AUTOMATYKA/ AUTOMATICS • 2013 • Vol. 17 • No. 2

http://dx.doi.org/10.7494/automat.2013.17.2.187

Grzegorz Karpiel*, Konrad Gac*, Maciej Petko*

FPGA Based Hardware Accelerator for Parallel Robot Kinematic Calculations**

1. Introduction

Quality improvement of a manufactured product is a natural process accompanying the development of civilization. Nowadays, designed machines are much more accurate and efficient. In addition, the flexibility of production needs to increase functionality built machines. Among all machines assigned for manufacturing purposes, milling machines play a significant role. Common three-axis milling machines characterized by simple kinematics are being increasingly replaced by universal machines with multiple degrees-of-freedom, as their versatility allows implementation of a variety of tasks, such as turning or milling in different planes. While the machines accuracy is determined by its mechanics parameters and applied control system, the flexibility and performance are often determined by the limitations of computational power of the milling machines control system. When choosing a parallel structure as a support structure for the milling machine, kinematics calculations, necessary for the correct tool movement along the desired trajectory path, are a major concern. Those calculations should be performed in real time, at a frequency level of the operating driver system. In this paper the increase of computation power, when determining the kinematics of the milling machine based on a parallel robot, with the usage of a FPGA system equipped with a processor with additional dedicated instructions, is presented.

1.1. Parallel robots

A parallel robot is a mechanism, which arms consisting of any number of kinematics chains are connected by a joint or a platform [1]. Closed kinematics gives the construction much more rigidity and enables it to work more precisely and faster than constructions with open kinematics. A disadvantage of this structure is a smaller workspace. The workspace is limited by an arm movement range and it is a common part of sets of points possible to achieve by each arm.

* AGH University of Science and Technology, Krakow, Poland
** The project was funded from The National Centre for Research and Development

1.2. Kinematics

To obtain the full advantages of the robots' movement capabilities, the forward and inverse kinematics must be determined. The solution of the forward kinematics gives a clear rule defining the position of the tools tip in Cartesian space, depending on the position of each arm. Much more useful information is provided by their reverse relationship, being the solution of the inverse kinematics, which for a given point in space defined by the Cartesian space gives the position of the robots joints in configuration which the desired point is being reached. For the parallel robots, it is much easier to find the solution of the inverse kinematics, than the forward one [2]. By knowing the solution of the inverse kinematics, the robot can be treated as a Cartesian structure and with tools such as NC code generators for milling machines can be therefore used directly. The only drawback of an approach such as this is the calculation of the inverse kinematics, converting coordinates from the Cartesian space directly in to the joint lengths. It should be noted that the calculations must be carried out at any point of the trajectory, in real time, with the operating frequency of the trajectory generator.

2. The parallel robot for milling

The construction of a parallel robot for milling (Fig. 1) was developed in Department of Robotics and Mechatronics AGH [3]. The robot consists of three arms connected together with a moving platform. Inside the platform there is a spindle rotating a milling bit at 40 000 rpm. This can be classified as high-speed machining (HSM). HSM means using spindle speeds that are significantly higher than those used in conventional machining operations. Typical HSM spindle velocities range between 8 000 and 35 000 rpm, although some spindles nowadays are designed to rotate at over 100 000 rpm.

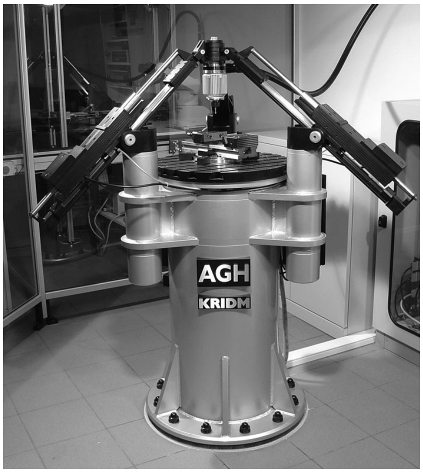


Fig. 1. Parallel robot with three degrees of freedom for milling

2.1. Kinematic structure

Figure 2 presents the kinematic structure of the robot. It consists of three arms I, II, III attached to the base on the vertices of an equilateral triangle inscribed into a circle with radius R. Each arm can protrude by changing lengths l_1, l_2, l_3. All rotary joints are based on universal joints.

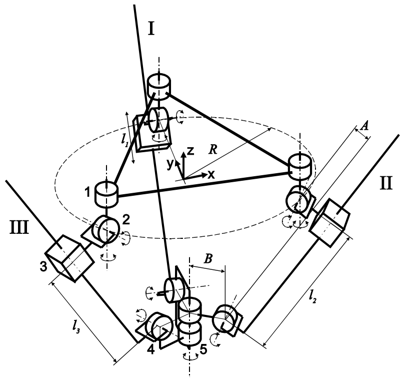


Fig. 2. The kinematic structure of the parallel robot

2.2. The solution of the inverse kinematics problem

For such a kinematic structure, the solution of the inverse kinematics problem was solved [4]. Distances A and B can be simplified [5]. The solution of the inverse kinematics problem takes form of the following equation sets: (1):

$$\begin{aligned} l_1 &= \sqrt{x^2 + (y-R)^2 + z^2} \\ l_2 &= \sqrt{\left(x - \frac{\sqrt{3}}{2}R\right)^2 + \left(y + \frac{1}{2}R\right)^2 + z^2} \\ l_2 &= \sqrt{\left(x + \frac{\sqrt{3}}{2}R\right)^2 + \left(y + \frac{1}{2}R\right)^2 + z^2} \end{aligned} \tag{1}$$

where:

x, y, z – the position of the tool in the Cartesian space,
R – radius of the circle on which robot arms are placed,
l_1, l_2, l_3 – arms lengths.

