A New Model for context-aware applications analysis and design*

Henryk Krawczyk, Sławomir Nasiadka
Faculty of Electronics, Telecommunications and Informatics,
Gdansk University of Technology
Gdańsk, Poland
hkrawk@eti.pg.gda.pl, slawomir.nasiadka@zak.eti.pg.gda.pl

***Abstract* — Context-aware applications that are working in intelligent spaces are taken into account and their properties are analysed. Based on this, the new approach to modelling and analysis of such applications is proposed that provides a separation between application logic regarding adaptation (to the environment) and its implementation. MVC and transition state models are considered. A quantitative measure of context-awareness level and a method of assessment of the adaptation time of the application is proposed. The relation between size of the context and execution time of a sample application is determined.**

Keywords – context-aware; application model; interactive.

I. INTRODUCTION

Intelligent spaces (IS) [8] are human centric computational environments [12] where applications and people exist together and are supported in their everyday tasks. They are realization of M. Weiser vision who defined a concept of ubiquitous computing [13]. According to him, the future of computer systems is a transparent integration with human living space. Another concept that is related to that vision is pervasive computing [17], which means a wide access to information with the usage of mobile devices that adapt to the space they exist in. As the purpose of the intelligent space is to support its users in efficient work on their tasks, applications working within such a space are user – centered (opposite to the classical computer – centered applications). Their main goal evolved from delivering functionality anytime anywhere to delivering it all the time everywhere. Hence, according to [12] the intelligent space can be summarized as a scalable and adaptable space designed for human and being aware of situations taking place within it, to which it should react. According to that description three main functions of the intelligent space are considered: observation, understanding and reacting [7].

The main idea of an intelligent space is presented in Figure 1 [9]. Users of the intelligent space (human or application) are surrounded by sensors and actuators, commonly named DIND (Distributed Intelligent Network Device) [6]. They can be treated as physical (for instance camera) or logical (service) objects that interact with the user. Thanks to them the user's behavior can be monitored and the space can deliver him non-physical (for instance information) or physical services – for example moving heavy objects. Processing data that comes from sensors

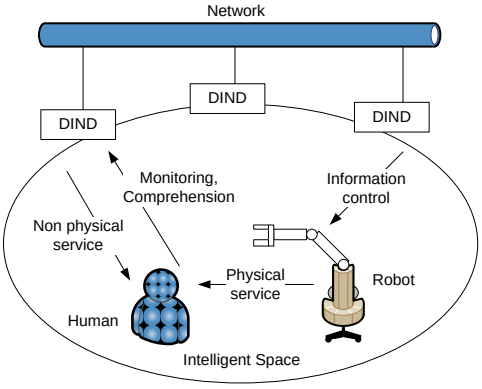

Figure 1. The idea of intelligent space and DIND – distributed networks of intelligent devices.

allows the space to understand what is happening inside it and react accordingly. However, both understanding and reacting is not performed by the space directly but through applications and DINDs. Hence, the level of its intelligence depends on DIND objects and applications deployed within the space.

Intelligent space, thanks to sensors, can gather data about its users. That data is creates the intelligent space state. Ubiquitous applications, embedded in the space, can use that state to appropriately modify their behavior and adapt it to new conditions that appeared within a space – becoming context-aware [9][10]. Such applications are called CAA (Context-Aware Applications). Process of their adaptation is iterative and takes place every time the state of the context changes. Hence, there is a need for a uniform model for such applications that emphasize their adaptability to the state of the IS. Because those applications are very often used by people that do not have programming knowledge the model should allow to create the applications by such users. In this article there has been proposed such a model. There is also presented evaluation of execution time as the function of the size of the context for the sample application built according to that model. The rest of this paper is organized as follows. Section 2 presents different approaches to create and model context-aware applications. In Section 3 and Section 4 we introduce a new model for CAA (MVC and more formal transition state based respectively). In Section 5 we present some early research on the usefulness of the model based on

* The work was realized as a part of MAYDAY EURO 2012 project, Operational Program Innovative Economy 2007-2013, Priority 2 „Infrastructure area R&D".

