio	AudioData	],???? S ??:??? [0????x?????$?a`???...
1437	666666	2019-07-01 04:25:35	Audio	AudioData	
1438	666666	2019-07-01 04:25:35	Audio	Natural	0.6

Figure 11. The image input data and associated attributes or voice input data and associated attributes are uploaded and stored in the Records relation of the Chronobot database.

6. Extracting Events from Data

Suppose we want to extract an event of decreasing sequence of data, we can specify such a pattern:

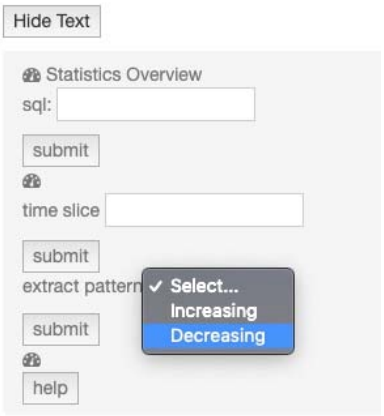


Figure 12. A pattern of decreasing sequence of data.

If the there is a decreasing sequence of data then an Event is detected, which is basically a linear graph with a sequence of nodes, and the edge being labeled by 'decrease' as the trend. Similarly for increasing sequence and so on. The EventGraphs are di-graphs that can serve as primitive building blocks for more complex EventGraphs. The basic algorithm could be exhaustive analysis by using a sliding window. The algorithm basically has the following steps:

Step 1. Get data from the database relation, because the raw data was stored in an array separated by "|", so after read the data, use explode function to split them into an array.

Step 2. Iterate through the array to get all the subarrays that are in descending order, then add them to events relation, with pattern type id = 1.
Step 3. Iterate through the array to get all the subarrays that are in increasing order, then add them to events relation, with pattern type id = 2.
Step 4. Display the raw data we get from the records relation.
Step 5. Visualize the resultant EventGraphs.

The visualization of an EventGraph is illustrated in Figure 13.

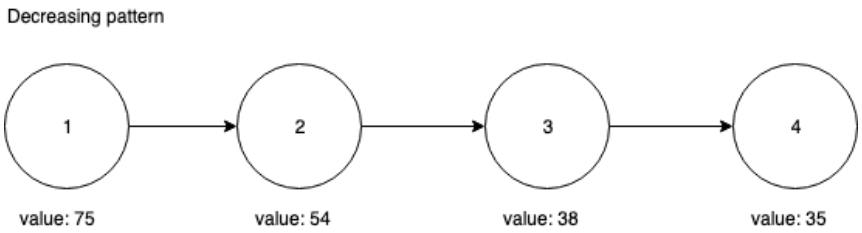


Figure 13. Visualization of an EventGraph from the Events Relation in Figure 8.

7. Changing the Color of Icons for the Carousel

In Figure 7, the patterns 101, 102, 103 and 104 are the rules governing the color changes of carousel icons. The attribute 'well-being' refers to the combined value from many attributes obtained in a computation cycle. For example it may be the average of a time sequence of Meditation attributes (or the average of a time sequence of Attention attributes) from the BrainWave data. Using this combined value, the well-being of the user is estimated. If the well-being value is high, the BrainWave icon's color is changed to 'blue', as shown in Figure 14.

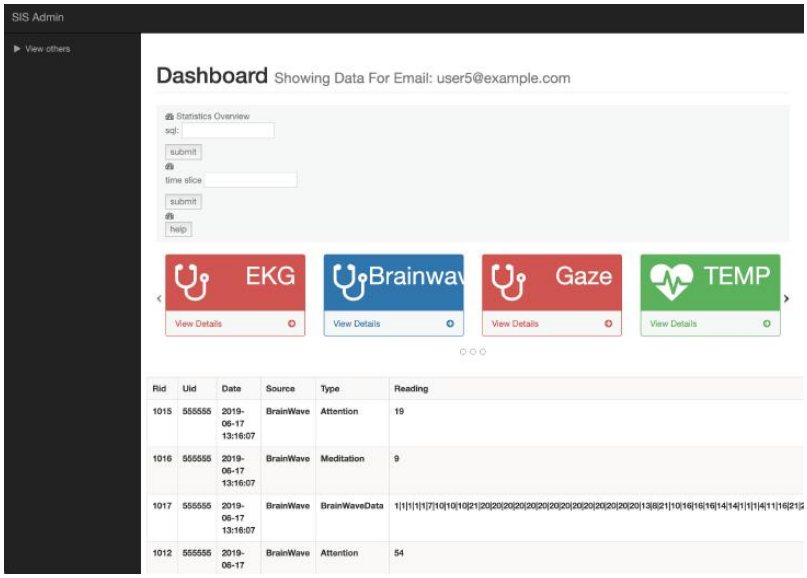


Figure 14. An example of changing the color of BrainWave icon to "blue".