For the full realization of the trajectory in the workspace derived equations (1) should be solved. These derivatives allow for the transformation of velocity and acceleration of the tool in the Cartesian space to the speed and acceleration of the drives (2)–(4)

$$v_1 = \frac{1}{2}\frac{2\cdot x\cdot v_x + 2\cdot(y-R)\cdot v_y + 2\cdot z\cdot v_z}{\sqrt{x^2+(y-R)^2+z^2}}$$

$$v_2 = \frac{1}{2}\frac{2\cdot\left(x-\frac{\sqrt{3}}{2}R\right)\cdot v_x + 2\cdot\left(y+\frac{1}{2}R\right)\cdot v_y + 2\cdot z\cdot v_z}{\sqrt{\left(x-\frac{\sqrt{3}}{2}R\right)^2+\left(y+\frac{1}{2}R\right)^2+z^2}} \tag{2}$$

$$v_3 = \frac{1}{2}\frac{2\cdot\left(x+\frac{\sqrt{3}}{2}R\right)\cdot v_x + 2\cdot\left(y+\frac{1}{2}R\right)\cdot v_y + 2\cdot z\cdot v_z}{\sqrt{\left(x+\frac{\sqrt{3}}{2}R\right)^2+\left(y+\frac{1}{2}R\right)^2+z^2}}$$

where:

v_x, v_y, v_z – tool velocity in the Cartesian space,
v_1, v_2, v_3 – drives velocity.

$$a_1 = -\frac{1}{4}\frac{\left(2\cdot x\cdot v_x + 2\cdot(y-R)\cdot v_y + 2\cdot z\cdot v_z\right)^2}{\left(x^2+(y-R)^2+z^2\right)^{\frac{3}{2}}} +$$

$$+\ \frac{1}{2}\frac{2\cdot v_x^2 + 2\cdot x\cdot a_x + 2\cdot v_y^2 + 2\cdot(y-R)\cdot a_y + 2\cdot v_z^2 + 2\cdot z\cdot a_z}{\sqrt{x^2+(y-R)^2+z^2}}$$

$$a_2 = -\frac{1}{4}\frac{\left(2\cdot\left(x-\frac{\sqrt{3}}{2}R\right)\cdot v_x + 2\cdot\left(y+\frac{1}{2}R\right)\cdot v_y + 2\cdot z\cdot v_z\right)^2}{\left(\left(x-\frac{\sqrt{3}}{2}R\right)^2+\left(y+\frac{1}{2}R\right)^2+z^2\right)^{\frac{3}{2}}} + \tag{3}$$

$$+\ \frac{1}{2}\frac{2\cdot v_x^2 + 2\cdot\left(x-\frac{\sqrt{3}}{2}R\right)\cdot a_x + 2\cdot v_y^2 + 2\cdot\left(y+\frac{1}{2}R\right)\cdot a_y + 2\cdot v_z^2 + 2\cdot z\cdot a_z}{\sqrt{\left(x-\frac{\sqrt{3}}{2}R\right)^2+\left(y+\frac{1}{2}R\right)^2+z^2}}$$

$$a_3 = -\frac{1}{4}\frac{\left(2\cdot\left(x+\frac{\sqrt{3}}{2}R\right)\cdot v_x + 2\cdot\left(y+\frac{1}{2}R\right)\cdot v_y + 2\cdot z\cdot v_z\right)^2}{\left(\left(x+\frac{\sqrt{3}}{2}R\right)^2+\left(y+\frac{1}{2}R\right)^2+z^2\right)^{\frac{3}{2}}} + \frac{1}{2}\frac{2\cdot v_x^2 + 2\cdot\left(x+\frac{\sqrt{3}}{2}R\right)\cdot a_x + 2\cdot v_y^2 + 2\cdot\left(y+\frac{1}{2}R\right)\cdot a_y + 2\cdot v_z^2 + 2\cdot z\cdot a_z}{\sqrt{\left(x+\frac{\sqrt{3}}{2}R\right)^2+\left(y+\frac{1}{2}R\right)^2+z^2}} \tag{4}$$

where:

a_x, a_y, a_z – tool acceleration in the Cartesian space,
a_1, a_2, a_3 – drives acceleration.

3. Accelerator of kinematic calculations

The equations presented in the previous section allow to transform the trajectory generated in the Cartesian space into parallel robot's trajectory in joint space. In the case of implementing equations in actual controller the most important factors are related to accuracy and speed of calculations. In this regard an interesting solution is the implementation of the kinematics equations in a controller based on FPGA [6, 7].

3.1. Desired trajectory

An example of the trajectory is shown in Figure 3. Was assumed, that trajectory is a horizontal circle with 0.150 m radius situated on the height of 0.3 m.

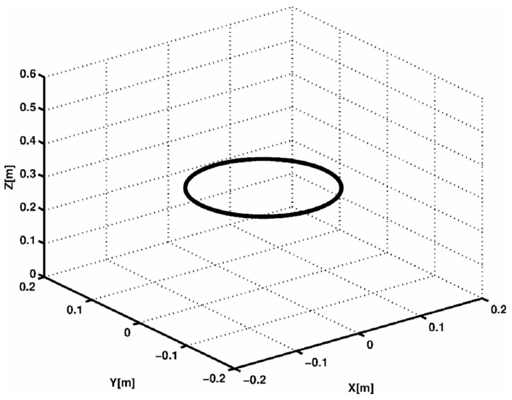


Fig. 3. Tool path in Cartesian space

Figure 4 shows the waveforms corresponding to the trajectory in Cartesian space and the joint space. Each point of the circle in the joint space was described by the nine parameters.: position *px*, *py*, *pz*, velocity *vx*, *vy*, *vz* and acceleration *ax*, *ay*, *az*. For each point calculated lengths *l*1, *l*2, *l*3, also the speed *v*1, *v*2, *v*3 and accelerations *a*1, *a*2, *a*3 were calculated. These points are determined by solving the inverse kinematics, which implemented on a PC with the use of double-precision floating-point variables (double).

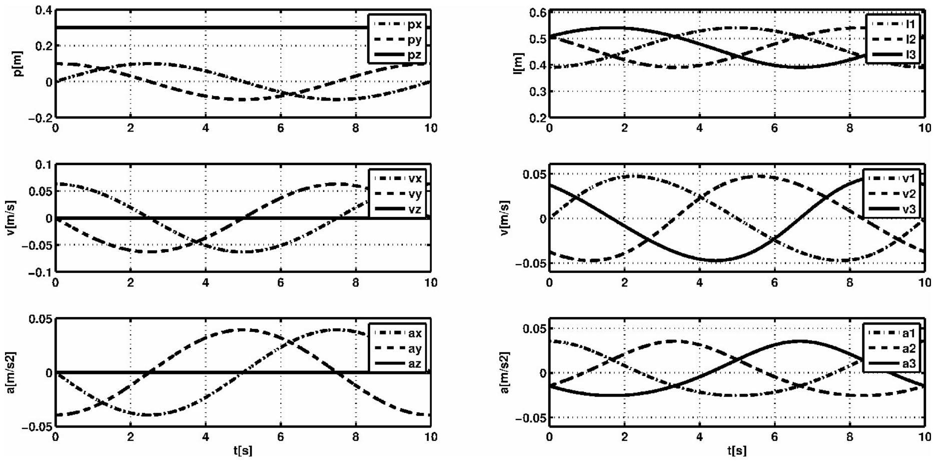


Fig. 4. Trajectory in Cartesian space (left) and in the joint space (on the right)

3.2. Implementation

Equations (1)–(4) were implemented in Altera FPGA chip [8]. Terasic DE2-115 board was chosen as a platform for testing (Fig. 5). Clock frequency of the Cyclone IV was set to 50 MHz.

