A Slow Intelligence System Test Bed Enhanced with Super-Components

Shi-Kuo Chang, Sen-Hua Chang, Jun-Hui Chen, Xiao-Yu Ge, Nikolaos Katsipoulakis, Daniel Petrov and Anatoli Shein
Department of Computer Science
University of Pittsburgh, Pittsburgh, USA
{schang, sec104, juc52, xig34, nik37, dpp14, aus4}@pitt.edu

***Abstract*—The slow intelligence system (SIS) technology is a novel technology for the design of a complex information system that is aware of the environment through multiple sensors and capable of improving its performance over time. In this paper we describe a practical slow intelligence system test bed where super-components can be specified to describe interactions among components. These super-components are automatically transformed into time controllers for components so they can be managed by the SIS test bed. We illustrate our methodology on personal healthcare system design using this SIS test bed enhanced with super-components.**

Keywords- slow intelligence system, environment-aware software engineering, super-component, SIS test bed, personal healthcare system.

I. Introduction

The **slow intelligence system** (**SIS**) is a general-purpose system characterized by being able to improve its performance over time in *iterative computation cycles* through a process involving *enumeration*, *propagation*, *adaptation*, *elimination* and *concentration*. An SIS continuously learns, searches for new solutions and propagates and shares its experience with peers [1].

The slow intelligence system (SIS) technology is a novel technology for complex information system design and a base for **Environment-Aware Software Engineering (EASE)**, which is the methodology and practice to design and/or improve a complex information system that is aware of the environment through sensors and capable of improving its performance over time in a changing environment. Such complex information systems have the following characteristics: *connected*, *multiple sourced*, *knowledge-based*, *personalized*, *hybrid* and *prodigious*.

The design of complex information systems faces the following challenges: (1) the operating environment, individual/collective user behavior and underlying technology base of such complex information systems are *constantly changing*; (2) there is *never a stable and static solution* for an "optimal" complex information system; and (3) there are *no general techniques* for the design of complex information systems. We believe that the SIS technology can be exploited to address these challenges.

There are many interesting theoretical issues concerning the design of slow intelligence systems [1]. To make the SIS technology useful to the practitioners, in this paper we describe a practical test bed for Slow Intelligence Systems enhanced with *super-components*, i.e., multiple components that can be activated either sequentially or in parallel and have complex interactions to search for better solutions. Furthermore these super-components can be automatically transformed into time controllers for components so they can be efficiently managed by the SIS test bed.

This paper is organized as follows. Section 2 introduces the essential characteristics of an SIS test bed enhanced with *super-components*. To illustrate our methodology, the essential components and super-components of a personal healthcare system are described in Sections 3 to 7. Background for slow intelligence system is presented in Section 8. Further research issues and applications of the SIS test bed to the design and analysis of *sentient networks* are discussed in Section 9.

II. SIS Test Bed with Super-Components

To design SIS-based systems, a practical SIS test bed is illustrated in Figure 2.1. The SIS test bed is a component-based software system. The center-piece of the test bed is the *SIS server* responsible for specification/creation/management of components and passing messages to/from components in the test bed. This test bed is implemented in Java and can run either under Windows or in the Eclipse environment.

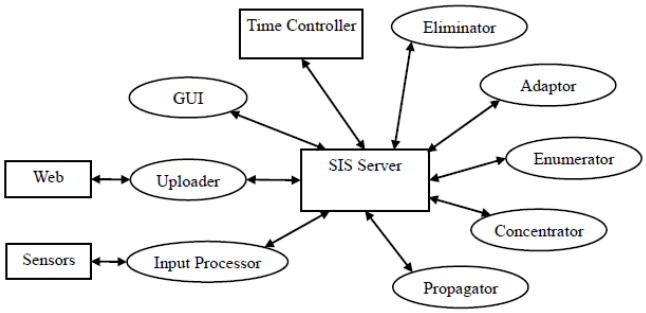

Figure 2.1. A slow intelligence system test bed.