 ISBN: 978-1-61208-171-7

the evaluation of the sample CAA executed in a prototype implementation of the execution environment.

II. RELATED WORK

There are many approaches to creating context-aware applications and frameworks for their execution. In [18] there has been proposed such a framework that uses three-layer context model. Applications can use low and middle layer to compute a context that is usable for them. In [15] authors introduce a complete architecture for framework for execution of context-aware applications. It uses a rich context model (which consists of 4 main parts - user, device, environment, service) and treats all the context data in a form of individuals. Then the rule based engine operating on the ontology is used to derive additional knowledge and decide whether or not an appropriate context appeared in the application environment. Some authors propose alternative methods of designing applications that use context. The work presented in [16] proposes a Model Driven Development "to promote reuse, adaptability and interoperability in context-aware applications development. By concerns separation in individual models and by transformation techniques context can be provided, modeled and adapted independently of business logic and platform details". Another example is [11] where authors show how aspect oriented programming can be used to introduce context-awareness. [3] discusses differences between service oriented programming and context oriented programming as two alternative approaches to design, implement and maintain applications in general. More theoretical view on context-aware applications has been presented in [4] where authors describe such applications using mathematic formulas and advise that context-driven programming is the most suitable for context-aware applications. However, neither of mentioned papers combine formal model for context-aware applications with their implementation and presents how that implementation can be analyzed based on the model. That paper uses MVC model to present context-aware applications interactive nature as well as a graph state and automata base description which allows to analyze several aspects of such applications. Those aspects include execution time with regard to the context-awareness and a level of context-awareness. A proposed model does not focus on a particular context representation (that allows to use it as a generic model) but rather on a set of necessarily mechanisms and processes that have to be implemented to be able to execute any context-aware application.

III. MVC BASED MODEL

Context-awareness means that CAA have to adapt to changes that are taking place in the IS. There are many definitions of context but the most adequate has been proposed by K. Dey in [1]. According to him the context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. In the following article the context, after that definition, is understood as part of the intelligent space that is important from the application point of view. Importance for the application is further defined as usefulness, which means that part of the space is somehow useful for the application execution. The context can contain both physical (e.g. room, user) and logical (e.g. sequential number of sensor's read) entities, each of which has a set of parameters describing it.

There are different users (people, machines and applications) within the intelligent space that interact with it (for instance, they change position of objects, switch on devices and so on). Apart from them there are external events occurring in the space. Those events, as well as IS users, introduce some changes in the state of the space. A context-aware application has to observe those changes as they may be useful from its point of view (for its execution). However, changes in the state of the context are usually observed by applications indirectly, meaning that applications rather analyze current state of the context and based on it decide whether the adaptation action should take place. That adaptation process is permanent, and as such can be modeled as iterative, with the iteration step consisting of analyzing the state of the context and performing an adaptation action.

Let us consider a sample ubiquitous CAA, which controls an intelligent car with embedded GPS system. The main function of the car is always to provide a possibility to be driven by a driver and present him instructions about how to drive to a destination. However, depending on the driver and the situation on the road, the car can behave differently. Drivers prefer different methods of presentation of instructions where to drive (some of them prefer visual methods and some audio messages). When the car drives into a city a system that recognizes people on the road is enabled, and can automatically enable brakes when a person is near in front of the car when its moving. Apart from that all other systems that are used in the car are notified that it drove into the city. Drivers drive more or less dynamically, which in case of driving in the city should cause the usage of a gas engine (instead of electric). Car drivers also have some favorite music and a temperature level that they feel comfortable with during driving (which is set through car air conditioning system control), that the car should set automatically when a particular driver is recognized. Moreover, some of drivers are extrovert and emotional, and because of that they may become quickly nervous because of the situation on the road (for example traffic jam). Because this can easily become dangerous, the car should react playing some relaxing music. Taking into account all the above aspects of the behavior of the car it is a typical intelligent space (along with the road and its surroundings). That space can have deployed a CAA that controls its behavior by recognizing the current state of the space and performing adaptation actions appropriately.

