

## Robot as a Service in Cloud Computing<sup>1</sup>

Yinong Chen, Zhihui Du\*, and Marcos García-Acosta\*\*

Computer Science and Engineering, Arizona State University, Tempe, AZ 85287-8809, USA

\*Department of Computer Science and Technology, Tsinghua University, China

\*\*Intel Corporation, Embedded Computing group, Chandler, Arizona, USA

**Abstract** — Service-oriented architecture and cloud computing are becoming a dominant computing paradigm, as all major computing companies are supporting this paradigm and more and more organizations are adopting this paradigm. Robotics and service-oriented robotics computing start to joint this new paradigm in the past five years and are now ready to participate in large scale. This paper reports our research on service-oriented robotics computing and our design, implementation, and evaluation of Robot as a Service (RaaS) unit. To fully qualify the RaaS as a cloud computing unit, we have kept our design to comply with the common service standards, development platforms, and execution infrastructure. We also keep the source code open and allow the community to configure the RaaS following the Web 2.0 principles of participation. Developers can add, remove, and modify the RaaS of their own. For this purpose, we have implemented our RaaS on Windows and Linux operating systems running on Atom and Core 2 Duo architectures. RaaS supports programming languages commonly used for service-oriented computing such as Java and C#. Special efforts have been made to support Microsoft Visual Programming Language (VPL) for graphic composition. We are working with high schools to use RaaS and VPL in robotics camps and robotics competitions.

**Keywords** – RaaS, SOA, SOC, Service-oriented robotics computing, and robot on Intel architecture

### I. INTRODUCTION

Service-Oriented Architecture (SOA) considers a software system consisting of a collection of loosely coupled services that communicate with each other through standard interfaces and via standard message-exchanging protocols. Cloud computing extends the scope of SOA to include the development platform and the execution infrastructure, and thus cloud computing is typically characterized by the features such as Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS), and Hardware as a Service (HaaS) [1]. However, any other virtualized services, say, X as a Service, can be submitted to the cloud infrastructure as long as the standards are followed [2]. Majitek applied the concept of Device as a Service to implement the machine-to-machine communication in SOA standards and in a Web-based IP network [3]. Cloud computing makes it possible to

completely move away from the desktop-based computing to the full Web-based development and application: Developers use a platform on Web through a Web browser, develop software and store data on the Web, configure the hardware/infrastructure (processing power, memory capacity, and communication bandwidth), and execute application on the Web.

SOA and cloud computing are pervasive. They are the foundation of today's Web-based applications [4][5]. All major computing companies support software development in SOA and cloud computing, including Amazon Web Services (EC2 Cloud), Google (Google Code and App Engine), IBM (Eclipse and IBM Cloud), and Microsoft (Visual Studio and Azure).

However, SOA and cloud computing have been limited to e-commerce and enterprise computing systems so far. Applying them in physical devices and embedded systems is one of the most recent challenges and research approaches. Sponsored by Microsoft Research's embedded system program in 2003, Chen applied the SOA concepts to recomposable embedded systems and robotics applications. A prototype of SOA robot was implemented in 2005 [6]. Microsoft Robotics Developer Studio (MRDS) was a vital step in applying SOA to embedded systems [7][8][9]. MSRS defined a mapping layer, called manifest, which separates the virtual services from the physical devices and makes the robotics application decoupled from the specifics of the devices. REST protocol, instead of SOAP, is used in MRDS to improve the real-time communication capacity. Trifa, Cianci and Guinard applied SOA for dynamically controlling swarm robots [10]. Tröger and Rasche applied SOA to 'Virtual Remote Laboratory' concept and used LEGO NXT robots for physical experiment installations [11]. K. C. Thramboulidis, G. Doukas, and G.Koumoutsos applied SOA for embedded systems development. They proposed the idea of modeling each feature of the development process as a web service and presented a prototype implementation showing the usefulness of the research [12].

SOA has been looked upon as a suitable and effective approach for industrial automation and manufacturing which ultimately uses SOA to control and manage robotics cells that are responsible for various functions in the automation process [18][19]. RoboLink Protocol that is

<sup>1</sup> The project is partly sponsored by U.S. Department of Education FIPSE program and by Intel Robotics Challenge initiative, 2007-2010

developed for reliable communication among robots is based on the use of Web Services [20]. These efforts made towards applying SOA in the field of robotics and automation clearly show the growing importance of SOA in robotics. Our project is a joint project with Intel. After the robotic kits are disseminated to high schools for volume testing, the robotic kits can enter the industrial robotics market [31].

Using SOA for robotics has certain inherent advantages over the traditional methods of building a robot. The main benefit is to have a layer of common services with standard interfaces. Each service represents a component such as sensor, actuator, or effector. The invocations of the services are device-independent. In this way, we can deploy the same application to different robotics platforms, as long as we rebind the service invocation to the services that are built for the robotics platform. The languages used in writing the application and the services can be different, which allows us to write robotics applications using high-level languages such as Java and C# in multithreaded environment and event-driven programming paradigm. Depending on the drivers available, we can use different languages, such as C, C++ and assembly languages to write the device control programs before wrapping them into services. The availability of a layer of services allows us to develop different robotics applications in a short period of time. Furthermore, we have developed the services on standard PC platform, which is pervasive and cost effective.