The essential components of the basic SIS test bed include the *Graphical User Interface* (GUI) to interact with the end user, the *InputProcessor* to process input data from sensors and transform them into XML formatted messages, the *Uploader* to upload messages to the Internet, the *Propagator* to communicate with other SISs to propagate information and the *SIS operators suite* (*Enumerator*, *Adaptor*, *Eliminator* and *Concentrator*) to generate, adapt, eliminate and concentrate solutions. For the advanced SIS test bed enhanced with super-

components, there is also a *Time Controller* to initiate and control iterative computation cycles through *guard* predicates. The control and management of heterogeneous sensors requires a slow intelligence system with iterative computation cycles so that different sensors with different characteristics such as resolution, sampling rate, accuracy, etc. can be monitored and properly dealt with. During each computation cycle, the SIS operators suite is employed to optimize the processing of application data obtained from the environment through multiple sensors. The Time Controller determines the invocation and timing of the components in the computation cycles. A super-component is therefore a structured set of SIS operators to perform the computation cycles under the control of the Time Controller. A formal model of the computation cycle is introduced in [10]. To specify and create a super-component, a Create-Super-Component (CSC) message can be sent to the SIS server. An example of the CSC message structure is shown in Table 2.1.

MsgID:21 **Description**: Create Super Component **Example**: Super component specification example

Variables:

- **Passcode** (Passcode of Administrator)
- **SecurityLevel** (Security Level of Administrator)
- **Name** (Name of created Component)
- **SourceCode** (Source code file name of created Component)
- **InputMsgID 1** (MsgID of first input message)
- ...
- **InputMsgID k** (MsgID of last input message)
- **OutputMsgID 1** (MsgID of first ouput message)
- ...
- **OutputMsgID m** (MsgID of last ouput message)
- **Component Description** (Description of created Component in PNML or UML)
- **Component Var 1** (Variable name 1 of created Component)
- ...
- **Component Var n** (Variable name n of created Component)
- **KnowledgeBase** (Name of knowledge base)

Table 2.1. The Create-Super-Component message structure.

In the above CSC message, the **component description** can be in the form of a PNML specification (if the computation model is a Petri net) or an XML document (if UML diagrams such as sequence diagrams, etc. are employed). In Table 2.2 a simple example of a (partial) PMNL specification of a super-component is shown.

After a super-component is formally specified (such as Table 2.2), the Time Controller and other components of the super-component can be automatically generated from this specification. As mentioned above the super-component is controlled by the Time Controller, which sends messages to the constituent components to coordinate their execution. When a computation cycle is completed, the Time Controller decides whether to start another iteration of the computation cycle, or send messages to other super-components (or ordinary components) of the SIS system, depending on the guard predicate.