Adaptation actions are part of a whole application logic, which also consists of actions independent from the state of the context (context-unaware part of the application). Hence, there can be distinguished two parts of the CAA: context-aware part that includes application logic performed according to the state of the context, and context-unaware part. In the example the context-aware part consists of

 ISBN: 978-1-61208-171-7

changing presentation method of instructions from GPS system, enabling a system recognizing people on the road, switching between electric and gas engine, changing music and temperature level and playing relaxing music. Context-unaware part, that can be treated as application's core functionality, is to provide a possibility to be driven by a driver, manually changing music and setting temperature level. Both parts can be organized as interactive processes, which means that any CAA works on two different kinds of data: state of the context and user's input data. There are two main interactive processing flows, that are taking place in CAA:

- users – input data – analysis – user action,
- events – state of the context – analysis – adaptation action.

The first flow represents context-unaware part of the applications (interaction with users as in the classical interactive application). In the example the user is a driver, input data is e.g., pressing an accelerator and brake pedal. Analysis of that input data is then made and the car behaves appropriately (e.g., turns, changes level of temperature, plays music) – which is a user action. The second class of processing flows regard context-aware part. In that case, sources of the data are events occurring in the intelligent space (indirectly, as data is read from the space by sensors). Those events can be generated by externals to the intelligent space (e.g., a traffic jam on the road makes the driver nervous) or internal factors (e.g., the space user's actions, like the driver entering the car). Because context can be treated as part of the intelligent space, data from those sources create the state of the context of CAA (e.g., driver is nervous, driver is John Smith), which is further analyzed, and if necessary the adaptation action can take place. That adaptation action can be executed in the same way as the context-unaware part of an application (as for example an interactive process), but the main difference is that it additionally uses context data. For the sample application the part of the adaptation process (context-aware part of the application) can be described as follows (see Table 1):

TABLE I. CONTEXT-AWARE PART OF THE SAMPLE CAA

Event	State of the context	Analysis	Adaptation action
Driver John Smiths entered the car.	Current driver is John Smiths, engine is not started.	Engine should be started.	Start the engine.
Engine has started.	Current driver is John Smiths, Engine is started.	John Smiths has his favorite music and temperature level.	Start playing John Smith's favorite music and set appropriate level of temperature in the air conditioning system.
Car's position changed.	Current position of the car is "E30ºN30º",	Position of the car has changed and there is a need to generate	Play new messages about further directions about the path to the destination. Set the car's location type to the city, and a enable system recognizing people on the road.
	Driver is John Smiths.	new GPS instructions to the driver John Smiths. The car is in the city now.	
Driver started to drive more dynamically.	Current driving style is dynamic. Car's location type is city.	More dynamic style of driving requires more power, which can be provided by the gas engine.	Start using the gas engine.
Traffic jam appeared on the road and the driver is becoming nervous.	Driver John Smiths is becoming nervous.	The driver John Smiths is less nervous when he listens to a special kind of music.	Start playing music that makes John Smith less nervous.

As can be seen, the adaptation process changes behavior of the CAA controlling the car. Depending on the current state of its context there are different actions performed. Hence, the execution of the CAA corresponds to the interactive process, and as such can be modeled using MVC model (see Figure 2). The CAA reacts to the appearance of the particular state of the context (modified by external events and read by sensors), which meets so-called expected conditions of an action invocation. When those conditions are met the appropriate action is invoked (and further executed). Checking whether conditions are met corresponds to the functionality of the View from the MVC model. The effect of the conditions met is a determination, of which action should be invoked. Here is another analogy to the MVC as, in that model, analysis of input data from the View is performed by the Controller. Hence, the choice of the action should also be performed by the Controller. The Model represents a logic contained in the actions that are invoked

