Updatable Multi-agent OSGi Architecture for Smart Home System

Piotr Jaszczyk[1] and Dariusz Król[2]

[1] Faculty of Computer Science and Management,
Wrocław University of Technology, Poland
piotr.jaszczyk@student.pwr.wroc.pl
[2] Institute of Informatics,
Wrocław University of Technology, Poland
dariusz.krol@pwr.wroc.pl

Abstract. Smart home is an adaptive system that controls house to make household members' life easier. There are plenty of not yet researched issues concerning this concept, such as the ability to update the architecture during system runtime. This paper presents a solution how to design such system on the JADE framework. To solve the problem of on-the-fly update, each agent needs to work in an OSGi environment. The proposal of new architecture is evaluated by means of the use case technique.

1 Introduction

Smart Home can be defined as home *that is able to acquire and apply knowledge about its inhabitants and their surroundings in order to adapt to the inhabitants and meet the goal of comfort and efficiency* [4]. In other words, the role of the system is to help inhabitants in their every-day life by performing some actions instead of them. Furthermore, the system should also take into consideration consumption of the resources (e.g. water and power) in order to minimize costs.

There are many possible types of architecture and topology which can be used to build a Smart Home System (SHS) [2]. Traditionally, it is founded on centrally managed architecture. In this approach there is a set of home appliances which are controlled by a single device (often called home or residential gateway) [7]. However, centralized architecture has many disadvantages [10]: it is unreliable and not scalable. In case of central device failure, the whole system is unable to work. Moreover, that approach was constructed when most of devices have little or even no processing power. Nowadays, many home appliances are equipped in considerable processing power, which fact opens a path to ubiquitous computing.

Smart Home consists of many sensors and effectors, which are distributed over the entire house. All the devices gather information from sensors and react in personalized, trained manner to make changes in their environment. They should be able to work even with incomplete data and when other devices are out of order. Multi-agent systems (MAS) appear to be an obvious choice if requirements of distributed reasoning in heterogeneous environment are presented. Each agent is an autonomous piece of software which has intelligence to accomplish its own design goals, as well as self-configuration, self-optimization, self-healing, and self-protection [9].

There are, however, a plenty of other issues concerning design of SHS. Many of them are presented in [5]. First problem is the fact that researchers design fully-developed systems in top-down manner, despite the fact that in reality people buy new devices continuously. That is why, architecture should provide some mechanisms for system extension. Moreover, it should be able to work even in not fully-developed state. Another challenge is *impromptu interoperability* – ability to interconnect devices produced by different manufacturers. The case of SHS consisting of few subsystems, which are not able to communicate is unacceptable. In addition, household members are not expected to perform complex administration tasks. In fact, SHS should work with as little user interference as possible.

Architecture design should also allow devices to update its components. It means that when new version of software appears, it should be easily and automatically installed during the operation of the system. The requirement of performing administration tasks on-the-fly is specifically important in SHS, which is expected to run continuously without the need to restart the whole system.

This paper advocates the idea that the SHS architecture should provide on-the-fly installation and update of software. It is based on multi-agent JADE framework and OSGi specification. JADE (Java Agent Development Framework) is a platform that simplify implementation of MAS. It is compliant to FIPA (Foundation for Intelligent Physical Agents) specifications, thus ensures that JADE based agents can interoperate with other FIPA-compliant agents. JADE provides programmer with a plenty of features including message interchange, simplified action (behaviour) creation and built-in services like: AMS (Agent Management System), DF (Directory Facilitator), and the ACC (Agent Communication Channel). More information on JADE can be found in [3].

OSGi$^{\text{TM}}$[1] is a widely adapted standard whose basic aim, among many others, is to integrate of multiple components to form a new application. Each component (bundle) is a JAR (Java ARchive) with some meta information about it. It can then be deployed on Java Virtual Machine (JVM) and communicate with other bundles via exposed services. Bundles can be installed and updated on running applications, which empowers systems to extend and update on-the-fly. The whole framework works in a single JVM. OSGi introduces concept of "private packages". It also provides some standardized services, e.g. Log Service, Preferences Service, Http Service, UPnP Service. More information on OSGi and JADE integration can be found in [8] and [11], while on SOA and OSGi integration in [12].