Multiple super-components with their respective Time Controllers may co-exist in an SIS system and interact with one another. The SIS server enhanced by the SC transformer is illustrated in Figure 2.2. All input messages except the new message 21 to create super-component are sent to the original SIS server. The new message 21 is processed by SC transformer that generates the Time Controller and sends a message 20 to SIS server to create the Time Controller component.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<pnml>
<place id="P0">
<name>
<value>TimeController</value>
</name>
<initialCode>
<value>C:\\Users\\Steve\\Desktop\\SIS\\initialFile.java</value>
</initialCode>
<endCode>
<value>C:\\Users\\Steve\\Desktop\\SIS\\endFile.java</value>
</endCode>
</place>
<place id="P1">
<name>
<value>T1</value>
</name>
</place>
<place id="P2">
<name>
<value>T2</value>
</name>
</place>
<transition id="1001">
<name>
<value>T1toTimeController</value>
</name>
<orientation>
<value>1</value>
</orientation>
<code>
<value>C:\\Users\\Steve\\Desktop\\SIS\\othercode.java</value>
</code>
</transition>
<transition id="1002">
<name>
<value>T2ToTimeController</value>
</name>
<orientation>
<value>1</value>
</orientation>
<code>
<value>C:\\Users\\Steve\\Desktop\\SIS\\othercode.java</value>
</code>
</transition>
<transition id="1003">
<name>
<value>TimeControllerToT1T2</value>
</name>
<orientation>
<value>0</value>
</orientation>
</transition>
</pnml>
```

Table 2.2. A partial PNML specification.

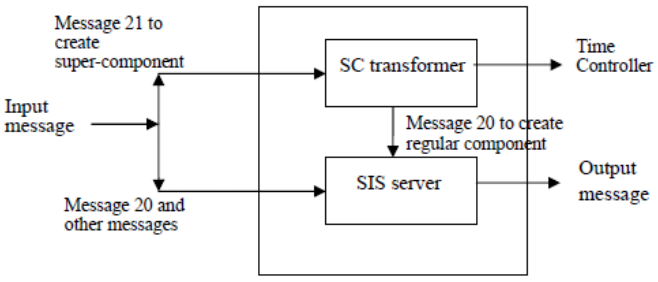

Figure 2.2. A super SIS server.

To illustrate the practical application of this methodology, an experimental personal healthcare system is shown in Figure 2.3. Although this is a specific application it nevertheless exhibits many important characteristics of a complex information system. Foremost among these characteristics is that heterogeneous multiple sensors are constantly changing due to technological advances or other reasons. Each monitor in a personal healthcare system can be a simple component in the simplest case, but more often than not it is a super-component to perform iterative computation cycles for the personal healthcare system. With our approach, the specification and upgrade of a super-components due to technological advances can be easily accomplished.

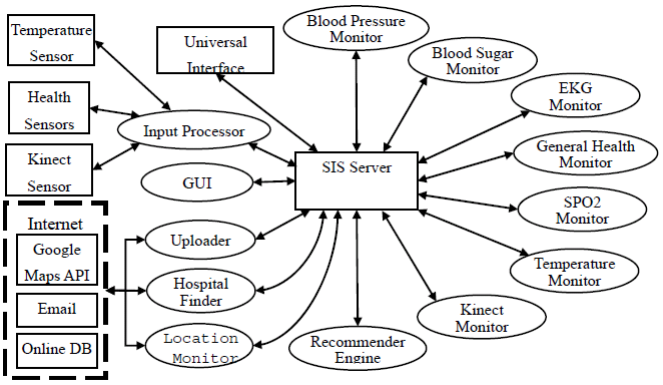

Figure 2.3. A personal healthcare system.

In the following sections we will describe the super-components, monitors and other novel components of the experimental personal healthcare system.

III. Temperature/Blood Pressure Super-Component

A personal healthcare system can assist a senior citizen who may not be computer-literate to deal with various monitors. For example, a *Temperature Monitor* can prevent a senior citizen from suffering from freezing or burning temperatures, and a *Blood Pressure Monitor* can monitor the person's blood pressure. With super-components, these monitors can exchange messages and work together to determine whether there is a need to send an alert message via the Internet to the Emergency Management System (EMS) or the responsible physician in case of an emergency. The situation is illustrated by Figure 3.1, which is a sub-network of Figure 2.3. The interacting monitors are in yellow color.

Once the SIS system is running, the GUI component is launched and the temperature settings such as start-monitor-time, end-time, refresh-time, high-temperature threshold and low-temperature-threshold can be set or adjusted, as shown in Figure 3.2.

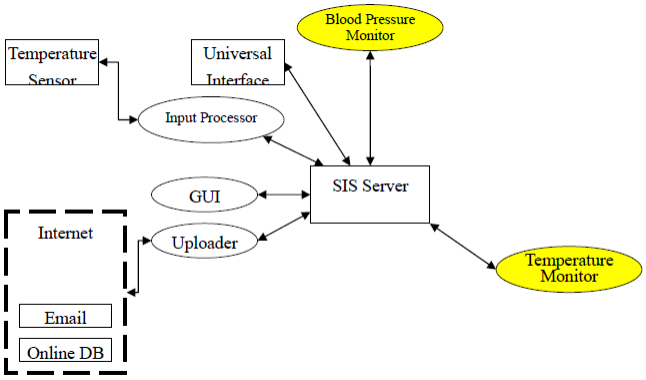

Figure 3.1. Interactions among Temperature and Blood Pressure Monitors.

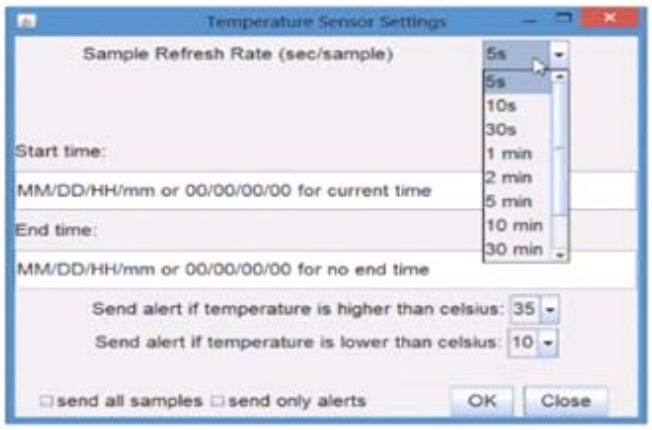

Figure 3.2. Temperature settings.

