A Scalable Distributed Architecture for Web-Based Software Agents

Dejan Mitrović[1(✉)], Mirjana Ivanović[1], Milan Vidaković[2], and Zoran Budimac[1]

[1] Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Novi Sad, Serbia
{dejan,mira,zjb}@dmi.uns.ac.rs

[2] Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia
minja@uns.ac.rs

Abstract. In recent years, the web has become an important software platform, with more and more applications becoming purely web-based. The agent technology needs to embrace these trends in order to remain relevant in the new era. In this paper, we present recent developments of our web-based multiagent middleware named Siebog. Siebog employs enterprise technologies on the server side in order to provide automatic agent load-balancing and fault-tolerance. On the client, it relies on HTML5 and related standards in order to run on a wide variety of hardware and software platforms. Now, with automatic clustering and state persistence, Siebog can support thousands of external devices hosting tens of thousands of client-side agents.

1 Introduction

During the last decade, there has been an obvious paradigm shift in software development. The *web* has evolved into an environment capable of providing functionalities not so long ago available only in desktop applications. One of the main reasons for the increasing popularity of web-only applications is their cross-platform nature, which allows the end-users to access their favorite applications in a wide variety of ways. As the overall result, the desktop-only technologies are becoming less and less relevant.

Computer clusters play an important role in modern web and enterprise applications. They provide *high-availability* of deployed applications [18]. This feature is concerned with continuous, uninterrupted delivery of services, regardless of hardware and software failures, or numbers of incoming requests. The high-availability is achieved through the so-called *horizontal scaling*, which is the processing of adding more nodes to the cluster as the demands for processing power increase.

Siebog is our multiagent middleware designed to provides support for intelligent software agents in this new setting [24]. It efficiently combines the *HTML5* and related web standards on the client side [20,22], and the *Enterprise edition*

M. Núñez et al. (Eds.): ICCCI 2015, Part I, LNAI 9329, pp. 67–76, 2015.
DOI: 10.1007/978-3-319-24069-5_7

of Java (Java EE) on the server side [21,23,29], in an effort to bridge the gap between the agent technology and industry applications.

By utilizing the standards and technologies readily-available in Java EE, Siebog offers “native” support for computer clusters on the server. The purpose of this paper is to discuss how this support is extended to the client side of Siebog as well. The goal is to provide the support for clusters that consist of arbitrary client devices, such as personal computers, smartphones, and tablets.

The motivation for this approach is straightforward. There can be an order of magnitude more external client devices than there can be server-side computers. Siebog could be used to distribute agents among connected clients and to support applications that require launching large populations of agents. Since the state-of-the-art smartphones have more processing powers that many laptop or desktop machines, they represent a significant computational resource.

This approach, however, does pose some technical challenges. Client-side clusters are highly dynamic, in the sense that clients are able to join and leave at any time. In order to deal with this situation, we added a highly-scalable infrastructure for agent state persistence which allows the agents to “rise above” the interruptions, and to operate regardless of their physical locations.

The rest of the paper is organized as follows. Section 2 discusses the overall motivation behind this paper, and presents relevant existing work. Section 3 shows how the support for dynamic and heterogeneous client-side clusters was added to Siebog. The overall performance evaluation is presented in Section 4. The final conclusions and future research directions are given in Section 5.

2 Background

Web application play an increasingly important role in the modern-day computing. They offer a number of advantages over traditional desktop application, such as the lack of need for installations, configurations, or upgrades. The importance of web applications is emphasized by the continuously increasing sales of mobile devices and the ability of web applications to run as native applications on these devices [30].

In order to maintain its relevance in this new era, the agent technology not only needs to move to the web, but it needs to do so in accordance to the modern standards and the end-users’ expectations. Agent-based applications need to seamlessly be integrated into web and enterprise applications in order to reach the end-users more easily, and to stay relevant in this new state of affairs.