This paper is based on our previous work on service-oriented robotics computing research. In [6], Chen laid out the main idea of service-oriented robotics computing and the initial implementation of the architecture and components using Parallax Boe-Bot and Windows CE-based handheld devices. The work was sponsored by Microsoft Research's Embedded System program. In [13] and [15], a new version of the service-oriented robotics computing architecture and the design are reported. The new implementation is conducted using a self-built robot based on Intel processor and off-the-shelf components. The focus of [13] was on the robot organization, interface design between the sensors/actuators and the processor board and the support to the service-oriented computing. The focus of [14] was the event-driven architecture, the floor-detecting algorithms, office patrolling algorithms, and the simulation results of efficiency of the algorithms. In [15], we reported the performance evaluation in terms of the execution time needed for a given set of tasks and the power consumption based on the implementation of Intel Core 2 Duo processor. Reference [16] gives the Website where the development history, documents, code, photos, videos, and the competitions among the robots are kept. Our goal is to create an easy-to-build and easy-to-program robotic kit for disseminating to high schools and for starting a competition series of autonomous robots. Current robotics competitions are mainly remote controlled. Students spend almost all their time in building a mechanic device, with little need of programming, as programming an autonomous robot is beyond the capability of high school students. The

development team at Arizona State University has extensive experience in service-oriented computing and distributed software development. We have developed a layer of services and implemented Microsoft VPL Visual Programming Language on standard Intel platform. We have taught VPL autonomous robotics programming to high schools students [17]. We know that with our approach, high school students can program autonomous robots, and we have achieved our goal of creating an easy-to-program robot. However, we do not have the expertise to develop professional robotic hardware. Currently, we are working with the Carl Hayden High School in Phoenix Arizona to create an Intel processor-based robotic kit. Carl Hayden has won numerous awards for their participation in many national and international robotics competitions and has extensive expertise in the assembly of robots. The combination of our software and Carl Hayden hardware will create the starting kit for the autonomous robotics competition series.

The key contribution of this paper is concept of **Robot as a Service** (RaaS), which enforces the design and implementation of a robot or a device to be an all-in-one SOA unit, that is, the unit includes services for performing functionality, service broker for discovery and publishing, and applications for client's direct access. In our previous design of SOA robots, the robot is an application that uses services from a remote backend computer. This all-in-one design gives the robot unit much more power and capacity, so that it can qualify as a fully self-contained cloud unit in the cloud computing environment. Another key contribution reported in the paper is the development of the services that translates the Microsoft Robotics Studio's VPL (Visual Programming Language) program into executables on Intel platform. These services allow the standard VPL programs to be developed on the Intel-based robot unit.

Performance of the RaaS unit on different Intel processors are evaluated, including the evaluation on the new embedded Atom processor.

The rest of the paper is organized as follows. Section 2 presents the requirements and design of RaaS in the cloud computing environment. Section 3 outlines the implementation of our RaaS unit and the execution model of the unit. Section 4 discusses composition and application building using RaaS units, and particularly, the implementation of the graphic composition language VPL on RaaS. Section 5 presents the experiment results and analyze the studies the capacity and performance of RaaS on Intel processors. Finally, section 6 concludes the paper.

## II. RAAS IN CLOUD COMPUTING

A robot is a mechanical or virtual artificial agent. It is usually a system, which, by its appearance or movements, conveys a sense that it has intent or agency of its own [21]. Let us consider the basic requirements of robot as a service in the cloud computing environment. There can be many kinds of robot cloud units or autonomous devices. For example, robot cops [22], restaurant robot waiters [23],

robot pets [24], and patient care robots [25]. These robots are distributed in different locations and can be accessed through internet. The basic requirement we consider here is that the RaaS should have complete functions of SOA, that is, as service provider, as a service broker, and as a service client:

1. A RaaS cloud unit is a service provider: Each unit hosts a repository of preloaded services. A developer or a client can deploy new services into or remove service from a robot. The services can be used by this robot can also be shared with other robots.
2. A RaaS cloud contains a set of applications deployed: A developer or client can compose a new application (functionality) based on the services available in the unit and outside the unit.
3. A RaaS unit is a service broker: a client can look up the services and applications available in the unit's directory. A client can search and discover the applications and services deployed on the robot by browsing the directory. The services and applications can be organized in a hierarchy of classes to facilitate the discovery.

Figure 1 shows the main components of a RaaS unit and typical applications and services deployed. Figure 2 shows how the RaaS units fit in the cloud computing environment. The SOC software in RaaS will communicate with the drivers and other operating system components, which further communicate with the devices and other hardware components. The RaaS units can directly communicate with each other through Wi-Fi, if the wireless infrastructure is available or through Bluetooth, if two units are close to each other. The communication between RaaS and other services in the cloud are through standard service interface WSDL. The communication protocol supporting the invocations is SOAP or REST protocol.