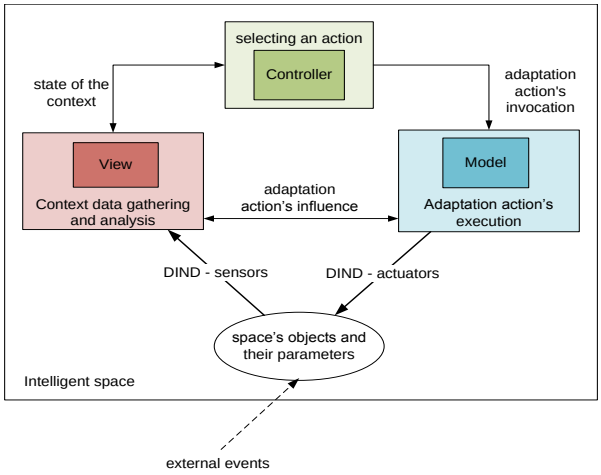

Figure 2. CAA presented using MVC model

by the Controller. The influence on their execution is determined by the state of the context that comes from the View and can be passed to actions during their invocation. Actions can also change objects in the intelligent space (values of their parameters), which influences the View. Changes in the Controller cause direct changes in the View (when expected conditions of an action invocation change). Also, actions are very often interactive as they are responsible for performing some change in the application behavior, which can consist of interaction with users or another application. Hence, for their design the MVC model can be used as well.

The key point in the CAA is to identify which states of the context of the application have to trigger an adaptation action. Because changes in the state of the context also represent changes in the state of the application (application performs some action), the CAA can also be described using transition state models, for which more precise definition of context is necessary.

IV. TRANSITION STATE MODEL

For each CAA application there can be distinguished three different types of context that have been taken into account during its execution. They are presented in Figure 3. The first one is an application context. According to the definition (from the Section 3) it can be any information that is useful from an application point of view. This means that this context represents a part of the intelligent space that is used by the application (processes its state or interacts with during execution of actions). For the sample application its context includes car's position, location type (in the city, or outside of the city), engine status, the driver and his emotional state (whether he is nervous) and style of driving (more or less dynamic). However, different actions need different data and are invoked under different conditions. The overall set of objects and parameters necessary to invoke the action and pass appropriate data is called an action context and part of it, responsible only for triggering the action, is called an action invocation context. The action context for the action of providing to the driver new instructions from the GPS system is a driver (who has preferences regarding graphical or audio presentation method) and the position of the car. However, action invocation context for the same action includes only the car's position. Those two are not the same because a change in the position of the car should lead to presentation of new instructions, but the information about the driver (necessary for choosing the method of the presentation) does not play a part in the determination of whether those instructions should be presented. In spite of which context is currently considered, it is always a part of the intelligent space, which consists of some objects and parameters describing them. For that reason the context can be formally described as a set of objects and associated parameters:

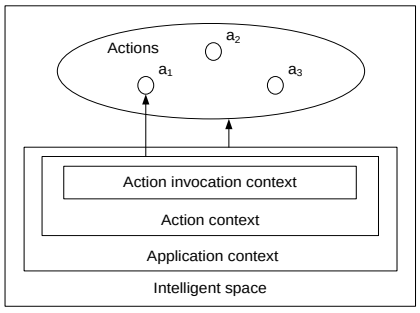

Figure 3. Different types of context in CAA

$$KT = <OK,PK,\alpha>, \quad (1)$$

where OK is a set of objects from the space, PK is a set of parameters of those objects and α is a function that assigns parameters to objects. For the sample application its context, and a context that corresponds to the action of presenting to the driver of new instructions from the GPS system (action's invocation and action context) can be presented as follows (Table 2):

TABLE II. TYPES OF CONTEXT FOR THE SAMPLE CAA

Type of context	OK	PK	α
Application	car, engine, driver,	position, has driver entered, location type, status, name, emotional state, style of driving	α(car) = {position, location type, has driver entered}, α(engine) = {status}, α(driver) = {name, emotional state, style of driving}
Action	car, driver	position, name	α(car) = {position}, α(driver) = {name},
Action's invocation	car	position	α(car) = {position}