Currently, there exists a large number of both open-source and commercial agent middlewares [3,4]. However, almost none of these systems has fully exploited the advantages of web environments. Some efforts aimed at extending existing systems with web support have been made, but usually in a inefficient manner. For example, in many Java-based middlewares, such as *JADE* [2] or *JaCa-Web* [19], the extensions are based on Java *applets*. But, Java applets require a browser plug-in to run, which is unavailable on some platforms (e.g. *iOS* and *Smart TVs*). With some desktop-based browsers also starting to disable

Java support[1], the applicability of Java-based web solutions becomes limited to a narrower set of hardware and software platforms.

One of the prominent ways of migrating the agent technology to the web is to use the expanding *HTML5* and related set of standards [12]. HTML5 covers various aspects of web and enterprise applications, from audio and video playbacks, to offline application support, to more advanced features, such as multi-threaded execution and *push*-based communication. In addition, since web browser vendors keep investing significant resources into improving the overall performance of their respective JavaScript virtual machines, the HTML5 is expected to become "a mainstream enterprise application development environment" [11].

2.1 Related Work

Over the years, many practical applications of agents on the web have been proposed. One domain that appears to be the most interesting to agent researchers and practitioners is *content personalization* and various *recommender systems* [17,26]. Another possible thriving area is to use the so-called *pedagogical agents* in web-based e-learning systems [6,15], as well as in knowledge management [9].

Along with the more recent trends, there have been several proposals of using mobile agents within the so-called *Internet of Things* (IoT) concept [27], in *smart objects* [1,10] and in *smart cities* [28].

The role of Siebog in these practical applications would be in providing a standards-compliant, platform-independent, and efficient [22] multiagent middleware. In addition, with the work presented in this paper, we intend to bring the more traditional agent applications to the web. As discussed later, Siebog is suitable for distributed systems with large populations of agents. A concrete example of its possible practical application would be in the area of swarm intelligence, such as [14].

As we discussed previously in [20,22], many traditional (i.e. desktop- or server-based) multiagent middlewares have exposed their functionalities to the web through Java applets. This approach does provide many important benefits, such as the immediate availability of complex reasoning agents in web browsers [19]. However, with the lack of Java support in many popular modern platforms, this approach is no longer sufficient.

To the best of our knowledge, currently there exists one additional HTML5-based multiagent middleware [16,27]. The middleware is focused on using (primarily) mobile agents to support the IoT requirements. On the technical viewpoint, the client side of their system does not utilize the full range of HTML5 and related standards (such as *Web Workers* [22]). It is also not clear how multiple agents could be started within the same host, and how would they interact with each other without the server. The backend is conveniently based on *Node.js*[2],

[1] https://java.com/en/download/faq/chrome.xml, retrieved on March 12, 2015.
[2] https://nodejs.org/, retrieved on March 12, 2015.

which simplifies certain development aspects (such as mobility), but lacks several advanced features found in Siebog, namely automated agent load-balancing and fault-tolerance.

3 Clustering Client-Side Siebog Agents

In this section we describe how Siebog is extended to support automatic clustering and load-balancing of its client-side agents. More concretely, we discuss how a possibly large set of heterogeneous client-side devices can be observed as a coherent cluster. The cluster can then be used to execute resource-demanding and computationally-expensive tasks, such as launching large populations of agents.

3.1 The Existing Architecture

The overall existing architecture of Siebog is shown graphically in Fig. 1. On the server side, Siebog includes the following three main modules [21,23,24,29]: *Agent Manager*, which acts as an agent directory and controls the agents' life-cycles, *Message Manger*, in charge of inter-agent communication, and the *WebClient Manager*, which acts as an intermediary for server-to-client (i.e. push) messaging, and also handles state persistence for client-side agents.

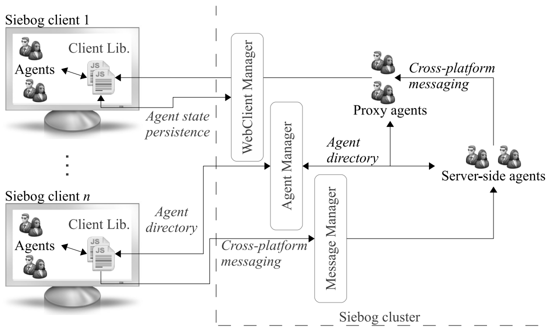

Fig. 1. The overall architecture of Siebog (adapted from [24]).