### III. IMPLEMENTATION OF RAAS

To prove the concepts, we have implemented a prototype of the RaaS shown in Figure 2. To make the RaaS more adaptable, we have made the following decisions in the implementation:

**Hardware:** Generic Intel processor and mother board are used. We have test the RaaS on Core 2 Due and Atom processors. The major component list includes:

- Intel Core 2 Duo Processor 1.6GHz; alternatively, Atom N270 processor 1.6GHz;
- Intel Embedded Mini-ITX Motherboard;
- USB to I<sup>2</sup>C Communications Module;
- Arduino Board
- M2-ATX Intelligent Power Supply
- MD23 Dual Motor Driver
- M2-ATX Intelligent Power Supply

Generic USB and common serial port devices, such as sonar sensor, compass sensor, motion sensor, and thermo sensor, webcam, digital servo, and motors are also used.

Figure 3 shows how the devices are accessed through services and drivers.

**Operating systems:** We have implemented an Windows XP version and a Linux version.

**Programming languages:** We have used C# and Java to program the services and the applications. We also implemented a service that interfaces Visual Programming Language (VPL) applications to Intel platform.

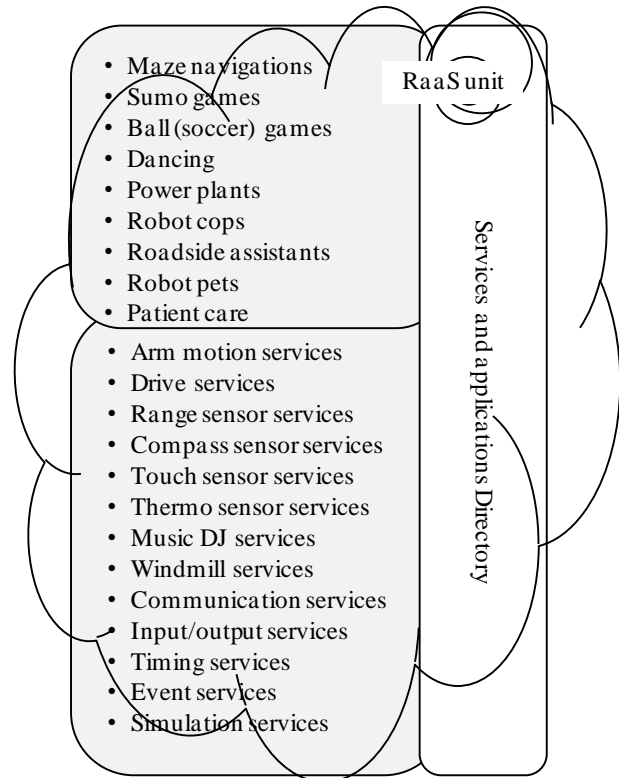


Figure 1. RaaS unit

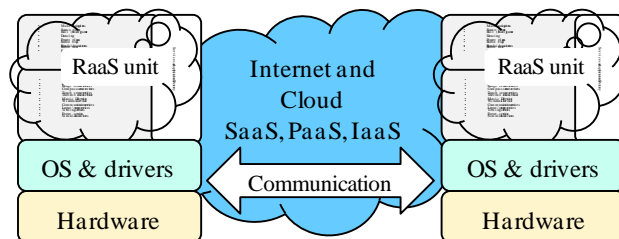


Figure 2. RaaS in cloud environment

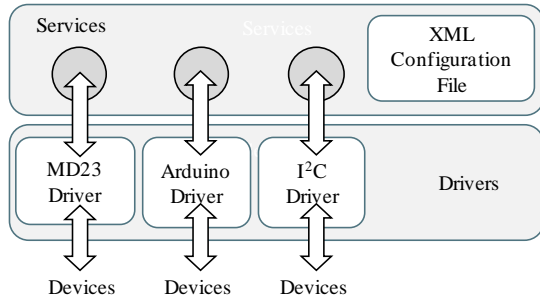


Figure 3. Interfacing devices to SOA

**Service hosting:** Using multithreading in C# and Java, we implemented a multitenant-based service hosting environment, where multiple versions of the service code are hosted by a worker process. Figure 4 shows the thread-based service hosting mechanism.

All requests are processed by a reactor as a single entry point, where the requests are buffered and checked against the directory. If an application or service is not registered, the request will be rejected immediately. Otherwise, the request will be redirected to the hosting worker process. A worker process will start a service as a thread and manage the service's lifetime. We have two worker processes, one for the applications and one for the services.