The Blood Pressure Monitor can then be launched to check whether the person's blood pressure is normal. In addition to working individually, these monitors can work together as a super-component to detect more complex conditions and upload and send *Complex Alert messages* to EMS as shown in Figure 3.3, where the e-mail contains the alert message that the blood pressure may not be normal perhaps due to the rising ambient temperature.

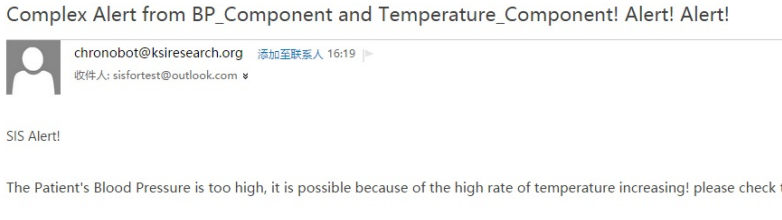

Figure 3.3. Complex alert from Blood Pressure Monitor and Temperature Monitor.

In Figure 3.4, the Petri-net description of a super-component involving the Temperature Monitor and the Blood Pressure Monitor is shown. The corresponding PNML specification can then be transformed into Time Controller to coordinate the interacting components.

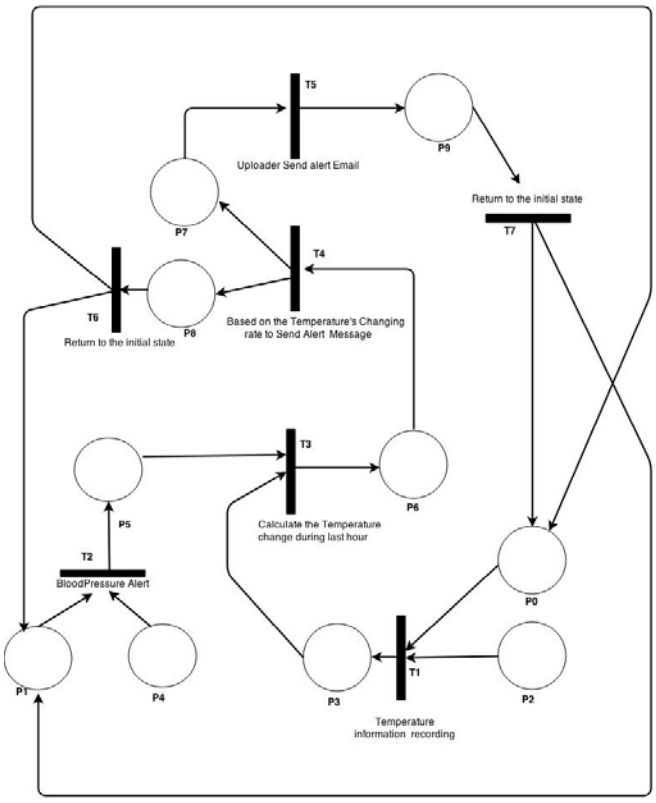

Figure 3.4. The Temperature/Blood Pressure super-component.

IV. Kinect/EKG Super-Component

A Kinect monitor is a component that accepts a series of messages from the Kinect sensor and sends out alerts to certain components when an emergency happens. The Kinect sensor and monitor together can detect and analyze motion patterns such as a person's fall (see Figure 4.1 and Figure 4.2), and the *fall detection* algorithm is incorporated into the Kinect monitor.

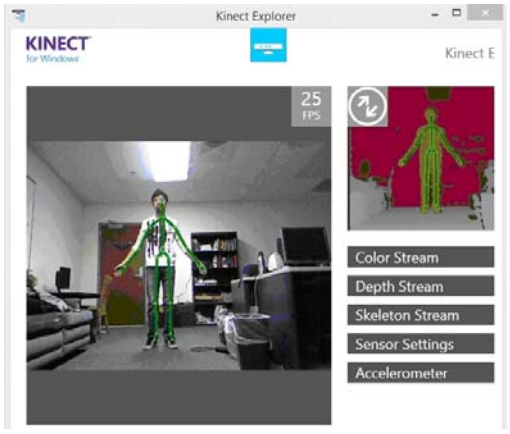

Figure 4.1. Skeleton figure of a person standing.

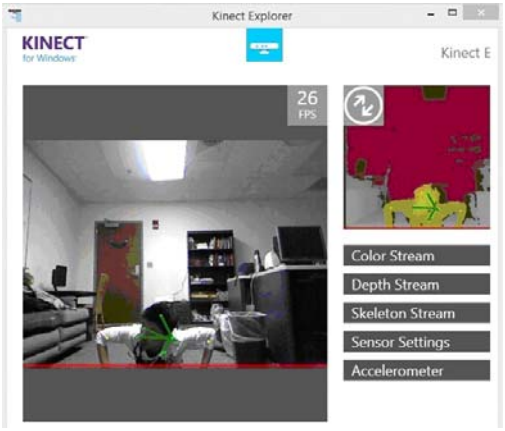

Figure 4.2. Skeleton figure of a person falling.

To achieve fall detection, we need to estimate the real-time position of the person. Kinect SDK software based upon Kinect sensor can track a person's skeleton consisting of more than 20 joints. Once we get a series of skeletons, we can easily extract the coordinates of joints. A frame is generated for each time interval, so we can calculate the difference between two consecutive frames and the speed of movement.

In the experimental personal healthcare system, we try to first track head movement and then estimate the positions of other joints to detect the motion of a person and in particular the fall of a person. When the position of the head cannot be reliably estimated, we can still use other available information to detect the person's movement pattern. We currently don't deal with multiple persons.

Fall detection alone is meaningless if it cannot be propagated and acknowledged by other components. We can send this information to the Uploader, which is the component responsible for collecting information from all other components and informing EMS and the physician in charge, and possibly building a knowledge base along the way. We can also send this information to other monitors responsible for the monitoring of different type of sensors so that they can make more accurate decisions. Likewise the Kinect monitor can also receive information from other monitors and work in a similar way.

As an example of such complex communication, if a person falls and the Kinect monitor also receives alerts from EKG monitor, it could mean this patient not only fell but also suffered heart problems. An alert can be generated either by Kinect monitor or EKG monitor or both, depending on the messaging sequence.

V. Location Monitor

In this section we discuss a Location Monitor for the SIS personal healthcare system, whose objective is to process the information about the location of the person, to track the

person's movements in real-time and to act accordingly in case of an emergency.

Dementia is a broad category of brain diseases. The number of patients, who suffer from dementia, is increasing in the United States of America for the last couple of years. Thus the mortal rate of fatalities, caused by dementia is increasing as well. The most common type of dementia is Alzheimer's disease. Some of the other more popular types are vascular dementia, Dementia with Lewy, Frontotemporal labor degeneration, mixed dementia, Parkinson's disease and Creutzfeldt-Jacob disease. One in three seniors dies with Alzheimer's or another dementia. The statistics shows that 15.4 million caregivers provided an estimated of 17.5 billion hours of unpaid care, valued at more than 216 billion USD in 2012 [2].