Client-side agents are executed inside web browsers [20,22,24] or possibly in dedicated JavaScript runtimes of external devices. Inside a device, agents rely on the Siebog client library for execution support and for communicating with the server. For example, the client library offers *proxy* implementations of server-side components. In order to send a message to the server-side agent, the client-side agent simply invokes the proxy implementation of the Message Manager. Underneath, the proxy then turns this invocation into a corresponding *AJAX* call to the server.

Another important feature of Siebog is that its server can (if configured) hold a proxy representation of each client-side agent. This representation simply forwards all incoming messages to the corresponding client-side counterpart (and through the WebClient Manager). This feature opens-up the possibility for transparent agent communication across different devices. An example is shown in Fig. 2. Let there be two agents, *AgA* and *AgB*, hosted by two different devices, *Device A* and *Device B*, respectively. When the AgA decides to send a message to AgB, the message will be delivered in the following way:

– AgA makes the appropriate call to the Message Manager Proxy.
– This call is transformed into an AJAX call to the server-side Message Manager.
– The Message Manager delivers the message to the AgB proxy.
– Since this is a proxy representation, it forwards the message to the WebClient Manager.
– The WebClient Manager, which is aware of all external clients, finally pushes the message to the target agent.

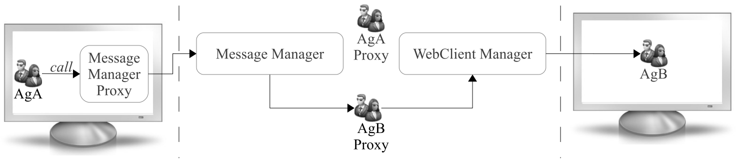

Fig. 2. Transparent communication of client-side agents across different devices.

Due to limitations imposed by certain web browsers [22], each client device can run up to a few dozens of agents. But, there can be large numbers of physical devices active simultaneously. The main idea here is to exploit this possibility in order to distribute portions of a large population of agents. This is conceptually similar to, for example, the famous *SETI@home* scientific experiment[3]. As an important advantage, the Siebog does not require any software installation: all the end-user needs to do is to visit the corresponding web page.

The main issue here is how to efficiently support these large numbers of external devices, and to deal with their inherently dynamic availability.

3.2 Managing Heterogeneous and Dynamic Clusters

A central component in a computer cluster is a *load-balancer* with the task of distributing the work across available machines. In the context of Siebog, the load-balancer continuously accepts tasks that need to be solved. For example, it

[3] http://setiathome.berkeley.edu/, retrieved on March 12, 2015.

can accept large maps for the *Traveling Salesman Problem*[4] and then partition each map [13] and send it (along with the corresponding set of ants) to a subset of available devices.

In the majority of existing non agent-based distributed architectures, the load-balancer selects the target device randomly. In this way, the workload is distributed "for free" and, in the longer run, equally among all available devices. However, the clusters that consist of Siebog clients are heterogeneous, in the sense that they can include devices with very different processing capabilities. Therefore, the load-balancing process is a bit more complex.

When it comes to load-balancing in heterogeneous systems, the agent-oriented research has proposed some rather complex approaches (e.g. [5,25,31]). In case of Siebog, however, we decided to follow the industry norms of keeping things as simple as possible. Once a device joins the cluster for the first time, a performance benchmark is executed. The results of this benchmark are used to assign a number of *compute units* (CUs) to the device. Now, during the load-balancing phase, the target device is selected with the probability that corresponds to its number of CUs.

From the end-user's point of view, joining the Siebog cluster is fairly simple: he/she only needs to visit the appropriate web page that hosts the worker agents. Unfortunately, it is also very easy to leave the cluster; once the end-user closes the web browser or switches off the device, the hosted agents are lost. For meaningful practical applications, however, the agents need to be able to run regardless of these interruptions.

In order to support possibly large numbers of agents, Siebog needs a scalable datastore, one capable of serving multitudes of requests per second. The datastore should also be fault-tolerant – capable of surviving server crashes. More formally, these requirements can be described as principles of the so-called *Dynamo systems* [8]. Currently, there exist several concrete Dynamo realizations. After a careful evaluation of these solutions, we determined that the open-source *Apache Cassandra*[5] datastore fulfills the needs of the Siebog client-side clusters. The client-side Siebog library has been extended to allow the agents the interact with the datastore directly, and over the *WebSocket* protocol [20,22]. The performance evaluation of the new architecture is discussed in the following section.