8. Formal Model for the Computation Cycles

The basic computation cycle consists of enumeration and elimination of solutions. For example if we are trying to detect the occurrence of a pattern in the data, we can enumerate all possible matches until we find the best match. This is true for nearly all computation cycles at different levels.

The Abstract Machine Model is specified by: (P, S, P0, $Cycle^1$, ...,, $Cycle^n$), where

P is the non-empty problem set,

S is the non-empty solution set, which is a subset of P,

P0 is the initial problem set, which is a subset of P,

$Cycle^1$, ...,, $Cycle^n$ are the computation cycles.

Each computation cycle will start from an initial problem set and apply different operators such as $+adap_{Aij}=$, $-enum<$, $>elim-$, $=prop_{Aij}+$ and $>conc=$ successively to generate new problem sets from old problem sets, until a non-empty solution set is found. If a non-empty solution set is found, the cycle is completed and later the same computation cycle can be repeated. If on the other hand no solution set is found, a different computation cycle is entered.

As an example the problem set P consists of problem elements p1, p2, p3, ..., pn, and each problem element pj is specified by a vector consisting of attributes Aij. A computation cycle x will attempt to find a solution set by first adapting based upon input from the environment:

P^x0 +$adap_{Aij}$= P^x1 is to adapt based on attribute Aij, for example, by appending Aij to each element in P^x0 to form P^x1

Then it may try to find related problem elements:

P^x1 -*enum*< P^x2 where P^x2 = { y: y is related to some x in P^x1, e.g. d(x,y) < D}

Next it may try to eliminate the non-solution elements:

P^x2 >*elim*- P^x3 where P^x3 = {x: x is in P^x2 and x is in S}

Finally the solution elements (or alert messages if there are no solutions) may be propagated to peers:

P^x3 =$prop_{Aij}$+ P^x4 is to export/propagate attribute Aij to peers

Therefore this computation cycle can be specified succinctly as follows:

$Cycle^x$ [guard x,y]: P^x0 +$adap_{Aij}$= P^x1 -*enum*< P^x2 >*elim*- P^x3 =$prop_{Aij}$+ P^x4

The above expression is a specification of the computation cycle, not a mathematical equation. This expression should be read and interpreted from left to right. If P^x4 is non-empty, the Abstract Machine will complete this cycle of computation and terminate at the end of $Cycle^x$, and it may later resume at the beginning of $Cycle^x$. Otherwise P^x4 is empty and the Abstract Machine will jump to a different $Cycle^y$. This is specified by [guard x,y] where x is the current computation cycle if a solution set is found (P^x4 is non-empty), and y is the computation cycle to enter if no solution set is found (P^x4 is empty). Before an Abstract Machine completes its current computation cycle, it will propagate the solution set (or alert messages) to its peers.

In the above, the elimination operator can be replaced by the concentration operator, whenever the solution set is not known apriori. The concentration operator applies a predefined threshold to filter out problem elements below the threshold.

P^x1 >*conc*= P^x2 where P^x2 = {x: x is in P^x1 and th(x) above a predefined threshold t}

For the TDR system, a problem element is a combination of Tian, Di and Ren attributes. Those problem elements that are favorable for human survival are in the solution set S. The problem set P consists of problem elements p1, p2, p3, ..., p^n, and each problem element is specified by a vector consisting of the attributes from Tian (heaven), Di (earth) and Ren (human being), i.e.,

pj = (t1j, t2j, ..., d1j, d2j, ..., r1j, r2j, ...)

For example, the Tian attributes tij are atmospheric variables such as amount of sunlight and water level, the Di attributes dij are residential variables such as ambient temperature and humidity, and the Ren attributes rij are personal health indicators such as blood pressure, spo2 value, heart rate, etc.

pj = (sunlightj, waterlevelj, tempj, humidityj, bloodpressurej, spo2valuej,

heartratej)

Initially some attributes may not be assigned any value and some may already have pre-assigned values. After most attributes have been assigned values one can decide whether the problem element is in the solution set. (The simplest case is that each attribute Aij has a solution range Rj, and if every attribute Aij falls within the solution range Rj then the problem element pj is in the solution set S).