The rest of this paper is organized as follows. Section 2 presents detailed view of proposed architecture. Later on, in Section 3, the architecture is evaluated on the basis of the usage scenarios. Finally, Section 4 presents conclusions and some future research directions.

2 Architecture of SHS

This paper shows how to integrate JADE and OSGi to provide SHS with possibilities of extension, update and self-optimization. The extension allows user to install new devices. The installed devices can be next updated with a new version of software. Having taken into consideration the system extension possibility, SHS should also check if new architecture still optimizes devices' processing power. All these actions are executed during normal system operation.

Architecture consists of three types of devices and is shown in Fig. 1:

– Agent-Less Devices (ALD), which are not equipped with much processing power, thus they cannot contain full JVM with running agents. However, they can serve some services, accessible via FIPA ACL messages or other standardized protocol (KNX, X10). ALD would be usually portable devices powered by batteries and devices in which incorporating much memory and processing power is still not profitable, for example: watches, LCD photo frames, coffee makers, toasters, light bulbs, faucets, TRVs.
– Agent Devices (AD) provide enough memory and processing power to run Java applications. This group includes devices such as television sets, set-top boxes, refrigerators, central heating units.
– Main Device (MD), a special case of AD, which incorporates JADE's Main Container. It can be nearly any AD device which is equipped with the access to the Internet. There can be only one MD in the entire SHS.

ALD are devices which, for various reasons, cannot serve JVM. However, they are capable to communicate with remote agent via some standardized protocol. Each ALD exposes simple service with methods used to manipulate device: turn-on, turn-off, make-coffee, change-channel, heat-up. However, ALD services are accessible only through agent deployed on an AD – the front-end. This agent

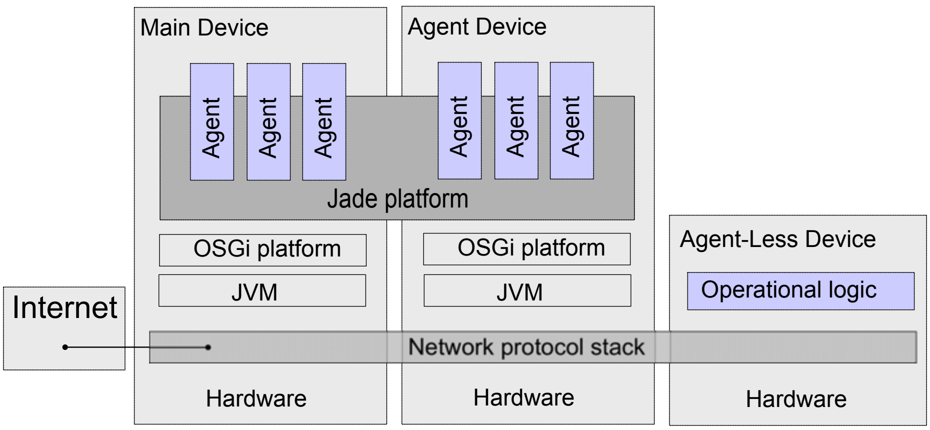

Fig. 1. Proposed architecture of Smart Home System

contains some higher-level logic (such as intelligence and learning) and uses ALD services to accomplish its own goals. On the other hand, ADs and MD are capable to incorporate OSGi platforms. There are some JADE agents deployed on each platform. Each agent should be packaged in a separate bundle, so that the services provided can be placed on different devices. At least one agent is mandatory: Device Management Agent. Originally, device contains only one DMA and its own service agent, for example hi-fi device contains DMA and HiFiAgent. DMA is responsible for managing (installation, update and so on) of agents set up on local OSGi platform. It also checks health of local OSGi platform – monitors memory and CPU usage.

Installation of new device can be divided into three cases: ALD, AD or MD. In first case, after connecting ALD, it sends a message to MD with information that it wants to be a part of the system. MD downloads device's driver (agent bundle with device front-end) from the Internet repository. Next, it asks all ADs if they can do new device's service. The respond is based on the information gathered by DMAs (memory & CPU usage) and, also, on the information sent within the message (requisite memory and processing power, estimated frequency of usage and so on). Then, MD selects one device and asks it to install previously downloaded bundle. Later on, agent registers itself in JADE's Yellow Pages Service and starts to operate by invoking ALD services (via ACL messages or some other protocol). In case of AD, the procedure is rather straightforward. After plugging in, it registers on the Yellow Pages Service. Optionally, it can download updates to its own agent right after installation. The installation of a new MD is similar to the installation of AD. However, the architecture demands presence of only one MD. Therefore, before registration on Yellow Pages Services, device checks whether the other MD is present. If this is the case, it connects itself to existing main container and starts operating as a regular AD.