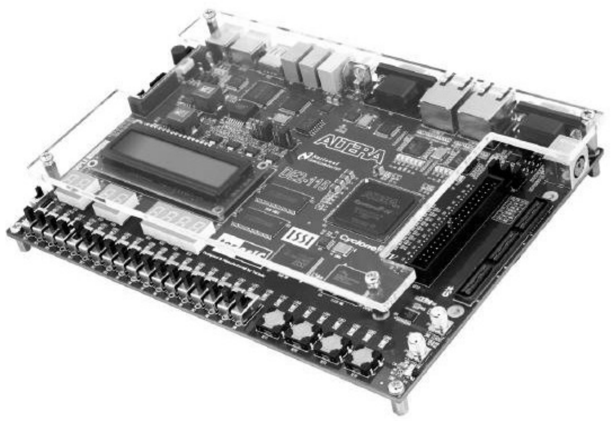
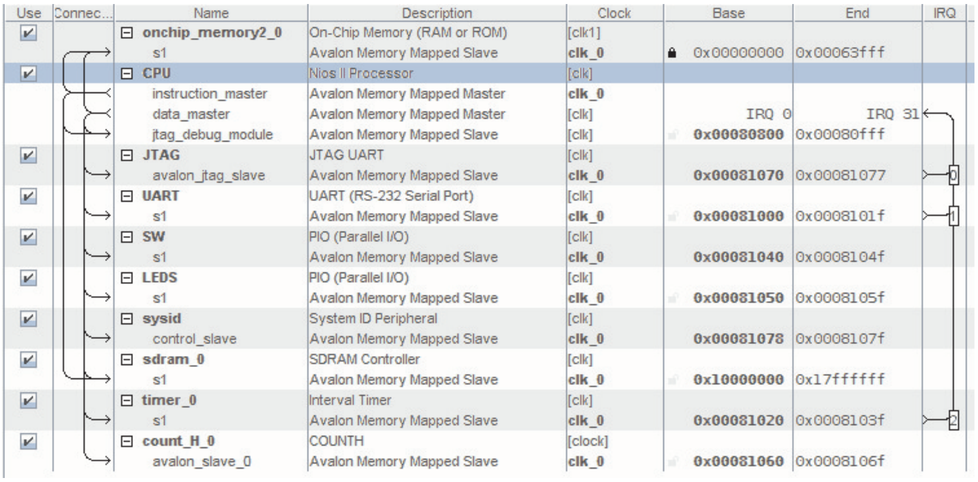


Fig. 5. DE2-115 board – test platform

In the first step by using the tools included in the package QSYS microprocessor system based on the Nios II processor was built (Fig. 6). It contains the memory of "OnChip" and data "sdram" on trajectory. Communication with a PC and the JTAG interface and an RS232 port located on the DE2-115.



Use	Connec...	Name	Description	Clock	Base	End	IRQ
✓		onchip_memory2_0	On-Chip Memory (RAM or ROM)	[clk1]			
		s1	Avalon Memory Mapped Slave	clk_0	0x00000000	0x00063fff	
✓		CPU	Nios II Processor	[clk]			
		instruction_master	Avalon Memory Mapped Master	clk_0			
		data_master	Avalon Memory Mapped Master	[clk]	IRQ 0	IRQ 31	
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x00080800	0x00080fff	
✓		JTAG	JTAG UART	[clk]			
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_0	0x00081070	0x00081077	0
✓		UART	UART (RS-232 Serial Port)	[clk]			
		s1	Avalon Memory Mapped Slave	clk_0	0x00081000	0x0008101f	1
✓		SW	PIO (Parallel I/O)	[clk]			
		s1	Avalon Memory Mapped Slave	clk_0	0x00081040	0x0008104f	
✓		LEDS	PIO (Parallel I/O)	[clk]			
		s1	Avalon Memory Mapped Slave	clk_0	0x00081050	0x0008105f	
✓		sysid	System ID Peripheral	[clk]			
		control_slave	Avalon Memory Mapped Slave	clk_0	0x00081078	0x0008107f	
✓		sdram_0	SDRAM Controller	[clk]			
		s1	Avalon Memory Mapped Slave	clk_0	0x10000000	0x17ffffff	
✓		timer_0	Interval Timer	[clk]			
		s1	Avalon Memory Mapped Slave	clk_0	0x00081020	0x0008103f	2
✓		count_H_0	COUNTH	[clock]			
		avalon_slave_0	Avalon Memory Mapped Slave	clk_0	0x00081060	0x0008106f	

Fig. 6. SoPC structure designer in QSYS application

Three systems were created. The first one is without arithmetic coprocessor, in which floating-point operations are executed in software with single precision using math library provided by a compiler. The second system was equipped with a floating-point coprocessor of single precision provided by Altera. This coprocessor enables to perform operations like hardware summation multiplication, subtraction and division. The last system is also equipped with the arithmetic coprocessor but is expanded with a custom instruction enabling hardware calculation of the square root. Written in Verilog algorithm of floating-point square root [10] is reduced to the calculation of the square root of an integer, which results was determined by a mathematical relation (5):

$$\sqrt{a \cdot 2^b} = \sqrt{a} \cdot 2^{\frac{b}{2}} \tag{5}$$

In the calculations of kinematic equations, implemented in C programming language, the root was replaced by a previously defined macro instruction:

```
#define  Sqrt(A)  __builtin_custom_fnf(0x000000000,(A))
```

The compiler in the process of compiling these instructions translated into a „custom" logic of hardware responsible for calculating the square root.

3.3. The results of the calculations

The calculation correctness obtained from the designed system, with its own square root calculation instruction has been tested by comparing the trajectories generated in the joint space with the previously calculated trajectory on a PC. It should be noted, that the Nios II processor performs all floating point operations with single-precision accuracy (float 32-bits). By comparing all the points of the obtained trajectories, the calculations error characteristic has been achieved along the trajectory (Fig. 7).

Different intensities of gray color show respectively the error for the first, second and third linear drive.