In the TDR system, there are continuous interactions among the three super-components Tian, Di and Ren. Each super-component has its own computation cycle, which is basically the following: Starting from some problem set P0, the super-component first adapts to the input from the environment as well as from other peer super-components. It then tries to find related problem elements by enumeration. After those problem elements not in the solution set have been eliminated either using the elimination operator or using the concentration operator, the termination condition can be tested. The termination condition is expressed by [guard x, y] where Cycle x is the current cycle and Cycle y is the cycle to jump to. Whenever one super-component completes its computation cycle, if a solution is found the computation ends, otherwise the control is transferred to the next super-component. Since there are three super-components, we will have three computation cycles.

The Tian super-component has computation Cycle1:

Cycle1 [guard1,2]: $P^1 0 + \boldsymbol{adap}_{Aij} = P^1 1 - \boldsymbol{enum} < P^1 2 > \boldsymbol{elim} - P^1 3 = \boldsymbol{prop}_{Aij} + P^1 4$

Likewise, the Di super-component has computation Cycle2:

Cycle2 [guard2,3]: $P^2 0 + \boldsymbol{adap}_{Aij} = P^2 1 - \boldsymbol{enum} < P^2 2 > \boldsymbol{elim} - P^2 3 = \boldsymbol{prop}_{Aij} + P^2 4$

Finally, the Ren super-component has computation Cycle3:

Cycle3 [guard3,1]: $P^3 0 + \boldsymbol{adap}_{Aij} = P^3 1 - \boldsymbol{enum} < P^3 2 > \boldsymbol{elim} - P^3 3 = \boldsymbol{prop}_{Aij} + P^3 4$

Notice the three computation cycles together form a higher-level computation cycle. High-level computation cycles are essential for a complex human-centric psycho-physical system such as the TDR system.

9. Related Work

This paper is an integration of the SIS approach, semantic knowledge representation and di-graph queries.

This paper is based upon the theory of Slow Intelligence. Therefore we will cite a few references in this section. The Slow Intelligence System was first introduced by S. K. Chang. It was applied to pet care [1]. The most recent application of the SIS approach can be found in [2].

Next comes the topic of semantic knowledge representation. Semantic knowledge must be transformed into a generalized format for automated reasoning. This is a well researched topic. We will only cite approaches closest to ours. One popular approach is to transform semantic knowledge into frames [3]. Another popular approach is to transform semantic knowledge into relational database by mapping classes, properties and individual instances of ontology to the database schema [4]. This is also the approach that has been adopted in our paper.

Finally, the idea of applying di-graphs to discovering functional dependencies [5] and querying databases based upon dependencies and association rules [6].

10. Discussion

We need to complete the experimental testbed, design individualized experiments to test the efficacy of our system, and observe its application in social networks. On the theoretical side, we will need to prove the multiple computation levels will converge to the solutions.

References:

[1] Shi-Kuo Chang, Wen-Hui Chen, Wen-Chyi Lin and Christopher Lee Thomas, "Application of Slow Intelligence Framework for Smart Pet Care System Design", International Journal of Software Engineering and Knowledge Engineering, Vol. 25, Nos. 9 & 10 (2015) 1429-1442, World Scientific Pub.

[2] Shikuo Chang, Cuiling Chen, Wei Guo and Nannan Wen, "Event-Based Data Input, Modeling and Analysis for Meditation Tracking using TDR System", Proceedings of 2018 DMSVIVA Conference, Hotel Pullman, Redwood City, San Francisco Bay, June 29-30, 2018, 71-82.

[3] S. U. Khan, "Transformation of Semantic Networks into Frames", *International Journal of Innovation, Management and Technology*, 4(1), 21, 2013.

[4] J. Lee, R. Goodwin and R. Akkiraju, "Ontology management for large-scale enterprise systems", in *Web Semantics & Ontology*, pp 91-114, IGI Global, 2006.

[5] Wenfei Fan, Chunming Hu, Xueli Liu, Ping Lu, “Discovering Graph Functional Dependencies”, *SIGMOD2018*, Houston, Texas, ,June 10-15, 2018, 427-439.

[6] Wenfei Fan and Chunming Hu, “Big Graph Analysis: From Queries to Dependencies and Association Rules”, Data Science and Engineering, 2, pp. 36-55, 2017.