The physicians measure what they call Clinical Dementia Rating (CDR), which changes between zero and three or more. The first stage is called CDR =0. There is no impairment for the patient at this stage. The next one is called Questionable Impairment; the value of CDR for it is -0.5. The third one is called Mild impairment and it is the last one, where the patient is capable of taking care of himself/herself. The value for it is 1. At this stage the patient gets geographically lost. For all CDR values beyond that point (2 and 3 – moderate and severe impairment), the person should have a personal caregiver. On the other hand, for all cases of CDR with value 1 or less, an automated monitor, such as the Location Monitor, can take over the responsibility of tracking the movements of the person and his/her location.

The Location Monitor communicates directly with four other components of the SIS Personal Healthcare System – GUI component, Input Processor component, Uploader and Hospital Finder component. The Location Monitor introduces four new messages to the system – GPS Reading, GUI Address Request, GPS Coordinates Request and GPS Coordinates Response. Location Monitor is using two different APIs of Google, Directions API and GmailAPI, to provide the desired functionality. The developers of Google Inc. provided a Java wrapper library for those APIs to be used.

Three basic scenarios involve the Location Monitor. The first one is the supply of data about the current location of the patient. The Input Processor receives the raw data from a sensor, which typically is a smart phone, which has a GPS receiver and some capability of reporting the data, obtained from the receiver, back to the system – Wi-Fi connection, GPRS, WCDMA, LTE or a combination of them. The Health Sensor sends the raw information, marshaled in a Sensor Data Input message. The Input Processor module processes it and sends the information further to the Location Monitor. The next scenario covers the case, when a physician sends the destination of the person to Location Monitor, which starts tracking the person. The third scenario involves the delivery of the last location of the patient, known by Location Monitor, to other components of the SIS system.

Once the Location Monitor receives an Address Request message, it contacts the Google Maps, using Directions API in order to receive a route for the person from his/her current location to the destination, received in the GUI Address Request message. The route contains a polyline, which is the smoothened polyline, which pins the route on the map. The Location Monitor tracks the location of the person for deviations from the received predefined route in the following way - upon every receiving of new location data information (i.e. GPS Reading message), the Monitor determines if the point, representing the received coordinates is within no more than 0.2 miles distance from every rectangle, formed by two consecutive points of the polyline of the route.

In case the location is outside the boundaries, the Location Monitor sends a short text message (SMS) to the person, informing him/her to stay where he/she is and letting him/her know that the help is on the way. An alert message is also sent to the uploader module and a mail is sent to the EMS and the physician on duty.

The Location Monitor was tested in SIS test bed for the Personal Healthcare system. The GPS Reading messages as well as the GUI Address Request messages were emulated with the Universal Interface component.

VI. Hospital Finder