From time to time, ADs ask MD if there are any updates to the hosted bundles. In this case, MD downloads a new version of the bundle from the repository and serves it through HTTP Service, so that the devices can install it locally.

Another aspect of the proposed architecture is its self-optimization. The information about agents' hardware requirements provided during installation is only an estimation. It can obviously vary in different environments. If device's DMA concludes that it can no longer serve specific agent it sends messages to other devices, whether they can serve the agent or not. The procedure is similar as in the case of installation.

3 Application Scenario

In this section, an example of a system behaviour in particular situations is presented. Firstly, we need to consider the situation when a user buys a new device and tries to install it in the running system. Next, the description of the system's behaviour during normal operation will be provided. The message interchange is described in terms of utterances (communicative acts) and locutions (messages). The detailed presentation of each locution can be found in Section 3.3.

3.1 Installing New Device

In the usage scenario the user tries to connect just-bought bathtub with a running system. We assume that the bathtub is not equipped with much processing power, so it cannot contain the fully functional agent. That is why, the manufacturer also provides the user with a software agent which acts as a front-end to the bathtub's simple services.

Following devices and agents are affected during execution of the described use case (see Fig. 2):

- SD (Selected Device) – a particular device selected during use case execution,
- cDMA (Central Device Management Agent) – an agent deployed on MC, which helps manage (installation, update and so on) devices set up in SHS,
- BD (Bathtub Device) – a device itself (bathtub with faucet),
- BA (Bathtub Agent) – a bathtub agent deployed on the other device (with more processing power),
- DMA (Device Management Agent) – an agent deployed on a particular device, which is responsible for managing (install, update and so on) of agents set up on local OSGi platform,
- HTTP (HTTP service) – an OSGi HTTP service,
- MC (Main Container) – JADE platform's main container device,
- YPS (Yellow Pages Service) – the naming service deployed on MC.

The use case:

1. After connecting BD into power it registers itself in the network (via DHCP or similar protocol). Next, it sends broadcast message to inform cDMA that a new device is now part of the system, see Fig. 3(a).
   $\mathbf{U_1}$: BD $\xrightarrow{\mathbf{L_1}}$ cDMA
2. cDMA downloads from the Internet Repository (placed outside of the system) a device driver. Driver is packaged as an OSGi bundle. cDMA saves bundle in his local storage, which is accessible by other agents via HTTP protocol, see Fig. 3(a).