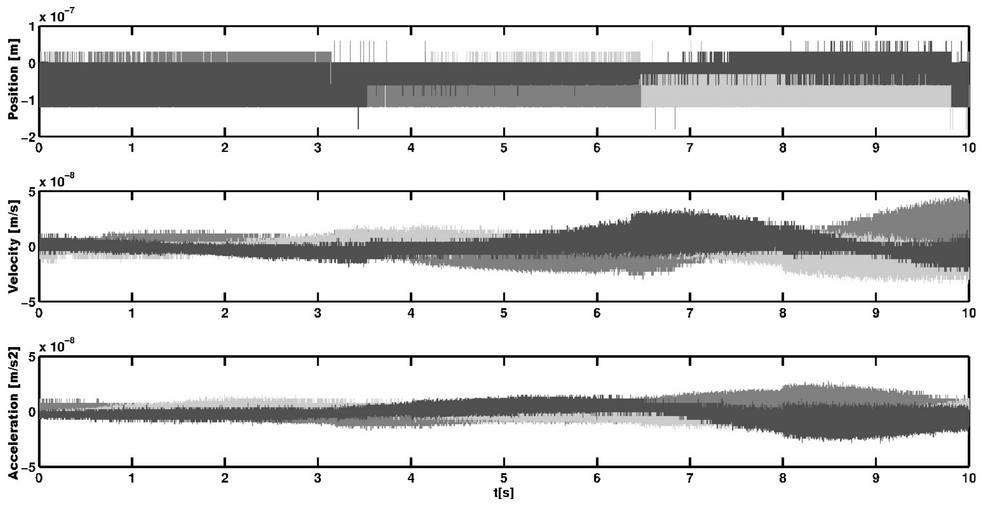


Fig. 7. Calculation error for position, velocity and acceleration

The resulting error of 50 nm for the position measurement is fully acceptable.

3.4. The time of calculations

Table 1 shows the changes in the calculation time and the amount of required FPGA resources depending on the microprocessor system version. For the first variation, the whole system need 4501 logical elements (LE) of FPGA. The calculation time for one point of the trajectory was 2324 μs. It means that without supporting the calculation by the arithmetic coprocessor, the maximum frequency is 430 Hz. In the case of standard hardware Nios II processor instructions for floating-point operations, the time of calculations shortens to 560 μs. It gives the calculation frequency of 1785 Hz. The addition of the arithmetic coprocessor slightly increased the amount of used resources.

The highest calculation efficiency was obtained by adding a custom instruction supporting square root calculation. The amount of used resources increased slightly (by about

1400 LE comparing to version with Altera coprocessor alone) however the time of calculation shortened to 143 μs. This enables the kinematic calculations to be made with frequency of 7 kHz. In typical aluminum machining by high-speed milling, the feed rate is 2 m/min. It means that in one second the tool moves by 33 mm. With a calculation frequency of 7 kHz, the distance between the points of trajectory is 4.7 μm. The distance is fully acceptable.

Table 1
Calculation time for a single point of the trajectory

Nios II	Calculation time [μs]	Total LEs used (114480 available)
No coprocessor	2324	4501 (4%)
With coprocessor	560	5840 (5%)
With coprocessor and custom square root instruction	143	7219 (6%)

4. Conclusion

In thise paper, knowledge of how to use a custom instruction for improving the performance of calculations when solving the inverse kinematics of a parallel robot has been presented. An example of an actually constructed parallel robot designed for milling applications has been shown, together with its geometric structure and solution of the inverse kinematics problem.

The accelerator featured in the paper expands the standard floating-point coprocessor for the Nios II processor with additional square root instruction. The calculation speed was quadrupled whereas the number of used logical elements increased by 1% for the Altera Cyclone IV FPGA chip. The obtained results, based on the developed accelerator, allow to build a driver, working at a frequency of 7 kHz and ensuring an accurate calculation of displacement up to 50 nm.

Further computational acceleration of the inverse kinematics would require the usage of an extended math coprocessor involving much more complex operations. The number of resources of currently available FPGA systems can fully perform the kinematics calculations by hardware. In addition, equations allow for a partial parallelization of individual operations, shortening the computation time. In future research, the authors plan to build a completely hardware accelerator for the kinetic calculations for the presented structure of the robot, based on fixed-point arithmetic.

References

[1] Gogu G., *Structural Synthesis of Parallel Robots. Part 1.* Springer, 2008, ISBN 978-1-4020-5102-9.

[2] Petko M., *Wybrane metody projektowania mechatronicznego*. AGH, Kraków-Radom, 2008, ISBN 978-83-7204-709-0.

[3] Karpiel G., Petko M., Uhl T., *Manipulator równoległy trzyramienny*. PL203631B1(B25J18/00).

[4] Karpiel G., *Zastosowanie podejścia mechatronicznego w projektowaniu robotów równoległych*. Ph.D. Thesis, AGH, 2007.

[5] Prusak D., Petko M., Karpiel G., *Manipulator trzyramienny o prostym modelu geometrycznym*. PL210002(B25J18/04;B25J9/06).

[6] Sanchez D., Munoz D., Llanos C., Motta J., *A Reconfigurable System Approach to the Direct Kinematics of a 5 D.o.f RoboticManipulator*. International Journal of Reconfigurable Computing, 2010, ArticleID 727909.

[7] Karpiel G., Prusak D., *System sterowania robotem równoległym oparty na układzie FPGA.* in: Projektowanie, analiza i implementacja systemów czasu rzeczywistego, Warszawa, Wydawnictwa Komunikacji i Łączności, 2011, ISBN: 878-83-206-1822-8.

[8] *http://www.altera.com/*, 2012-04-25.

[9] *http://www.terasic.com.tw/,* 2012-04-25.

[10] Piromsopa K., Aporntewan C., Chongsatitvatana P., *An FPGA implementation of a fixed-point square root operation.* http://pioneer.netserv.chula.ac.th/~achatcha/Publications/0012.pd, 2012-04-25f.