Since there are an increasing number of sensors utilized to monitor the health conditions of senior citizens in today's world, it is possible to receive alert messages instantly when a person is in a dangerous state. This Hospital Finder component reacts to these messages by locating the person on the map, providing directions from the nearest hospitals to the person, and providing the contact information of these hospitals in order to rescue the person in a timely fashion. This component fits well with the rest of the Personal Healthcare System, and provides a useful enhancement that could potentially help save human lives. In what follows we will explain the system model of this Hospital Finder component and give an example of its operation using a real life scenario. The following scenario will utilize the Location Monitor component that monitors the person's location when he or she is out for a walk, and generates alert message 38 if the person's route drastically changes in an unexpected way.

When the Hospital Finder component receives alert message 38, it quickly reacts and opens the map based Graphical User Interface (GUI). The operator who is monitoring the person's health using the SIS system at this point knows that the person is in an emergency state. The operator should be looking for the closest hospitals to deliver help to the patient immediately.

The map-based GUI of the Hospital Finder component offers a variety of ways of locating the person. In this specific scenario, assuming the person has a GPS sensor with him/her, the easiest way to locate the person is by clicking the "Acquire GPS location" button. The other options include locating the

person (a) by address, (b) by coordinates, (c) by position on the map and (d) by GPS coordinates and so on.

In order to place the person's location on the map, the operator clicks the button "Acquire GPS location". The person's location is immediately requested with message 47 (Coordinates Request). The Location Monitor component receives this message and quickly sends the person's last location reading from the GPS sensor back to the Hospital Finder component. The incoming message 48 (Coordinates Response) is handled by the Hospital Finder component by populating the Latitude and Longitude fields on the map-based GUI, and placing an icon of the person on the map:

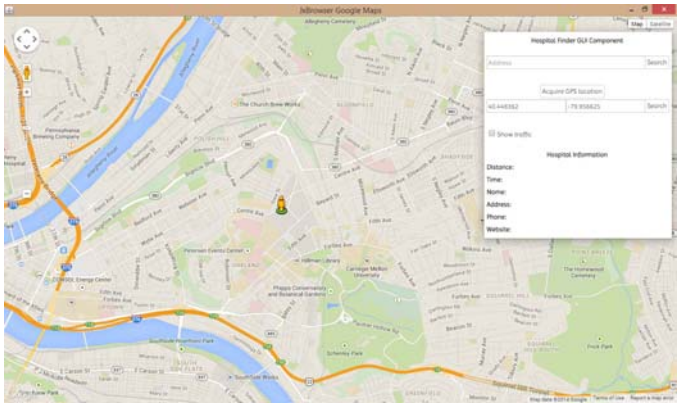

Figure 6.1. The person's last known location is displayed.

Now, since the operator has received the person's location, he or she can click the "Search" button next to the person's coordinates to find the closest hospitals:

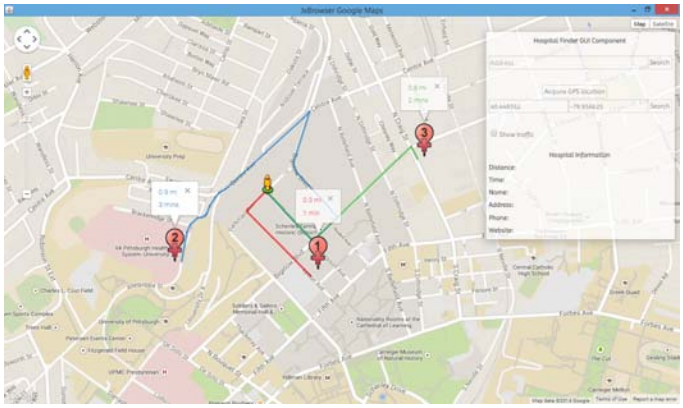

Figure 6.2. Closest hospitals with routes are displayed.

When the search button is clicked, the GUI requests the three nearest hospitals using the Google Maps Places API from the Google database. When the locations are received, the Hospital Finder component requests the directions from Google using the Google Maps Directions API to get to the person from these hospitals. As soon as the directions are received, the Hospital Finder component draws them on the map, and displays an information window above each one of the hospitals stating the distance and the time that it would take to get to the person from each hospital.

VII. Recommender Engine

The Recommender Engine for the Personal Healthcare System is capable of making elaborate decisions and proactively generate alert messages. This way, unwanted situations can be avoided in which the person may be in a state of imminent health deterioration. In what follows we will describe the design of the Recommender Engine, and usage scenarios in a real-life situation.