4 Performance Evaluation

The newly proposed architecture of Siebog needs to be able to serve large numbers of running agents, which are concurrently, and at high frequencies, storing and retrieving their respective internal states. In order to evaluate this feature, we used the open-source *Yahoo! Cloud Serving Benchmark* (YCSB) [7] tool. YCSB is designed for load-testing of (primarily) NoSQL databases, and can be

[4] http://www.math.uwaterloo.ca/tsp/data/, retrieved on March 12, 2015.
[5] http://cassandra.apache.org/, retrieved on March 12, 2015.

configured through a range of parameters, including the desired number of operations per second (throughput), the number of concurrent threads, maximum execution time, etc.

The experiments were performed using two machines, each with 8 virtual CPUs and 28 GB of RAM, running 64-bit version of *Ubuntu 14.04 LTS*. One machine was hosting the Apache Cassandra datastore and the Siebog server, while the other one was used to launch YCSB-simulated external devices. Each external device was represented by a separate WebSocket that the server needed to maintain.

In a realistic use-case, there will be many more writes to the store than reads. That is, the internal state of an agent will usually be read only once: when the web page is loaded and the agent is started. On the other hand, the state can be stored multiple times during the agent's execution, e.g. after each processed message or after each computational sub-step. Therefore, the YCSB workload was set up as *write-heavy*, so that 90% of all operations are *writes*.

The goal of the experiment was to determine the maximum number of external devices as well as client-side agents that our system can support. For this goal, a number of test-cases was executed. With each successive test-case, the total number of connected devices was increased. Then we would try to find the maximum throughput (i.e. the number of operations per second) that the Siebog server can support. A test-case was executed for one hour, and the maximum throughput value that was stable during this period was taken as the end-result.

We started with 100 external devices, increased the number for each test-cased, and finally reached the limit of approximately 16,000 devices. This number actually represents the maximum number of open connections that the operating system could support. Nonetheless, being able to support 16,000 external devices using a single-node Siebog cluster is an excellent result, given the fact that the cluster can easily be extended with more nodes as the demands grow. The results of this test-case are shown in Fig. 3. More concretely, the figure shows average read and write latencies during the one hour period, calculated at one minute intervals. The latencies are very low (expressed in microseconds), due to the WebSocket protocol's support for asynchronous I/O.

For each test-case, through trial-and-error, we determined that the value of approximately 6,000 operations per second is the maximum throughput that remained stable during the one hour period. Our systems is capable of serving much larger numbers than this (i.e. up to 100,000 operations per second), but only in "short bursts," after which the backend datastore needed some time to manage all the write operations.

Although the 6,000 operations per second might not seem as a large number at first, it is worth noting that an agent is not supposed to store its internal state at every second. So even if agents store their respective states at every 10 seconds, we reach the conclusion that our Siebog multiagent middleware can manage 60,000 agents distributed across 16,000 devices, using only one server-side node.

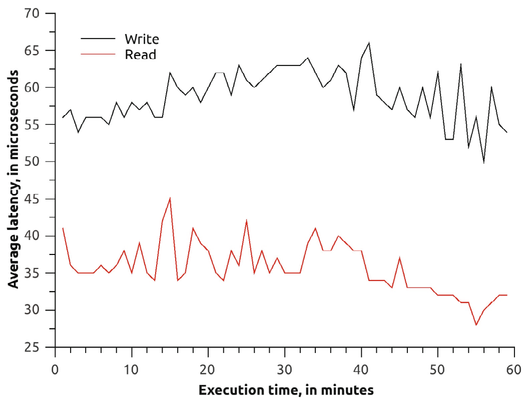

Fig. 3. Average read and write latencies of the test-case simulating 16,000 external devices and 6,000 operations per second, during the one hour period. The latencies are calculated at one minute intervals.