With every type of the context there can be associated a corresponding state: state of an application's context, state of an action's context and state of an action's invocation context. State of the context is defined as values assigned to parameters of objects (from context):

$$SK = <KT,WK,\beta>, \quad (2)$$

where KT is a context, WK is a set of values that can be assigned to the parameters of objects belonging to the context, and β is a function that assigns a value to the parameter of the object (β:(OK,PK)–>WK). Example of the state of the application context (regarding the first line from Table 2) for the sample application is presented in Table 3. However, not every state of an action invocation context triggers an action. Those that trigger are called expected state of an action invocation context. In practice it is impossible to define all those states separately (for example describe all possible positions of a car). For that reason we will introduce

so-called expected conditions of an action invocation (*oz*), that define which conditions have to be met to trigger an

TABLE III. STATE OF THE CONTEXT FOR THE SAMPLE CAA

Type of context	KT	WK	β
Application	Defined in the Table 2 (first line)	"E30ºN30º", "city", "yes" "started" "John Smiths", "nervous", "dynamic"	β(car, position) ="W30ºN30º", β(car, location type) ="city", β(car, has driver entered) ="yes", β(engine, status) ="started", β(driver, name) ="John Smiths", β(driver, emotional state) = "nervous", β(driver, style of driving) = "dynamic"

action. They use objects and parameters from an action invocation context. For the action, regarding presentation of new directions from GPS system, the expected conditions of an action invocation consist of checking whether the position of the car has changed. Finally, the application can be defined as a set of such conditions (*oz*) and associated actions (*a*) that should be invoked when those conditions are met.

$$CAA = <OZ,A,\gamma>, \quad (3)$$

where OZ is a set of expected conditions of an action invocation, A is a set of actions and γ is a function that associates actions with conditions. If conditions are empty then actions are not context-aware, that means they are to be executed no matter what is the state of the context. That corresponds to the context-unaware part of the application. Hence, the application presented in Table 1 can be described as follows (see Table 4):

TABLE IV. DEFINITION OF THE SAMPLE CAA (CONTEXT-AWARE PART)

Expected conditions of an action's invocation (*oz*)	Adaptation action (*a*)
oz_1 - Whether a driver entered the car.	a_1 - Start the engine.
oz_2 - Whether the engine has started.	a_2 - Start playing driver's favorite music and set the appropriate level of the temperature in the air conditioning system.
oz_3 - Whether the car's position has changed.	a_3 – Present new instructions to the the driver about further directions about the path to the destination. Set car's location type to the city, and enable system recognizing people on the road.
oz_4 - Whether the car is in the city and the driver started to drive more dynamically.	a_4 - Start using the gas engine.
oz_5 - Whether the driver is nervous.	a_5 - Start playing music that makes the driver less nervous.

Using the above definition of CAA it is possible to measure its level of context-awareness which is a number of pairs: *oz* – *a*. The more such pairs an application includes (throughout its whole execution), the more context-aware it is. That is a quantitative measure that allows you to compare applications.

On the one hand, expected conditions of an action invocation allow you to shortly describe an application (rather than explicitly point-out all expected states of the context), and on the other hand they group expected states of an action invocation. By definition expected conditions are associated with a particular action. Comparing that structure to the classical iterative application [2] it can be seen that they are both very similar. The classical application can be described using some algorithm, in which every line has some label and performs some operations on a set of objects. Values of those objects create a state of the application in a particular line. In the CAA lines can be interpreted as pairs of expected conditions of an action invocation and associated action (each line is one pair). Further, within each line (pair) the operation is an action and objects (used by the operation) comes from the context. Expected conditions are unique for each line so they represent labels that identify a line. The state of the application can then be treated as the state of the intelligent space (part of it corresponding to the context). That further allows one to tie the state of the space with the state of the application. Example of a state transition graph for the sample CAA is presented in Figure 4. Nodes represent a set of expected state of an action invocation context that are grouped into expected conditions of an action invocation (*oz*). Each of expected conditions has assigned an action (*a*). Arcs represent transitions in the state, that can be a result of an action (bold line), external events (dashed line) or both (bold dashed line). Thin lines represent potential transitions. In Figure 4 there has been introduced an additional pair oz_6-a_6 that represents all potential pairs from the application definition that are not triggered during execution of the application (their conditions have never been met). What is typical for the CAA is that their state transition graph is always complete, which means every transition is possible. This is caused by external events that can occur every time during execution of the application (whether it performs an action or not). As a result the path that represents the actual application execution may vary