The Recommender engine waits for messages produced by sensors and mediated by the InputProcessor. Specifically, in its current state the Recommender Engine receives messages from: (a) blood pressure, (b) blood sugar, (c) EKG, and (d) SPO2 sensors. In the event that the Recommender Engine detects an imminent dangerous state, it produces alert messages and disseminates them in the SIS system.

The different components inside the Recommender Engine are as follows. First, the message parsing interface is responsible for handling input messages and produce alerts in the SIS network. The data transformation logic receives sensor data and turns them into a binary form that is readable by the recommender logic. The latter is responsible for building prediction models in order to implement the prediction logic needed for identifying dangerous situations in a proactive manner. Finally, the Recommender Engine keeps an internal storage module for storing user-defined Rules (conditions under which an alert should be generated) and pre-computed Prediction Models. Information are stored in the form of tuples (records) so that the system is able to predict dangerous situations.

The Recommender Engine works by using Collaborative-Filtering Algorithms to predict users' preferences. This approach is more generic compared to the Context-based approach of other recommendation systems. Hence, it can be easily modified to work with different scenarios. The only information to be stored should be tuples of the form:

user-id, item-id, preference

The user-id refers to a user showing interest (or a general connection) for a specific object (item-id). The interest can represent any kind of connection among two entities. For instance, it can represent how much a user likes a product, or it can represent that a user has had a characteristic represented by a specific id (object). Preference models the intensity of the connection of a user with an object. A preference can take values from 1 to 5, but it can also have a binary interpretation if a binary recommendation system is needed (i.e. yes or no answer). By forming the data accordingly, one can approximate any kind of situation and have the Recommender Engine produce successful predictions.

The Recommender Engine can be used when we need to predict dangerous situations for people. For any person we have sensor input for blood pressure, blood sugar, SPO2 and an EKG. The Engine's responsibility is to combine readings

from the aforementioned sensors, and by consulting user-defined rules, produce alert messages to the system. The Apache Mahout library (http://mahout.apache.org), which is a complete framework for recommendation systems, is used in implementation. Three scenarios are presented, each using different rules:

1. A patient is in alert if blood pressure and blood sugar are in near-dangerous levels.
2. A patient is in alert if blood pressure and SPO2 levels are in near-dangerous levels.
3. A patient is in alert if blood pressure, blood sugar and SPO2 levels are in neardangerous levels.

The Alert message produced by the Recommender Engine has the following form:

Name Value MsgID 64
Descriptiom Recommender System Alert
AlertType Recommender Alert
DateTime Current Date (i.e. "2014-10-30 15:05:10")

VIII. Related Work

The slow intelligence approach was first proposed by Shi-Kuo Chang [1]. In this section we will briefly review recently published papers in this area. The visual specification of component-based Slow Intelligence Systems is described in [3]. This work introduces the visual description of super-components by Petri nets or other UML diagrams. It provides the foundation of the present work. Component-based Slow Intelligence Systems has been applied to many areas, including social influence analysis [4, 5], topic and trend detection [6], high dimensional feature selection [7], image analysis [8], swimming activity recognition [9], and most recently pet care systems [10] and energy control systems [11]. In [10] the notion of an abstract machine for computation cycle was introduced. Our current approach is based upon it.