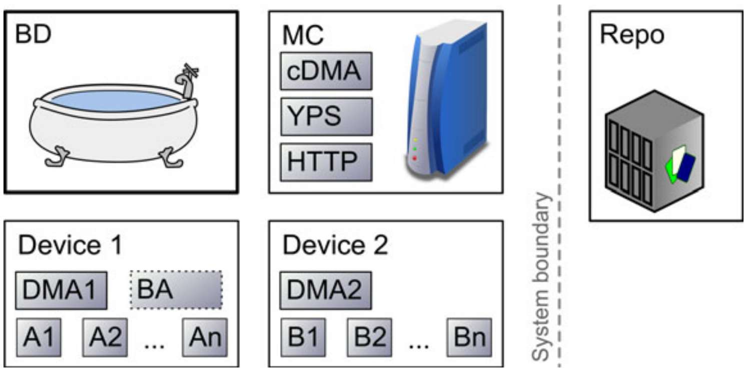

Fig. 2. Installing new device – architecture scheme

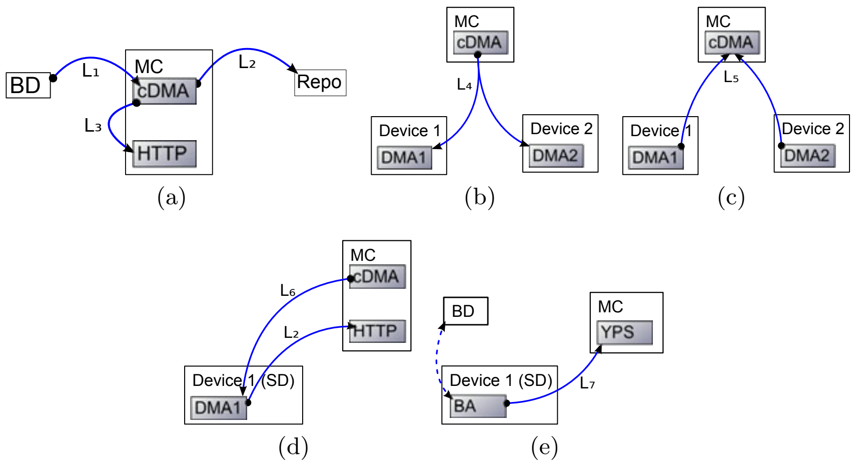

Fig. 3. Messages interchanged during installing new device - use case

$\mathbf{U_2}$: cDMA $\xrightarrow{\mathbf{L_2}}$ Internet Repository

$\mathbf{U_3}$: cDMA $\xrightarrow{\mathbf{L_3}}$ HTTP

3. cDMA selects devices which can act as a bathtub front-end. The choice is based for example on the location. Then, cDMA sends corresponding messages to selected devices' DMAs. Message contains the requisite processing power and estimated frequency of usage, among other things, see Fig. 3(b).

   $\mathbf{U_4}$: cDMA $\xrightarrow{\mathbf{L_4}}$ DMAs
4. After receiving the proposal, agents decide whether they can act as a device front-end or not. The decision is based on supplied information and own state. Next, they reply to cDMA, see Fig. 3(c).

   $\mathbf{U_5}$: DMAs $\xrightarrow{\mathbf{L_5}}$ cDMA
5. If no device agrees to the proposal cDMA sets MC as a selected device. Otherwise, cDMA selects one device from those that replied to the proposal positively. Next, it sends the *acknowledge* massage with information about the driver bundle to the selected DMA, see Fig. 3(d).

   $\mathbf{U_6}$: cDMA $\xrightarrow{\mathbf{L_6}}$ DMA

   $\mathbf{U_7}$: DMA $\xrightarrow{\mathbf{L_2}}$ HTTP
6. DMA downloads a device driver (actually BA) from the repository and installs it into its local OSGi platform. During installation, BA registers itself on Yellow Pages Service. Next, BA can communicate with BD and act as BD's front-end agent, see Fig. 3(e).

   $\mathbf{U_8}$: BA $\xrightarrow{\mathbf{L_7}}$ YPS

3.2 Normal Operation

In this use case, there is a description of a simple scenario involving just-installed bathtub.

Following devices and agents are affected during the execution of the described use case:

- BA (Bathtub Agent) – a bathtub agent deployed on other device than BD,
- BD (Bathtub Device) – a bathtub device (bathtub with faucet),
- ERA (Energy Resource Agent) – an agent which distributes an electric power,
- UPDA (User Presence Detector Agent) – an agent which detects whether specific household member is present at home,
- WRA (Water Resource Agent) – an agent which distributes cold and hot water.

The use case:

1. UPDA detects that a certain person came back home. Then, it informs agents that previously subscribed this kind of information about user's action. One of these agents is BA, see Fig. 4(a).
   $\mathbf{U_9}$: UPDA $\xrightarrow{\mathbf{L_8}}$ BA
2. BA "knows" that the user bathe every time he comes back home. Because all preconditions are met (eg. nobody is bathing) it wants to prepare a bath. To accomplish this task it needs some water. Hence, it asks WRA if it can use some particular amount of water. WRA allows BA to use requested amount of water, see Fig. 4(b)
   $\mathbf{U_{10}}$: BA $\xrightarrow{\mathbf{L_9}}$ WRA
   $\mathbf{U_{11}}$: WRA $\xrightarrow{\mathbf{L_{10}}}$ BA
3. Bathtub is able to produce whirlpools in water, but to turn on a spa program it requires some energy. It has to ask ERA if it can consume particular amount of energy, see Fig. 4(c).
   $\mathbf{U_{12}}$: BA $\xrightarrow{\mathbf{L_9}}$ ERA
   ERA responses, that CMA can use some power, but in less amount than requested.
   $\mathbf{U_{13}}$: ERA $\xrightarrow{\mathbf{L_{10}}}$ BA
   BA decides to use energy-saver spa program (it produces less whirlpools but still has a relaxing property).
4. BA requests BD to prepare a bath using energy-saver program, see Fig. 4(d).
   $\mathbf{U_{14}}$: BA $\xrightarrow{\mathbf{L_{11}}}$ BD