AUTOMATYKA/ AUTOMATICS • 2013 • Vol. 17 • No. 2

http://dx.doi.org/10.7494/automat.2013.17.2.207

Sławomir Nasiadka*, Henryk Krawczyk*

An Architecture of Execution Environment for Context-aware Applications Running in Intelligent Space**

1. Introduction

Context-aware applications are currently one of the most developed group of systems. They can be described using the CAA model presented in [1]. It allows to express the fact that activities occurring during the interaction between the application and its environment are not only limited to reading data from sensors. They are enriched by the analysis of the data, which can result in starting a particular adaptation action. The data creates a context which represents a situation in the space, and the action modifies the application behavior (i.e. adapts it to the new situation). Additionally, an analysis of the situation and taken action are carried out in real time, so that the CAA application responds quickly enough to changes in the space where it is executed. This requires an adequate architecture of that space. It is realized by providing a special engine in a form of a virtual machine, that defines the required mechanisms to support the CAA application execution [2]. With this machine, it is possible to design an architecture of the CAA execution space. The purpose of this article is to propose a model of such machine and an architecture of CAA execution environment (ICA – Interactive Component Architecture). Introduced machine is called the PCAA (Parallel CAA) and is derived from the PRAM machines [3]. This article describes an implementation of the execution environment created according to the proposed architecture.

In Section 2 it has been presented an overview of the literature on intelligent spaces and context-aware applications running in them. Section 3 introduces a definition of a context and the CAA applications model, as well as an automata that describes their behavior. It is the foundation for building a model of the PCAA machine presented in Section 4.

* Gdansk University of Technology, Gdansk, Poland

** The work was realized as a part of MAYDAY EURO 2012, Operational Program Innovative Economy 2007–2013, Priority 2 “Infrastructure area R&D”

An architecture of the CAA execution environment (ICA), which components are defined on the basis of mechanisms contained in the machine, is described in Section 5, while Section 6 presents its implementation. Section 7 provides a summary of the work.

2. Related work

An Intelligent Space (IS) is an environment where there are people and applications (collectively called space users), which aim is to support its users in their everyday duties [4]. To achieve this goal the space must have a possibility to carry out functions of data collection, analysis, and action execution. Data collection is performed by sensors, which read values of parameters of objects existing in the IS. Objects and parameters can be physical (e.g. temperature) and non-physical (e.g. a value of a cell in a database). To perform actions the IS uses actuators. Both sensing and acting, together with data analysis and processing, are performed by so-called DIND (Distributed Intelligent Network Devices) objects which are an essential part of any IS. The article assumes that those objects are wrapped and available for applications in the form of SOA services.

One of the most general layered model of context-aware applications running in the IS has been proposed in [5]. It distinguishes a sensor layer, a raw data layer, a pre-processing layer, a presentation layer and an application layer. Within the last layer there are performed an interpretation of a context and an execution of a proper application logic (e.g. a user interaction). This model allows for the separation and grouping of mechanisms needed for a context management during an application execution. An example of how those mechanisms can be organized is described in [6]. A programmer specifies which controllers are to be executed at a time of the occurrence of a specific situation. They are then dynamically triggered and executed at run-time. A similar approach was proposed in [7]. It is based on the concept of volunteers which is used as a way for selecting services in heterogeneous environments. It can therefore be noted that context-aware applications that exist in the IS may be executed within a special engine, that provides mechanisms supporting this execution (with respect to particular layers of the layered model). In the approaches described in the cited publications this engine is embedded in applications. Hence, it makes it not universal and changing the way it works requires programming modifications in the code of the application. This leads to a limited portability of applications between different IS and makes them hard to maintain.

To provide the necessary functionalities, the engine uses monitoring and control system mechanisms, which can be found in closed-control loop systems. However, in the case of context-aware applications execution those mechanisms are used to modify the application behavior so as to support its user more adequately in a new situation and not to control a particular system. Also, expert systems can be treated as context-aware applications. Depending on an expertise knowledge they are supplied with, that systems can make decisions taking into account a current context. From this point of view, the CLIPS, Drools and

Gensym/G2 tools allow to create context-aware applications. Moreover, functionality that they provide may be used in certain components of the CAA execution engine (e.g. analysis of the current situation in the IS). However, a complete implementation of the engine requires an extension of those mechanisms (such as the use of a knowledge base).

3. Definition of a context and a CAA application

An execution process of context-aware applications includes a continuous analysis of data from the IS. That data are called context data. They are delivered by sensors and create a context that reflects a situation in the IS. Depending on the situation, a particular adaptation action is executed. Such a definition has been adopted as the basis to build the CAA model of context-aware applications running in the IS. Its comprehensive concept is shown in Figure 1.

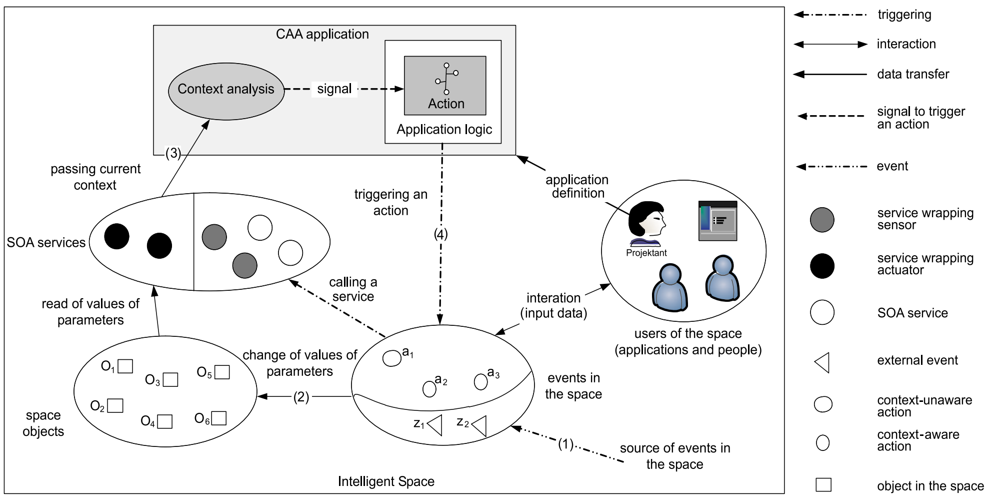


Fig. 1. CAA application running in the intelligent space

Situations occurring in the IS are caused by events that can be divided into external (i.e. those on which the application has no influence) and internal (i.e. those that are a result of the application execution – actions). Thus, external events start an execution of the application with regard to context-awareness (1). They cause changes in the IS (i.e. they change the values of parameters describing the state of the IS) (2). The state of a part of the IS that is important for the application is read by DIND objects which act as sensors (3). Importance means that the application should react for situations taking place in that part. Its state is then sent in a form of context data to CAA applications execution mechanisms.