IX. Discussion

The super-component transformation algorithm can be extended to handle parallel/distributed processing of super-components in multiple computation cycles. At the implementation level we introduce one additional pair of tags, <parallel> and </parallel>, into the pnml specification to signify levels of parallel computation. Therefore the SC translator will append the super-component type as the suffix to the message that this transition represents. For example if the original message id is 1002 and a super-component <parallel> 03</parallel> is specified inside the related pair of transition tags of message 1002, then this message will become 1002.03. We can extend this technique to define messagetype.SCsubtype.SCtype and so on, so that messages are exchanged at different levels. At the theoretical level, we envision complex information systems as iterative slow intelligence systems with multiple and interacting computation cycles. In Wiener's Theory of General Resonance, he envisioned the interaction of multiple computation cycles. We can call such general systems *Sentient Nets*. With the above proposed different levels of computation cycles and messages, we propose to continue the investigation of the properties of such general systems.

References

[1] Shi-Kuo Chang, "A General Framework for Slow Intelligence Systems", International Journal of Software Engineering and Knowledge Engineering, Volume 20, Number 1, February 2010, 1-16.

[2] Alzheimer's Disease Facts and Figures, www.alz.org/downloads/facts_figures_2013.pdf, 2013.

[3] Shi-Kuo Chang, Yingze Wang and Yao Sun, "Visual Specification of Component-based Slow Intelligence Systems", Proceedings of 2011 International Conference on Software Engineering and Knowledge Engineering, Miami, USA, July 7-9, 2011, 1-8.

[4] Shi-Kuo Chang, Yao Sun, Yingze Wang, Chia-Chun Shih and Ting-Chun Peng, "Design of Component-based Slow Intelligence Systems and Application to Social Influence Analysis", Proceedings of 2011 International Conference on Software Engineering and Knowledge Engineering, Miami, USA, July 7-9, 2011, 9-16.

[5] Yingze Wang and Shi-Kuo Chang, "User Profile Visualization to facilitate MSLIM-model-based Social Influence Analysis based upon Slow Intelligence Approach", Proceedings of 2014 International Conference on Software Engineering and Knowledge Engineering (SEKE 2014), Vancouver, Canada, July 1-3, 2014.

[6] Ji Eun Kim, Yang Hu, Shi-Kuo Chang, Chia-Chun Shih and Ting-Chun Peng, "Design and Modeling of Topic/Trend Detection System By Applying Slow Intelligence System Principles", Proc. of DMS2011 Conference, Florence, Italy, Aug. 18-20, 2011, 3-9.

[7] Yingze Wang and Shi-Kuo Chang, "High Dimensional Feature Selection via a Slow Intelligence Approach", Proc. of DMS2011 Conference, Florence, Italy, Aug. 18-20, 2011, 10-15.

[8] Shi-Kuo Chang, Li-Qun Kuang, Yao Sun and Yingze Wang, .Design and Implementation of Image Analysis System by Applying Component-based Slow Intelligence System., Proc. of DMS2012 Conference, Miami, USA, Aug. 9-11, 2012.

[9] Wen-Hui Chen and Shi-Kuo Chang, "Swimming Activity Recognition Based on Slow Intelligence Systems", Proc. of SEKE2013 Conference, Boston, USA, June 27-29, 2013.

[10] S. K. Chang, W. H. Chen, Bin Kao, L. Kuang, and Y. Z. Wang, "The design of pet care systems based upon slow intelligence principles," *Int'l Journal of Software Engineering and Knowledge Engineering*, 2014.

[11] Wen-Hui Chen and Shi-Kuo Chang, "Applications of Slow Intelligence Frameworks for Energy-Saving Control", Proceedings of 2014 International Conference on Software Engineering and Knowledge Engineering (SEKE 2014), Vancouver, Canada, July 1-3, 2014.