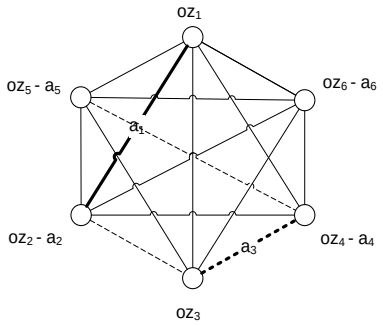

Figure 4. State transition graph for the sample CAA

between executions even when the same input data was passed from other users of the space to the application.

The state transition graph shows that CAA is actually a highly interactive application, whose execution consists of steps of recognizing whether expected conditions of an action invocation are met, and if yes, invoking (and further executing) an action. As such, CAA have to be real-time applications concerning the time of reaction on a change in the state of the context. Hence, CAA can be modeled using automata presented in Figure 5, which is based on the timed automata (TA) [14]. The CAA invoke an adaptation action when the current state of application context meets expected conditions of an action invocation (*oz*). For the automata that actually means infinite input alphabet because during application execution the new expected conditions can be added (e.g., by the user). To be able to use an approach based on the TA it is necessary to use transformation of input alphabet (known from data automata). That transformation is performed by a transducer, which associates with every symbol from infinite alphabet a symbol from a finite alphabet. That approach can be used because for CAA (regarding adaptation process), the exact state of the context or expected conditions are not important. The only important information is whether any expected conditions has been met (and associated action should be invoked). Function δ:OZ–>Σ^* represents a transducer behavior that changes

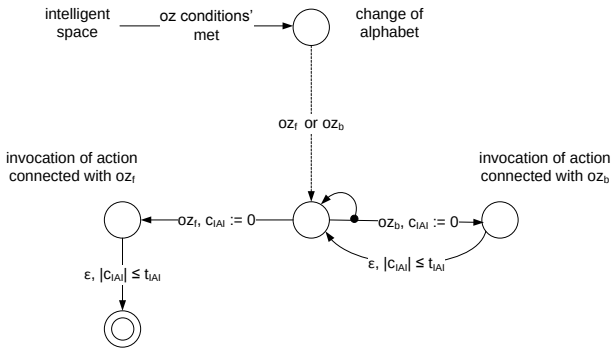

Figure 5. Automata describing CAA

expected conditions of an action invocation (*oz*) to a symbol from a finite set $\Sigma^*=\{oz_b, oz_f\}$. For each *oz* that equals the end conditions of the application the symbol oz_f is assigned and for all other conditions the symbol oz_b is used. When the latter symbol appears on the input of the TA based automata it moves into the state of action invocation. During that transition the clock *c* is reset (its value is set to 0). That clock is used to ensure that invocation of an action is done in real-time, which is represented by a condition $|c| < t$. The *t* is the amount of time, before which the automata has to move into the state of waiting for another symbol on the input (the transition has to be made without reading a symbol from the input). At the same time, when the action is invoked the automata has to be ready to read further symbols from the input. That is why it concurrently moves to the state of reading the symbols along with moving to the invocation state. In the case of the oz_f symbol the automata behaves similarly, however, after the action has been invoked the automata moves to the final state.