The data are analyzed in terms of the meeting of so-called expected conditions of an action invocation. The result of this analysis can lead to trigger and execute adaptation actions (4). Executing an action is realized as calling a service that wraps an actuator or other services in terms of SOA. An executed action may interact with users of the IS and can create internal events. In addition, during execution of the action there may appear more external events (1). Effects of the action execution and external event occurrence can further change the state of objects (2) and, consequently, further actions are taken (creating iterative steps). Moreover, adaptation actions which are a result of the appearance of a specific situation means that the CAA application interacts with the IS. In Figure 1, there is distinguished a designer who determines to which situations in the IS (i.e. in what context), and how the application will react. One of the actions, which start is not dependent on the current context, starts at the beginning of the application execution and can last until it is finished. This represents the portion of the application logic that start is not associated with any situation in the IS. Thus, the application model contains two basic elements: the expected conditions of an action invocation and actions that are to be executed when conditions are met. They are grouped into pairs that create the CAA application definition.

The behavior of the CAA application can be represented using the automata shown in Figure 2. The automata is based on the Timed Automata [8] and is described in detail in [1].

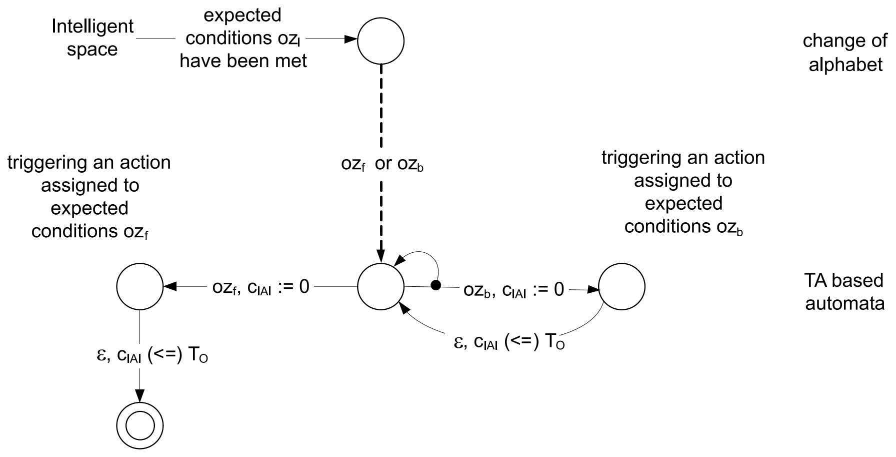

Intelligent space
expected conditions oz_l have been met
change of alphabet
triggering an action assigned to expected conditions oz_f
oz_f or oz_b
triggering an action assigned to expected conditions oz_b
oz_f, $c_{|A|} := 0$
oz_b, $c_{|A|} := 0$
TA based automata
ε, $c_{|A|}$ (<=) T_O
ε, $c_{|A|}$ (<=) T_O

Fig. 2. Automata that describes a behavior of a CAA application

The CAA application reacts when particular expected conditions of an action invocation (oz_l) are met. Those conditions create an input alphabet of the automata. Due to the fact that the conditions may be changed during run-time the alphabet is infinite. Therefore, it is necessary to convert it to a finite alphabet $\Sigma^* = \{oz_b, oz_f\}$. For this purpose, there has been defined a function $\delta\colon OZ \to \Sigma^*$ which represents the behavior of a transducer:

$$\delta(oz) = \begin{cases} oz_f, \text{ for } oz \text{ reflecting conditions finishing the application execution} \\ oz_b, \text{ in another case} \end{cases}$$

The approach of changing an input alphabet can be applied because from the point of view of the CAA applications execution a particular context (situation) does not need to be known. The most important is that the expected conditions have been met and the action that is associated with them was triggered. When the TA based automata receives the oz_b symbol it moves to the state of triggering an action. During this transition, the c_{IAI} clock is reset. Since the CAA are real-time applications, triggering of an action must be made within a period of time represented by the constant T_o (reflecting the time limit). After that time the automata should move without reading an input symbol to the state of waiting for another symbol. The (<=) operator checks whether a value of the clock in the left operand is less than the time period in the right operand. After the action has been triggered it must be executed in a CAA execution environment. Moreover, it is possible that more expected conditions will be recognized simultaneously (at the same moment in time) – then all actions associated with them must be triggered in parallel. This means that there is a need for parallel data analysis and processing as well as actions execution. Therefore, the automata moves in parallel to the state of waiting for the next symbol. In case of receiving the oz_f symbol a behavior of the automata is similar to the behavior when processing oz_b, but after the action is triggered the automata moves to a final state where it remains. It is assumed that the conversion of the alphabet is made in real-time. Because the application operates on expected conditions and actions, it means that the automata should take into account operations that regard selection and triggering of an action. Processes of recognizing whether the current context meets expected conditions, and executing an action are performed by an application execution environment. Created automata is the foundation for building a machine (PCAA – Parallel CAA), according to which such an environment should be organized. This is described in the next section, which defines what mechanisms the machine must provide to execute a CAA.

4. A PCAA machine

Within the PCAA machine there can be defined 4 PRAM machines, corresponding to mechanisms described in Section 3. Those mechanisms include operations that are executed in parallel. The first one (PRAM 1) controls the execution of other machines (PRAM 2, PRAM 3 and PRAM 4), stores objects which create a shared memory, and allows to communicate with the designer. Context data are delivered by services which wrap sensors and are received by the PRAM 2. They complement the application context, which is located in the shared memory. Then, it is analyzed in the PRAM 3 machine that checks whether any expected conditions of an action invocation contained in the application definition are met. An algorithm that is used for this analysis (which may also be described based on the PRAM machine) depends directly on the implementation and on the representation of the context. If the expected conditions are met, then (within the PRAM 4 machine) a particular action is triggered. The current context is passed to the action. Communication between the PRAM 3 and the PRAM 4 is based on signals. Thanks to the described organization of the PCAA machine, its execution can be done in parallel with regard to collecting context data, analyzing them, selecting and executing an action.

During the CAA execution, the PCAA machine can communicate with the following types of services that exist in the IS (they are called base services):

- services that wrap sensors, which deliver context data (delivered to PRAM 2),
- services that wrap actuators, used to execute actions (used by PRAM 4),
- supporting services, used during context analysis (used by PRAM 3).

It is assumed that all services located in the IS are of a type of SOA (see Fig. 1), so they can be used by the environment built according to the PCAA machine regardless of the executed CAA application.

Services existing in the IS may have different semantics (the meaning of data exchanged with the CAA execution environment) and syntax (different ways of data exchange). Therefore, with regard to the cooperation between the PCAA machine and heterogeneous services, it is necessary to introduce components that would allow to transform data to the uniform representation and adapt the way how the data are exchanged. This transformation must be performed when the data are delivered to the PCAA, where it is possible to analyze them in terms of checking whether they meet expected conditions of an action invocation. Similarly, when an action is triggered it is necessary to transform data from the uniform representation to the representation acceptable by a particular service. A general scheme of how the PCAA machine works is shown in Figure 3.