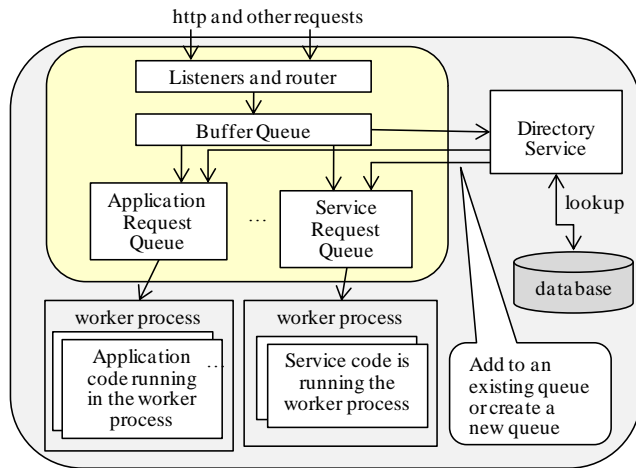


Figure 4. Hosting and execution model

#### IV. VPL COMPOSITION OF RAAS APPLICATIONS

Applying RaaS in education, particularly in high school computer science education, has been an important objective of RaaS design and implementation [26]. We have ported Microsoft Robotics Developer Studio (MRDS) and its graphic language, Visual Programming Language (VPL), into our design. We have been teaching high school students programming robots in VPL since 2007 [17].

Microsoft Robotics Studio and VPL have been implemented on specially design robotics platforms and their components, such as iRobot, LEGO NXT Mindstorm, Nex Robotics, and Parallax [27]. VPL is not running on

general purpose architecture like Intel and the generic USB or serial port devices (sensors and actuators). We implemented a layer of mapping services from the device drivers to Microsoft DSS (Decentralized Software Services) in MRDS, so that C# and VPL applications can invoke these device drivers, as shown in Figure 5.

In the typical applications of MRDS and VPL, applications and DSS services are running on a backend computer, while the devices and device drivers are running on board of the robot. The robot and its devices are merely input/output devices of the backend computer. In our design, as the RaaS is running on a powerful Intel processor, we could integrate all applications, services, and devices on the robot, or RaaS unit.

We have implemented a number of DSS services in the RaaS, including those that wrap various sensors and actuators. Figure 6 shows the VPL diagram (application) that uses three sonar sensor services on Intel platform. These sensors are placed in the front, left, and right of an Intel based robot.

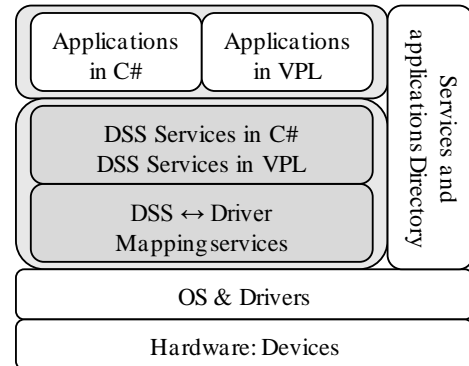


Figure 5. DSS-driver mapping services

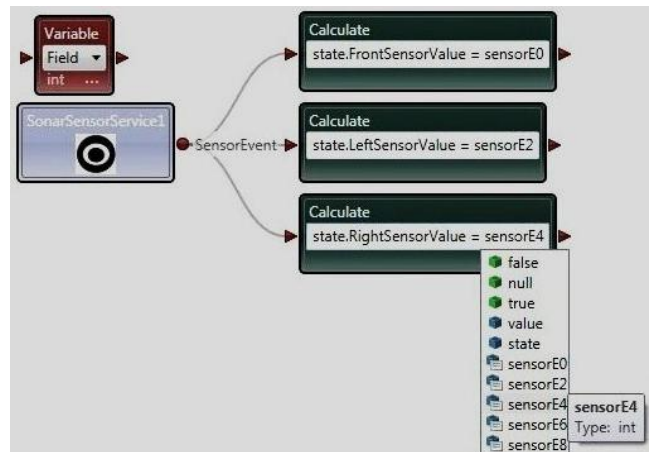


Figure 6. Binding a sonar sensor DSS in VPL

Figure 7 shows the motor service used in VPL diagram and the data connection settings before the proceeding activity and the motor service.