5 Conclusions and Future Work

Siebog is a web-based, enterprise-scale multiagent middleware. It combines the enterprise technologies on the server with HTML5 and related standards on the client in order to support multiagent solutions whose functionalities meet the expectations of modern software systems.

In this paper, we have presented how Siebog was recently updated to support dynamic clusters of heterogeneous client-side devices. The two new components of our system are the load-balancer, which is in charge of distributing agents across the connected devices, and a highly-scalable backend datastore used for persisting the internal states of client-side agents.

For any meaningful application of Siebog, its client-side agents need to become "detached" from their host environments (e.g. web pages). As shown in the paper, thanks to the use of a Dynamo architecture and the WebSocket protocol, on just one server node the state persistence system in Siebog can support thousands of external devices hosting tens of thousands of client-side agents, which is an excellent result.

Future developments of Siebog will be focused on an even tighter integration of client-side and server-side agents. Also, the system will be extended with an interoperability module, allowing it to interact with third-party multiagent solutions. Although Siebog already supports BDI agents on the server, the work is underway to develop a unique architecture for intelligent agents.

Acknowledgments. This work was partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, Grant III-44010, Title: Intelligent Systems for Software Product Development and Business Support based on Models.

References

1. Aiello, F., Fortino, G., Gravina, R., Guerrieri, A.: A java-based agent platform for programming wireless sensor networks. Computer Journal **54**(3) (2011)
2. Bellifemine, F., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. John Wiley & Sons (2007)
3. Bordini, R.H., Braubach, L., Dastani, M., El, A., Seghrouchni, F., Gomez-sanz, J.J., Leite, J., Pokahr, A., Ricci, A.: A survey of programming languages and platforms for multi-agent systems. Informatica **30**, 33–44 (2006)
4. Bădică, C., Budimac, Z., Burkhard, H.D., Ivanović, M.: Software agents: languages, tools, platforms. Computer Science and Information Systems, ComSIS **8**(2), 255–298 (2011)
5. Cao, J., Spooner, D.P., Jarvis, S.A., Nudd, G.R.: Grid load balancing using intelligent agents. Future Generation Computer Systems **21**(1), 135–149 (2005)
6. Cheng, Y.M., Chen, L.S., Huang, H.C., Weng, S.F., Chen, Y.G., Lin, C.H.: Building a general purpose pedagogical agent in a web-based multimedia clinical simulation system for medical education. IEEE Transactions on Learning Technologies **2**(3), 216–225 (2009)
7. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, pp. 143–154. ACM, New York (2010)
8. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: Amazon's highly available key-value store. In: Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles, SOSP 2007, pp. 205–220 (2007)
9. Dignum, V.: An overview of agents in knowledge management. In: Umeda, M., Wolf, A., Bartenstein, O., Geske, U., Seipel, D., Takata, O. (eds.) INAP 2005. LNCS (LNAI), vol. 4369, pp. 175–189. Springer, Heidelberg (2006)
10. Fortino, G., Guerrieri, A., Russo, W.: Agent-oriented smart objects development. In: 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD), pp. 907–912, May 2012
11. Gartner identifies the top 10 strategic technology trends for 2014, October 2013. http://www.gartner.com/newsroom/id/2603623 (retrieved on March 12, 2015)
12. HTML5: a vocabulary and associated APIs for HTML and XHTML, October 2014. http://www.w3.org/TR/html5/ (retrieved on March 12, 2015)
13. Ilie, S., Bădică, A., Bădică, C.: Distributed agent-based ant colony optimization for solving traveling salesman problem on a partitioned map. In: Proceedings of the International Conference on Web Intelligence, Mining and Semantics, WIMS 2011, pp. 23:1–23:9. ACM (2011)
14. Ilie, S., Bădică, C.: Multi-agent approach to distributed ant colony optimization. Science of Computer Programming **78**(6), 762–774 (2013)
15. Ivanović, M., Mitrović, D., Budimac, Z., Jerinić, L., Bădică, C.: HAPA: Harvester and pedagogical agents in e-learning environments. International Journal of Computers Communications and Control **10**(2), 200–210 (2015)
16. Jarvenpaa, L., Lintinen, M., Mattila, A.L., Mikkonen, T., Systa, K., Voutilainen, J.P.: Mobile agents for the internet of things. In: 17th International Conference on System Theory, Control and Computing (ICSTCC), pp. 763–767, October 2013
17. Lops, P., Gemmis, M., Semeraro, G.: Content-based recommender systems: state of the art and trends. In: Recommender Systems Handbook, pp. 73–105 (2011)