3.3 Locutions in Scenarios

Locutions are simply ACL messages as described in the FIPA specification. Description of particular locutions interchanged during evaluation of both use cases follows:

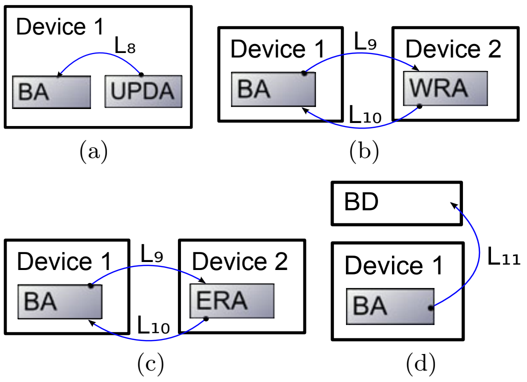

Fig. 4. Messages interchanged during normal operation - use case

- Locution $\mathbf{L_1}$: *introduce*(*info-description*)
  New device introduces itself in SHS. *info-description* is a structure defined in the same way as in the FIPA Device Ontology Specification [6]. It contains a device name, a vendor name and a device version.
- Locution $\mathbf{L_2}$: *get-driver-bundle*(*info-description[, repository-url]*)
  cDMA downloads a bundle from the manufacturer's repository containing the driver (agent) used to manage device described by *info-description*. It is also executed by other DMAs to download bundle from MD's HTTP Service. *repository-url* is needed only for initial download of drivers from the Internet. Next it may be cached and associated with a vendor.
- Locution $\mathbf{L_3}$: *publish-bundle*(*info-description, bundle-file*)
  Publishes a bundle in the local repository, which is accessible by all agents.
- Locution $\mathbf{L_4}$: *propose-serving*(*min-hw-spec, est-usage-freq*)
  cDMA asks other DMAs if they can serve an additional agent. This locution is derived from the *propose* ACL message, thus it also contains parameters defined in the FIPA Specifications [6]. *min-hw-spec* is a structure containing main and external memory usage, requisite processing power and others. It can be based on FIPA's *hw-description*. *est-usage-freq* is number of times device's services will probably be executed in some time period (for example bathtub usage can be estimated as $numberOfUsers/day$.
- Locution $\mathbf{L_5}$: *answer-serving-proposal*(*degree-of-acceptance*)
  This locution is an answer to "serving proposal" and takes form of FIPA *reject-proposal* or *accept-proposal* ACL message. The *degree-of-acceptance* (number from 0 to 1) tells how much the specified device is suitable to serving the new agent. 0 indicates that device cannot serve this agent.
- Locution $\mathbf{L_6}$: *serve*(*proposal-id, device-id, info-description*)
  cDMA requests the selected DMA to serve a specified bundle. Locution contains *proposal-id*, thus DMA can check if it previously accepted the proposal. *device-id* identify ALD in a way further used by driver, eg. IP address. *info-description* is used to download bundle using *get-driver-bundle*.

- Locution $\mathbf{L_7}$: *register*(*df-agent-description*)
  Registers an installed agent in the Yellow Pages Agent. This locution is defined in the FIPA Agent Management Specification [6].
- Locution $\mathbf{L_8}$: *inform-result*(*event, params*)
  After detecting some event an agent informs other agents (which previously subscribed such information) about its occurrence. *event* is the type of an event and *params* are additional parameters which vary between event types.
- Locution $\mathbf{L_9}$: *request-resource*(*amount, duration*)
  Agents sends this message to resource agent (for example WRA) in order to request usage of some *amount* of resource in estimated time represented as *duration.*
- Locution $\mathbf{L_{10}}$: *resource-response*(*allowed, amount, duration*)
  Resource agent responses to other agent if it can use resource (*allowed*). *amount* and *duration* are allowed resource usage.
- Locution $\mathbf{L_{11}}$: *invoke-device-service*(*method, params*)
  Invoke specified *method* of a service exposed by device. This message can be transported via variety of protocols including JADE's one, UPnP, KNX or X10.