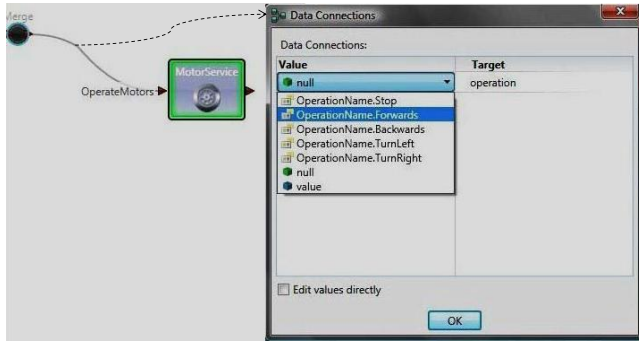


Figure 7. Binding a motor DSS in VPL

The C# code below shows the implementation of the MotorService shown in Figure 7. The code implements service contracts and service ports that map to the operations of the device driver.

```
using System; using System.Collections.Generic;
using System.ComponentModel; using Microsoft.Ccr.Core;
using Microsoft.Dss.Core.Attributes;
using Microsoft.Dss.ServiceModel.Dssp;
using Microsoft.Dss.ServiceModel.DsspServiceBase; using W3C.Soap;
namespace Robotics {
    public sealed class Contract {
        [DataMember]
        public const string Identifier =
            "http://schemas.tempuri.org/2009/11/motorservice2.html";
    }
    [DataContract]
    public class MotorService2State {
        private OperationName _operation = OperationName.Stop;
        [DataMember(IsRequired = true)]
        public OperationName operation {
            get { return _operation; }
            set { _operation = value; }
        }
    }
    [DataContract]
    public enum OperationName { Stop = 0, Forwards, Backwards,
        TurnLeft, TurnRight,
    }
    [ServicePort]
    public class MotorService2Operations : PortSet<DsspDefaultLookup,
        DsspDefaultDrop, Get, Replace> {
        public class Get :
            Get<GetRequestType, PortSet<MotorService2State, Fault>> {
            public Get() {}
            public Get(GetRequestType body)
                : base(body) {}
            public Get(GetRequestType body, PortSet<MotorService2State,
                Fault> responsePort)
                : base(body, responsePort) {}
        }
        [DisplayName("OperateMotors")]
        [Description("Indicates when the TrialService1 state changes.")]
        public class Replace : Replace<MotorService2State,
            PortSet<DefaultReplaceResponseType, Fault>> {
            public Replace() {}
        }
    }
}
```

VPL is based on the event-driven and data-driven approach, and the algorithm can be described by a finite state machine or state diagram. Figure 8 gives the maze navigation algorithm that guides the RaaS to traverse the maze. Tutorials on VPL programming and robotics application development can be found in Microsoft Robotics Development Studio. Detailed explanations on the principles as well as development guide are also given in [4].

In the algorithm that guides the robot traversing a maze, it is assumed that the robot has range sensor installed to measure the distance in the front. The robot starts to move forward. When an obstacle is detected, it turns left 90 degrees and measures the distances. Save the distance into a variable. It then spins 180 degrees and measures the distance. It compares this distance with the distance saved in the variable. If the current value is greater, the robot moves forward. If the current value is smaller, the robot spins 180 degrees, and then moves forward.

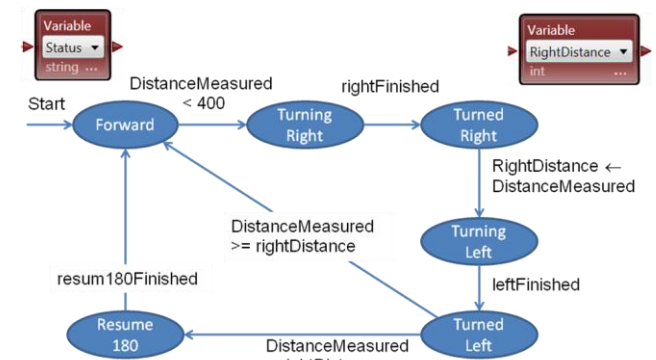


Figure 8. State Diagram of a maze navigation algorithm

In the state diagram, we use two VPL variables. The variable "Status" can take six possible string-type values of Forward, TurningRight, TurnedRight, TurningLeft, TurneLeft, and Resume180. The int-type variable "RightDistance" is used to store the distance to the obstacle after the robot turned right.

The photo of our RaaS prototype is given in Figure 9(a). The video that the RaaS unit navigates in a maze is available at the link given in [28].

We also implemented an application involving two RaaS units on different platforms: One on Intel platform and one on NXT Mindstorm, as shown in the photo in Figure 9(b). As can be seen in the photo that a laptop is used to control the larger robot. As it is Intel platform-based, we can use any Intel-based computer to replace the RaaS onboard computer.

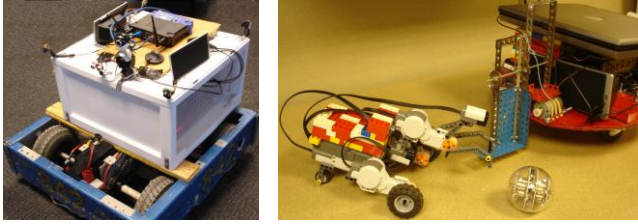


Figure 9. (a) A RaaS unit navigating in a maze (left) and (b) Two RaaS units playing a ball (right)

The video of the two RaaS units playing a ball and dancing around each other is available at the link [29]. All the technical reports and the source code are open and available at the site given at [16]. As a part of the smart home project, we are also developing a Wiki site that allows participants to contribute their documents and code into the site [30]. RaaS public service repository will be hosted in the Wiki site.

On the hardware side, we have teamed up with Carl Hayden Community High School Robotics Club and have a new robotic kit developed, as shown in Figure 10. Carl Hayden's Robotics Club has extensive expertise in the assembly of robots and accolades for its participation in many national and international competitions.