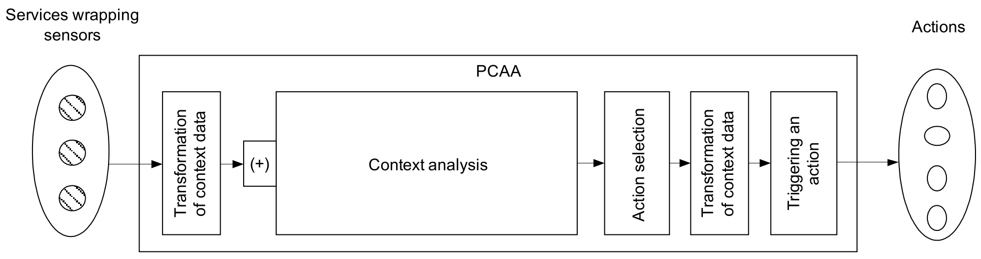

Services wrapping sensors
Actions
PCAA
Transformation of context data
(+)
Context analysis
Action selection
Transformation of context data
Triggering an action

Fig. 3. The process of context data analysis within the PCAA

The (+) operation represents the addition of contexts in terms of adding the data sent by the service to the application context. According to the scheme, there are following instructions (which are executed sequentially for each context data) performed within the PCAA:

- l_{ZD} – collecting context data,
- l_{TDK} – context data transformation,
- l_{AK} – context analysis (checking whether expected conditions of an action invocation have been met),

- l_{DA} – selecting an action (assigned to expected conditions),
- l_{TDA} – transformation of context data for particular action,
- l_{UA} – triggering an action.

Data collection (l_{ZD}) corresponds to gathering context data that are delivered from services wrapping sensors and it can differ depending on the way how the communication between an execution environment and services is organized. Then, context data are transformed (l_{TDK}), added into the application context and its analysis (l_{AK}) is performed. These instructions (l_{ZD}, l_{TDK} and l_{AK}) are associated with context data. If the result of the analysis is a recognition that the context meets expected conditions of an action invocation, a particular action (assigned to the conditions) is selected – l_{DA}. This instruction as well as l_{TDA} and l_{UA} are associated with conditions. From the point of view of the execution of a CAA application, there can also be defined additional instructions corresponding to gathering data from the IS through sensors and to action execution. However, they do not relate directly to the PCAA machine (see Fig. 3). It is worth to mention that because l_{DA}, l_{TDA} and l_{UA} instructions regard expected conditions of an action invocation rather than context data, a delivery of data does not always result in starting all PCAA instructions. This occurs only when the PCAA receives context data that create a context that further meets conditions.

CAA applications executed in the environment which is organized according to the PCAA machine is unloaded from operations related to the acquisition and analysis of the current context and execution of actions. This is shown in Figure 4, which summarizes the change in the way how CAA applications are constructed compared to traditional context-aware applications (see Section 2).

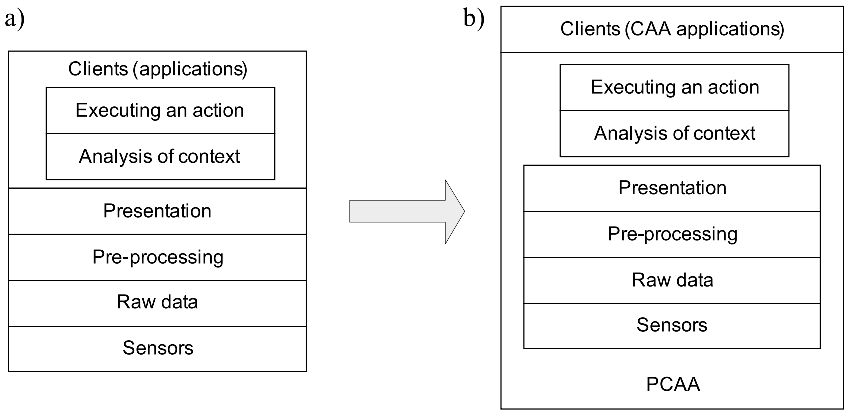


Fig. 4. Ways of constructing a traditional context-aware application (a) and a CAA application executed within the PCAA (b)

In the case of traditional context-aware applications (Fig. 4a) the analysis of the context and execution of the appropriate adaptation action was carried out by each application. Additional features, such as communication with sensors and pre-processing of raw data obtained from them, could also be implemented by the application or by some intermediate layer. In the approach which is based on the use of the PCAA machine (Fig. 4b), the machine provides a communication with services wrapping sensors, analyzes the context and executes the appropriate action. Thus, the CAA application definition contains a description of the rules how the machine should work while the PCAA provides adequate mechanisms of interpretation and implementation of this rules. The traditional context-aware application is therefore divided into a section describing the rules of its operation (corresponding to the CAA definition) and an executive part (the PCAA). Having defined the PCAA machine it is possible to define the components which make up the architecture of the CAA execution environment (the IS).

5. An architecture of the CAA execution environment

The architecture of the CAA execution environment, which results from the PCAA machine presented in Section 4, is presented in Figure 5.

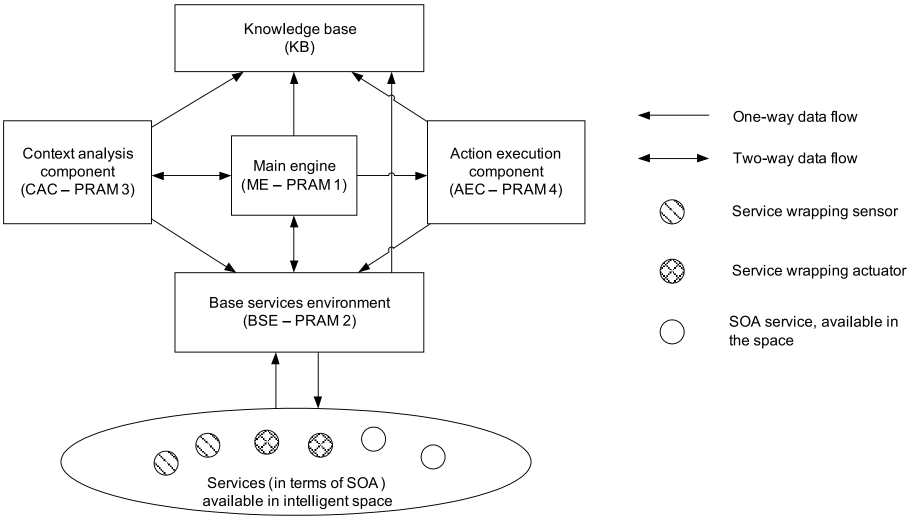

Knowledge base
(KB)
Context analysis component
(CAC – PRAM 3)
Main engine
(ME – PRAM 1)
Action execution component
(AEC – PRAM 4)
Base services environment
(BSE – PRAM 2)
Services (in terms of SOA) available in intelligent space
One-way data flow
Two-way data flow
Service wrapping sensor
Service wrapping actuator
SOA service, available in the space