We have assumed that the transducer works on the already recognized expected conditions. That is because the automata models the application that consists of conditions and actions. Recognition of whether conditions have been met has to be done by the CAA execution environment. Similarly, the automata models only action invocation (not its execution). However, this is enough to assess the whole adaptation time of the application. Its upper boundary is a sum throughout all the pairs from application definition of recognition time whether expected condition of an action invocation has been met, *t* and time of execution of an action. That expresses the time of adaptation in the worst case – where all the pairs are executed sequentially. Computed value of adaptation time can change during application execution, because the application can change its definition by introducing new or deleting old pairs (e.g. as a result of one of its actions).

V. EVALUATION RESULTS

One of the interesting characteristics of the CAA applications is how the size of the context impacts their execution time. Size of the context can be expressed by a number of parameters of objects that have been used in the expected conditions of an action's invocation (for example oz_1 uses one parameters, but oz_4 uses two parameters). For the sample application the total number of all parameters used in all conditions (size of the context) is 7. Execution time is computed as a sum of processing times of gathering context data delivered by sensors, checking whether state of the context meets expected conditions from the application definition, and choosing an action. Time of an action invocation and execution has not been measured as actions are often external to the application (for example delivered by external suppliers). To be able to perform necessary measurement the CAA execution environment has been created. For processing state of the context there was used an engine described in [5]. The research has been made for different values of context size ranging from 10 to 200 with steps of 10, and all of them have been set at one time

 ISBN: 978-1-61208-171-7

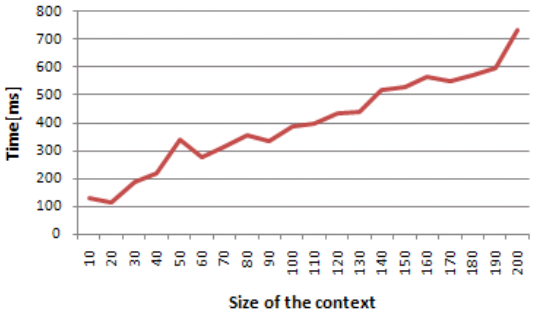

Figure 6. Execution time of the sample CAA

(appropriately to meet expected conditions). To be able to perform measurement for those sizes we have assumed that the emotional state of the driver (indicating whether he is nervous) is described using a set of objects (each of which contains a single parameter). To achieve repeatable results the CAA application has been constructed as a single pair (based on oz_5-a_5) in such a way that only a full context state (appropriate values of all parameters from all objects) triggered the adaptation action. The results are presented in Figure 6. As can be seen, the execution time is increasing (almost linearly) along with the increasing size of the context. Thanks to that, the processing time of the application can be easily estimated based only on its definition.

VI. CONCLUSION AND FUTURE WORK

The proposed model for CAA shows that this kind of applications are highly interactive and allows one to separate the application logic concerning adaptation to the current state of the application context, from an application implementation. Thanks to this, the model systematizes and supports a way of design and the implementation process of such applications. The definition of the CAA can be made by users that do not have programming knowledge. They only have to express rules about how the application should react to changes in the IS – define expected conditions and actions, for which some natural language processing can be used. Based on the model we have introduced a quantitative measure of context-awareness level for the CAA applications and present a method of assessing application adaptation time.

Future work will be focused on introducing reliability and quality mechanisms into the implemented execution environment. Some applications may have to be executed within a specified (by the user) time. As the adaptation time of the CAA can be assessed before its execution, the environment can choose appropriate trusted services (for both context state analysis and action execution).

REFERENCES

[1] Anind K. Dey and Gregory D. Abowd, "Towards a better understanding of context and context- Intelligent Space, Its Past and Future awareness", Technical Report GIT-GVU-99-22, Georgia Institute of Technology, College of Computing, 1999.

[2] Antoni Mazurkiewicz, "Problems of information processing" , WNT, Poland, 1974.

[3] Hen-I Yang, Erwin Jansen, and Sumi Helal, "A Comparison of Two Programming Models for Pervasive Computing", International Symposium on Applications and the Internet Workshops, 2006. SAINT Workshops, 2006, doi: 10.1109/SAINT-W.2006.1.