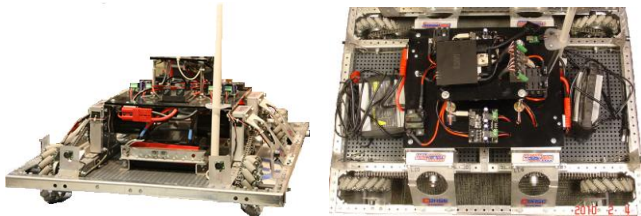


Figure 10. The Intel Robotics Starter Kit (two views)

Table 1 lists the key components used in the Starter Kit in Figure 10, which is similar to the component list used in the ASU versions, as listed in section 3.

TABLE 1. ROBOTICS STARTER KIT COMPONENT LIST

Component	Description
Intel Atom N270 processor	Process all data, including sensor data and control motors
Digital Compass	Determine direction
Fuses Fuse Panel	Protects electronics components
I <sup>2</sup> C to Converter board	Covert Sensor Data to Interface with Computer
Mecanum Drive System	Mounting Platform/Structure
Motor Speed Controller	Allows user to control the speed of the motors
Phidget Interface Kit	Interfaces Analog Input & Digital input/output with a

	computer via USB port
Sonar Sensor	Distance Feed Back

In a joint white paper between Intel, Arizona State University, and Carl Hayden Community High School, a Robotics Starter Kit is informally introduced and explained in detail [31]. A video presentation of the system is at [<http://vimeo.com/9740048>].

## V. PERFORMANCE EVALUATION

A large number of experiments have been performed and data on performance and power consumption are collected for different processors (Core 2 and Atom) and at different clock rates. This section will present a sample set of data and the interpretation of the data.

Experiment setting: Experiments were conducted on Intel Core 2 Duo and Intel Atom N270 processors by running a specific number of threads in parallel. Each thread represents an instance of a service. The experiments have been conducted on physical services that read sensors and control actuators. However, we have a limited number of sensors (three sonar, three touch sensors, one compass sensor, and one light sensor) and actuators (two drive motors and one servo for the rotating webcam). For both Core 2 Duo and Atom processors, these loads are to low: around 0.4% of the processor usage.

To measure the higher load scenarios, we have written simulated threads and used a sleep time to model the interval in which the service is not requested. In our experiment setting, we use 100ms, 500ms, and 1000ms sleep times, which represent the arrival rates of 4/second, 1/second, and 0.5/second, respectively. This setting assumes that the processing time of each service is short, as the services are reading with reading the sensor values and sending control data to the motors. The time consuming services, such as the video data processing for object recognition, are not included in this part of the simulation. With the simulated services, we can create thousands of threads to stress test the RaaS and its underlying software and hardware.

In this specific experiment, we have used a half of the threads to run in a continuous loop and to act as clients that make requests to the server. The other half of the threads run in a continuous loop and act as services to handle the requests. Processor usage values shown in the Windows' Task Manager were recorded for 2 minutes in the experiment period and the average CPU usage was calculated. The experiments were repeated for different values of sleep times per iteration and were conducted at different processor clock rates for Intel Core 2 Duo. The data collected are shown in Figure 11(a), (b), and (c), respectively.

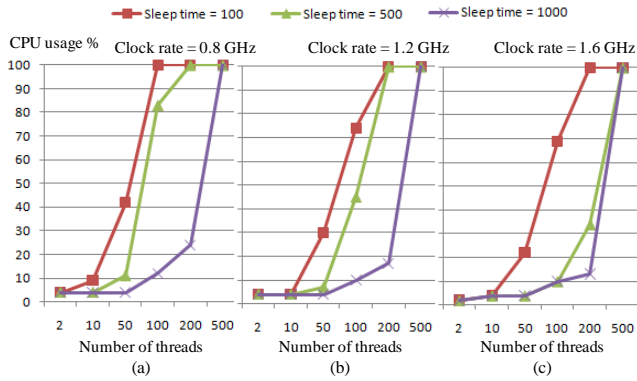


Figure 11. Data from Intel Core 2 Duo processor

For Intel Atom N270, the experiment was conducted for 1.6 GHz only, as the Atom was running in Windows XP, which does not support the adjustment of clock rate. The data collected are shown in Figure 12.

The processor usage data collected in the experiments give us a good idea how many services, modeled by the number of threads in the diagrams, a RaaS can host, and how many requests, modeled by the sleep times/request intervals in the diagrams, a RaaS can handle.