Fig. 5. ICA architecture

The ICA architecture consists of the following components, which realize particular machine instructions (there are also given the corresponding PRAM machines):

– Main engine component (ME) – PRAM 1 – which is the central point of the system, through which it communicates with the designer of the application. It keeps the continuity of execution (controls operation of other components) and selects (based on expected conditions which are met) actions to be triggered – it carries out the l_{DA} instruction.

- Base services environment (BSE) – PRAM 2 – communication with base services is handled through this component – it carries out l_{ZD}, l_{TDK}, l_{TDA} instructions.
- Context analysis component (CAC) – PRAM 3 – component which is responsible for context data analysis and decision whether the context meets expected conditions of an action invocation – implements l_{AK} instruction.
- Action execution component (AEC) – PRAM 4 – is responsible for the execution of a selected action (l_{UA}).
- Knowledge base (KB) – a component that contains a knowledge about the IS. It is used by other components e.g. to interpret context data.

Particular components correspond to mechanisms defined in the PCAA machine which are necessary to execute CAA applications. They can be implemented in a distributed manner, so that each of them will operate independently and communicate with various components of the same type. In the next section there is described an implementation of the CAA execution environment developed according to the proposed architecture.

6. Implementation

The architecture presented in Section 5 has been used to develop the CAA execution environment called CAAEE (CAA Execution Environment). All components are realized as Web services and communication between them is based on managing sessions. CAC, BSE, AEC components and base services have been implemented in the .NET technology (they are located on the IIS server), and KB in Java (it is placed in the Apache Axis container). The knowledge is stored in the ontology which is described using OWL.

A CAA application designer provides its definition consisting of a set of pairs, each of which (according to the application model as described in Section 3) contains a description of expected conditions of an action invocation and a description of an action (in the form of a BPEL scenario). The ME component passes the definition of conditions to the CAC component. They are then converted into a Petrie network [9]. This network has the ability to receive data from sensors wrapped in the SOA services, to interpret them, to add to the context and to check whether it meets the conditions set by the designer. Data analyzed in the network may come from the real world or from other applications. They are processed (using XSLT transformations and the XPath language) in the BSE component and transformed to the uniform representation (using the ontology contained in KB), so that they can be analyzed in the CAC component. If the current context derived from delivered data meets expected conditions, a control signal is sent to the ME. The ME component selects an action assigned to recognized conditions and communicates it to the AEC (as a signal that contains context data received from the CAC). The action is then executed in the Apache ODE engine. At the time of its execution the CAAEE may use the services which are present in the IS.

7. Conclusions

A practical implementation of the PCAA machine requires a definition of components that support a CAA application execution. They create an implementable architecture for a CAA execution environment. This paper describes both the PCAA machine and the resulting ICA architecture. Its particular components can be prepared using different programming languages. Such implemented components together with mechanisms necessary for communication and service management creates the CAA execution environment, which is called CAAEE. Its proper operation has been checked using a set of sample base services. It has also been shown that the implementation works in the real world. This allows to create a fully functional environment, which is able to execute CAA applications in real-time. It is worth to note that this type of environment includes mechanisms that may also be found in modern control systems. However, they differ as to their purpose (i.e. a modification of the application behavior in order to more adequately support the user in case of the CAA) as well as the technology of an implementation.

The ICA architecture defines the functional components which correspond to the instructions defined in the PCAA model. Thus, it possible to define an application by an end-user, assuming that he does not have a programming knowledge. The user can prepare a set of rules according to which the application is supposed to behave, and the execution environment will take the responsibility of an implementation of these rules in the form of the executable application. By placing the machine directly in the intelligent space, the application execution becomes its integral function, which is in line with current trends in the development of intelligent spaces. Thus, the fact of an existence of the PCAA machine in the space makes it intelligent, while the degree of the intelligence depends on the applications running in it.

References

[1] Krawczyk H., Nasiadka S., *A New Model for context aware applications analysis and design*. The Fifth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, IARIA Conference, 2011, pp. 211–217.

[2] Nasiadka S., *Wpływ kontekstu na efektywność wykonania interaktywnych aplikacji iteracyjnych w dedykowanej przestrzeni usług* (*Influence of context on execution efficiency of interactive iterative applications running in dedicated services environment*). Doctoral dissertation, Gdansk University of Technology, 2012.

[3] Fortune S., Wyllie J., *Parallelism in random access machines*. Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, 1978, pp. 114–118.

[4] Lee J.H., Morioka K., Ando N., Hashimoto H., *Cooperation of Distributed Intelligent Sensors in Intelligent Environment*. IEEE/ASME Transactions On Mechatronics, vol. 9, No. 3, 2004, pp. 535–543.

[5] Ailisto H., Alahuhta P., Haataja V., Kyllönen V., Lindholm M., *Structuring Context Aware Applications: Five-Layer Model and Example Case*. Proceedings of the Workshop on Concepts and Models for Ubiquitous Computing, 2002.

[6] Rehman K., Stajano F., Coulouris G., *An Architecture for Interactive Context-Aware Applications*. Pervasive Computing, vol. 6, issue 1, 2007, pp. 73–80.

[7] Kim M.J., Kumar M., Shirazi B.A., Chong H.J., *A Novel Architecture for Provisioning Basic Services in Heterogeneous Pervasive Environments*. World of Wireless, Mobile and Multimedia Networks, 2007, pp. 1–4.

[8] Alur R., Dill D.L., *A theory of timed automata*. Theoretical Computer Science, vol. 126, issue 2, 1994, pp. 183–235.

[9] Krawczyk H., Nasiadka S., *Metoda wyznaczania kontekstu dla aplikacji sterowanych zdarzeniami* (*A method for deriving context for event driven applications*). Methods of Applied Computer Science, PAN, 2008, pp. 81–90.