18. Michael, M., Moreira, J.E., Shiloach, D., Wisniewski, R.W.: Scale-up x scale-out: a case study using Nutch/Lucene. In: IEEE International Parallel and Distributed Processing Symposium, pp. 1–8, March 2007
19. Minotti, M., Santi, A., Ricci, A.: Developing web client applications with JaCa-Web. In: Omicini, A., Viroli, M. (eds.) Proceedings of the 11th WOA 2010 Workshop, Dagli Oggetti Agli Agenti, Rimini, Italy, September 5–7, 2010. CEUR Workshop Proceedings, vol. 621. CEUR-WS.org (2010)
20. Mitrović, D., Ivanović, M., Bădică, C.: Delivering the multiagent technology to end-users through the web. In: Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics, WIMS 2014, pp. 54:1–54:6. ACM (2014)
21. Mitrović, D., Ivanović, M., Budimac, Z., Vidaković, M.: Supporting heterogeneous agent mobility with ALAS. Computer Science and Information Systems **9**(3), 1203–1229 (2012)
22. Mitrović, D., Ivanović, M., Budimac, Z., Vidaković, M.: Radigost: Interoperable web-based multi-agent platform. Journal of Systems and Software **90**, 167–178 (2014)
23. Mitrović, D., Ivanović, M., Vidaković, M., Budimac, Z.: Extensible Java EE-based agent framework in clustered environments. In: Müller, J.P., Weyrich, M., Bazzan, A.L.C. (eds.) MATES 2014. LNCS, vol. 8732, pp. 202–215. Springer, Heidelberg (2014)
24. Mitrović, D., Ivanović, M., Vidaković, M., Budimac, Z., Bădică, C.: An enterprise-scale multiagent middleware based on HTML5 and Java EE technologies. Advances in Electrical and Computer Engineering (in print)
25. Nehra, N., Patel, R.: Towards dynamic load balancing in heterogeneous cluster using mobile agent. In: International Conference on Conference on Computational Intelligence and Multimedia Applications, vol. 1, pp. 15–21, December 2007
26. Swezey, R.M.E., Shiramatsu, S., Ozono, T., Shintani, T.: Intelligent page recommender agents: real-time content delivery for articles and pages related to similar topics. In: Mehrotra, K.G., Mohan, C.K., Oh, J.C., Varshney, P.K., Ali, M. (eds.) IEA/AIE 2011, Part II. LNCS, vol. 6704, pp. 173–182. Springer, Heidelberg (2011)
27. Systä, K., Mikkonen, T., Järvenpää, L.: HTML5 agents: mobile agents for the web. In: Krempels, K.-H., Stocker, A. (eds.) WEBIST 2013. LNBIP, vol. 189, pp. 53–67. Springer, Heidelberg (2014)
28. Verma, P., Gupta, M., Bhattacharya, T., Das, P.K.: Improving services using mobile agents-based iot in a smart city. In: International Conference on Contemporary Computing and Informatics (IC3I), pp. 107–111 (2014)
29. Vidaković, M., Ivanović, M., Mitrović, D., Budimac, Z.: Extensible Java EE-based agent framework – past, present, future. In: Ganzha, M., Jain, L.C. (eds.) Multiagent Systems and Applications. Intelligent Systems Reference Library, vol. 45, pp. 55–88. Springer, Heidelberg (2013)
30. Xanthopoulos, S., Xinogalos, S.: A comparative analysis of cross-platform development approaches for mobile applications. In: Proceedings of the 6th Balkan Conference in Informatics, BCI 2013, pp. 213–220. ACM, New York (2013)
31. Zhang, Z., Zhang, X.: A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation. In: 2nd International Conference on Industrial Mechatronics and Automation (ICIMA), vol. 2, pp. 240–243, May 2010