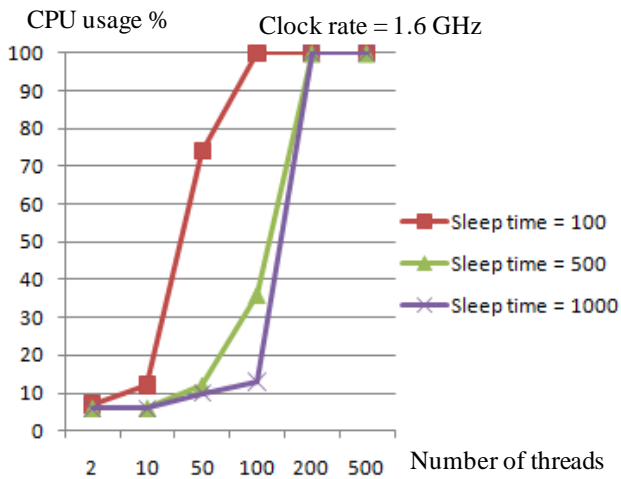


Figure 12. Data from Intel Atom N270 processor

The results obtained show that Intel processors are powerful enough to host different types of RaaS. Table 2 shows number of services that the processors can run before achieving their full capacity (100% usage).

TABLE 2. NUMBER OF SERVICES ON PROCESSORS

Processor	Clock	Arrival rate	No. of services
Core 2 Duo	1.6	5/second	200
	1.2	5/second	160
	0.8	5/second	80
Atom	1.6	5/second	100

Core 2 Duo	1.6	1/second	500
	1.2	1/second	350
	0.8	1/second	140
Atom	1.6	1/second	200

As can be seen from the table, the processors can handle several hundreds of parallel services even at high arrival rate. The services simulated in this table are simple services such as reading sensor values, sending control data to motors, and passing data to communication protocol. The time consuming services such as video processing are not simulated in the current experiment.

If a RaaS is running significantly lower number of services than its full capacity, the system should adjust its clock rate to a lower level to achieve desired performance while reducing processor heating and increasing power consumption. Processor temperature and power consumption rise exponentially to the clock rate increase. Similarly, one may make a choice between a multiprocessor and a single processor system as per the estimated load of the system.

We also collected data for other type of Intel processors, as well as the on power consumption for the processors. The data will be presented with the further experiments in the subsequent publications.

## VI. CONCLUSIONS AND FUTURE WORK

This paper defined the concept of Robot as a Service (RaaS), presented the design and implementation of a RaaS prototype. The features of the RaaS design include (1) All-in-one design of service provider, service broker, and service client in RaaS; (2) Generic hardware based Intel architecture and generic USB and common serial port devices; (3) Graphic composition based on Robotics Developer Studio and VPL language; (4) Multiple designs have been implemented on multiple platforms, including Java on Linux and C# and VPL on Windows. Extensive experiments and evaluations have been performed and the results show the flexibility of the design that can be easily ported to different systems. The experiments also show the effectiveness of the software and hardware system supporting the complex RaaS unit. A robotics starter kit is being developed jointly by Intel, Arizona State University, and Carl Hayden Community High School for dissemination to high school robotics camps and robotics competition. Other applications are also being explored.

## ACKNOWLEDGEMENT

Many students at Arizona State University participated in the design and implementation of different versions SOA robot and the new RaaS unit. Srushti Abhyankar, Edward Raleigh, Steven Rowe, James Woodford, Le Xu, and Jesus Yopez built a Java-Linux version in 2007. Terrence Cuny,

Scott Fisk, Benjamin Gauronskas, Adolfo Molina Von Ahn, and Joyce Tang extended the Java-Linux version to include more functions in 2008. Brad Bastian, Daniel Bogert, Tobias Hartana, Tim Kourt, Brian Partridge, and Eddy Suparjo implemented the C# version on Windows in 2008. Alan Hogan, Joseph Pfeister, and Ashutosh Sabnis implemented the two-robot duo in Figure 9(b) based on C#-Windows version in 2008. Chance Cox, Javier Colunga, Braymil Pavlovic, Ryan Mazur, and Satoshi Yoshida implemented the version of RaaS in Figure 9(a) in 2009. The final version of the robot (hardware) is built by the Robotics Club at Carl Hayden Community High School. The software developed on RaaS in Figure 9(a) is being moved to the new platform.