4 Conclusion and Future Research

In this research, an architecture which tries to solve a few problems that may occur while developing Smart Home Systems has been developed. First problem is "The Accidentally Smart Home", see [5]. The vast majority of smart homes are not build in the ground-up approach. People will upgrade their existing houses by buying new smart devices (architecture upgrade). The proposed architecture allows new devices to integrate and cooperate with the existing system. Other issue is the upgrade of devices' agents (software upgrade). Software is continuously and rapidly developing, thus the need of some mechanism of upgrading. Next problem which proposed architecture tries to solve is the architecture self-optimization. Since the architecture will be continuously changing there may occur new ways of agents' distribution. The result is an open, expandable agent architecture implemented on the JADE and OSGi platforms.

However there are few more issues which should be considered in future work. Probably the most important one is security. While installing a new device it registers itself in the cDMA via wireless network. But what if user's neighbour also has got his own SHS. It is needed to register new device in device's owner's system. This may be solved by setting a WiFi network name. Another possibility is to register device manually, via some kind of short-distance network (e.g. Bluetooth). This aspect still needs more research.

Another problem is a system scalability. In case of SHS the proposed architecture should work with good performance. But if we extend the system to any Smart Environment (for example Smart Company or even Smart City) there may occur some scalability problems. They can be solved by grouping agents into subsystems (e.g. by function, by spatial location, by used resources). After choosing the grouping strategy there may be created a hierarchic architecture, based on the proposed one. MC acts as subtree's root in a global tree-like architecture and allows inter-group communication. Nevertheless this problem also needs more research.

In the future we intend to implement SHS simulator using the proposed architecture and analyse its performance. The further step is testing usability of presented approach to see if the design goals are met in practice.

References

1. About the OSGi Service Platform. Technical Whitepaper Revision 4.1. OSGi Aliance (2007)
2. Aiello, M., Dustdar, S.: Are our homes ready for services? A domotic infrastructure based on the Web service stack. Pervasive and Mobile Computing 4, 506–525 (2008)
3. Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. John Wiley & Sons, Chichester (2007)
4. Cook, D.J., Youngblood, M.: Smart Homes. In: Berkshire Encyclopedia of Human-Computer Interaction, Berkshire Publishing Group (2004)
5. Edwards, W.K., Grinter, R.E.: At Home with Ubiquitous Computing: Seven Challenges. In: Abowd, G.D., Brumitt, B., Shafer, S. (eds.) UbiComp 2001. LNCS, vol. 2201, pp. 256–272. Springer, Heidelberg (2001)
6. Foundation for Intelligent Physical Agents. FIPA Specifications, `http://www.fipa.org/repository/standardspecs.html`
7. Gu, T., Pung, H.K., Zhang, D.Q.: Toward an OSGi-Based Infrastructure for Context-Aware Applications. IEEE Pervasive Computing 3(4), 66–74 (2004)
8. Kim, N.H., Jeong, Y.S., Ryu, S.H., Shin, D.R.: Mobile Healthcare System based on Collaboration between JADE and OSGi for Scalability. In: Proceedings of the International Conference on Multimedia and Ubiquitous Engineering, pp. 126–129. IEEE Computer Society, Washington (2007)
9. López de Vergara, J.E., Villagrá, V.A., Fadón, C., Gonzáez, J.M., Lozano, J.A., Campana, M.A.: An autonomic approach to offer services in OSGi-based home gateways. Computer Communications 31(13), 3049–3058 (2008)
10. Roberts, L.G., Wessler, B.D.: Computer network development to achieve resource sharing. In: Proceedings of the AFIPS Joint Computer Conference, pp. 543–549. ACM, New York (1970)
11. Velasco, J.R., Maestre, I.M., Navarro, A., López, M.A., Vicente, A.J., de la Hoz, E., Paricio, A., Machuca, M.: Location-aware Services and Interfaces in Smart Homes using Multiagent Systems. In: Proceedings of the 2005 International Conference on Pervasive Systems and Computing, pp. 104–110. CSREA Press (2005)
12. Wu, C.L., Liao, C.H., Fu, L.C.: Service-Oriented Smart-Home Architecture Based on OSGi and Mobile-Agent Technology. IEEE Transactions on Systems, Man, and Cybernetics–Part C: Applications and Reviews 37(2), 193–205 (2007)