[4] Hen-I Yang, Jeffrey King, Abdelsalam (Sumi) Helal, and Erwin Jansen, "A Context-Driven Programming Model for Pervasive Spaces", Pervasive Computing for Quality of Life Enhancement Lecture Notes in Computer Science, 2007, pp. 31-43 vol. 4541/2007, doi: 10.1007/978-3-540-73035-4_4.

[5] Henryk Krawczyk and Sławomir Nasiadka, „A method for context determination for event driven applications", Metody Informatyki Stosowanej, PAN, Szczecin, Poland, 2008.

[6] Hideki Hashimoto, "Intelligent space - how to make spaces intelligent by using DIND?", IEEE International Conference on Systems, Man and Cybernetics, pp. 14-19, 2003, doi: 10.1109/ICSMC.2002.1167940.

[7] Joo-Ho Lee and Hideki Hashimoto, "Intelligent space," Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1358-1363 vol. 2, 2000, doi: 10.1109/IROS.2000.893210.

[8] Joo-Ho Lee and Hideki Hashimoto, "Intelligent Space, Its Past and Future", The 25th Annual Conference of the IEEE Industrial Electronics Society, pp. 126-131 vol. 1, San Jose, CA, USA, 1999, pp. 126–131.

[9] Kouhei Kawaji, Mihoko Niitsuma, Akio Kosaka, and Hideki Hashimoto, "Acquisition of objects' properties in Intelligent Space", SICE, 2007 Annual Conference, pp. 259-263, 2008, doi: 10.1109/SICE.2007.4420987.

[10] Matthias Baldauf and Schahram Dustdar, "A Survey on Context-aware systems", International Journal of Ad Hoc and Ubiquitous Computing, pp. 263-277 vol. 2 no. 4, 2004.

[11] Lidia Fuentes and Nadia Gámez, "Modeling the Context-Awareness Service in an Aspect-Oriented Middleware for AmI", Advances in Soft Computing, pp. 159-167 vol. 51/2009, 2009, doi: 10.1007/978-3-540-85867-6_19.

[12] Mohan M. Trivedi, Kohsia S. Huang, and Ivana Mikic, "Dynamic context capture and distributed video arrays for intelligent spaces", IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, pp. 145-163 vol. 35 Issue 1, 2005, doi: 10.1109/TSMCA.2004.838480.

[13] Päivi Kallio and Juhani Latvakoski, "Challenges and requirements of ubiquitous computing", WSEAS Transactions on Information Science and Applications, pp. 234-239 vol. 1 Issue 1, 2004.

[14] Rajeev Alur and David L. Dill, "A theory of timed automata", Journal Theoretical Computer Science, pp. 183-235 vol. 126 Issue 2, Elsevier Science Publishers Ltd. Essex, UK, 1994.

[15] Ramón Hervás and José Bravo, "COIVA: context-aware and ontology-powered information visualization architecture", Software—Practice & Experience, pp. 403-426 vol. 41 Issue 4, 2011, doi: 10.1002/spe.1011.

[16] Samyr Vale and Slimane Hammoudi, "Context-aware Model Driven Development by Parameterized Transformation", Architecture, pp.1-10, 2008.

[17] Stan Kurkovsky, "Pervasive computing: Past, Present and Future", ITI 5th International Conference on Information and Communications Technology, pp.65-71, 2008, doi: 10.1109/ITICT.2007.4475619.

[18] Thomas Pederson, Carmelo Ardito, Paolo Bottoni, and Maria F. Costabile, "A General-Purpose Context Modeling Architecture for Adaptive Mobile Services", ER '08 Proceedings of the ER 2008 Workshops on Advances in Conceptual Modeling: Challenges and Opportunities, pp. 208-217, 2008, doi: 10.1007/978-3-540-87991-6_26.

 ISBN: 978-1-61208-171-7