#### REFERENCES

- [1] Wikipedia, "Cloud Computing", [http://en.wikipedia.org/wiki/Cloud\\_computing](http://en.wikipedia.org/wiki/Cloud_computing)
- [2] H.E. Schaffer, "X as a Service, Cloud Computing, and the Need for Good Judgment", IT Professional, Volume 11, Issue 5, Sept-Oct. 2009 pp. 4 - 5.
- [3] Majitek, "Service Oriented Architecture for Machine-to-Machine Communication", retrieved from <http://www.majitek.com/technology/device-as-a-service/>, January 2010.
- [4] Yinong Chen and W.T. Tsai, Service-Oriented Computing and Web Data Management, Kendall/Hunt Publishing, 2010.
- [5] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall, August 2005.
- [6] Yinong Chen, "Service-Oriented Computing in Reconfigurable Embedded Systems", Joint IARP/IEEE-RAS/EURON/IFIP 10.4 Workshop on Dependability in Robotics and Autonomous Systems, Tucson, AZ, February 15-19, 2006.
- [7] Microsoft Corporation, "Microsoft Robotics Developer Studio", <http://www.microsoft.com/Robotics/>
- [8] W. T. Tsai, Qian Huang, Xin Sun, "A Collaborative Service-Oriented Simulation Framework with Microsoft Robotic Studio," 41st Annual Simulation Symposium (ANSS-41), pp. 263-270, 2008.
- [9] W.-T. Tsai, X. Sun, Q. Huang, H. Karatza, "An Ontology-Based Collaborative Service-Oriented Simulation Framework with Microsoft Robotics Studio", Simulation Modelling Practice and Theory, Elsevier, vol.16, no.9, 2008, pp.1392-1414.
- [10] V. Trifa, C. Cianci, D. Guinard, "Dynamic Control of a Robotic Swarm using a Service-Oriented Architecture", Proceedings of International Symposium on Artificial Life and Robotics. Bepu, Japan, January 2008.
- [11] P. Tröger, A. Rasche, F. Feinbube, and R. Wierschke. "SOA Meets Robots - A Service-Based Software Infrastructure For Remote Laboratories" In Proceedings of the 2nd International Workshop on eLearning and Virtual and Remote Laboratories, Germany, February 2008, pp.57--62.
- [12] K. C. Thramboulidis , G. Doukas , G. Koumoutsos, A SOA-based embedded systems development environment for industrial automation, EURASIP Journal on Embedded Systems, vol.2008 no.1, p.1-15, January 2008
- [13] Yinong Chen and X. Bai, "On Robotics Applications in Service-Oriented Architecture", The 28th IEEE International Conference on Distributed Computing Systems, ADSN Workshops, pp. 551-556.
- [14] Yinong Chen, S. Abhyankar, L. Xu, W.T. Tsai, Marcos Garcia-Acosta, "Developing a Security Robot in Service-Oriented Architecture", 12th IEEE Internal Workshop on Future Trends of Distributed Computing Systems (FTDCS), Kunming, China, Oct. 21-23, 2008, pp. 106-111.
- [15] Yinong Chen, A. Sabnis, and M. Garcia-Acosta, "Design and Performance Evaluation of a Service-Oriented Robotics Application" In Proceedings of the 29th IEEE international Conference on Distributed Computing Systems, ADSN Workshop, Montreal, June 2009, pp.292 - 299.
- [16] Arizona State University Software Research Laboratory "Intel Robotics Challenge", <http://asusrl.eas.asu.edu/srlab/Research/RoboticsChallenge.html>
- [17] ASU Summer Robotics Camps, <http://engineering.asu.edu/roboticscamp/>
- [18] Veiga, G., Pires, J.N., Nilsson, K., "On the use of SOA platforms for industrial robotic cells", Proceedings of Intelligent Manufacturing Systems, IMS2007, Alicante, Spain, June 2007, pp. 3-8.
- [19] Theodor Borangiu, "Trends in Service Oriented Architectures for Robot-CNC Manufacturing Control", <http://www.iit.bas.bg/PECR/59/10-31.pdf>
- [20] Masahiko Narita, Makiko Shimamura, Makoto Oya, "Reliable Protocol for Robot Communication on Web Services", International Conference on Cyberworlds (CW'05), pp.210-220, 2005.
- [21] Wikipedia, "Robot", <http://en.wikipedia.org/wiki/Robot>
- [22] Wired blog, Robot Cops to Patrol Korean Streets", January 17, 2006, [http://www.wired.com/gadgetlab/2006/01/robot\\_cops\\_to\\_p/](http://www.wired.com/gadgetlab/2006/01/robot_cops_to_p/)
- [23] Robot waiters, <http://www.technovelgy.com/ct/Science-Fiction-News.asp?NewsNum=771>
- [24] Robot pets, <http://en.wikipedia.org/wiki/AIBO>
- [25] Ian Hamilton, Robot to be added at Hoag Hospital Irvine, InTouch News, October 8, 2009, <http://www.intouchhealth.com/>
- [26] W. T. Tsai, Yinong Chen, C. Cheng, and X. Sun, G. Bitter and M. White, "An Introductory Course on Service-Oriented Computing for High Schools", Journal of Information Technology Education, Volume 7, pages 323-346.
- [27] Microsoft, Microsoft Robotics, <http://msdn.microsoft.com/en-us/robotics/default.aspx>
- [28] RaaS maze demonstration, recorded in Dec. 2009: <mms://asusrl.eas.asu.edu/srlab/Research/Robots/VPL/RobotMazeDec09.wmv>
- [29] RaaS cooperation demonstration, Dec. 2008, <mms://asusrl.eas.asu.edu/srlab/Research/Robots/PartyBots/PartyRotsASU.wmv>
- [30] ASU, "Smart home project service repository", <http://smarthome.engineering.asu.edu/content/services>
- [31] J. D. Oliver, M. E. Garcia-Acosta, J. Harris, F. Lajvardi, Yinong Chen, L. Gutierrez, "Robotics Starter Kit Using Intel Architecture", Intel White Paper, Mar 2010, <http://download.intel.com/design/intarch/papers/323505.pdf>