2011

# Detecting the Onset of Dementia using Context-Oriented Architecture

Basel Magableh
*Technological University Dublin*, 453543@tudublin.ie

Nidal AlBeiruti
nbeiruti@glam.ac.uk



## Recommended Citation

Magableh, B., AlBeiruti, N.(2011) Detecting the Onset of Dementia Using Context-Oriented Architecture. *5th International Conference and Exhibition on Next Generation Mobile Applications, Services, and Technologies (NGMAST'12),* 1, Paris, France, Sept. doi : 10.21427/2trz-2m29

2012

# Detecting the Onset of Dementia Using Context-Oriented Architecture

Basel Magableh

Nidal AlBeiruti

# Detecting the Onset of Dementia using Context-Oriented Architecture

Basel Magableh
School of Computer Science and Informatics,
University College Dublin
Dublin, Ireland
Email:basel.magableh@ucd.ie

Nidal AlBeiruti
Centre of Excellence in Mobile Applications and Services
Faculty of Advanced Technology,
University of Glamorgan, UK
Email: nbeiruti@glam.ac.uk

***Abstract*—In the last few years, Aspect Oriented Software Development (AOSD) and Context Oriented Software Development (COSD) have become interesting alternatives for the design and construction of self-adaptive software systems. An analysis of these technologies shows them all to employ the principle of the separation of concerns, Model Driven Architecture (MDA) and Component-based Software Development (CBSD) for building high quality of software systems. In general, the ultimate goal of these technologies is to be able to reduce development costs and effort, while improving the adaptability, and dependability of software systems. COSD, has emerged as a generic development paradigm towards constructing self-adaptive software by integrating MDA with context-oriented component model. The self-adaptive applications are developed using a Context-Oriented Component-based Applications Model-Driven Architecture (COCA-MDA), which generates an Architecture Description language (ADL) presenting the architecture as a components-based software system. COCA-MDA enables the developers to modularise the application based on their context-dependent behaviours, and separate the context-dependent functionality from the context-free functionality of the application. In this article, we wish to study the impact of the decomposition mechanism performed in MDA approaches over the software self-adaptability. We argue that a better and significant advance in software modularity based on context information can increase software adaptability and increase their performance and modifiability.**

***Index Terms*—model-driven architecture, context oriented programming, component composition, self-adaptive application, context oriented software development, ambient assisted living.**

## I. Introduction

Context-dependent applications refer to a class of software systems that are able to monitor and detect context changes in the environment where they operate. They can autonomously modify their own structure and behaviour in response to context changes [1]. Software in distributed and mobile computing environments needs to cope with variability as software systems are deployed on an increasingly large diversity of computing platforms and operate in different execution environments. Mobility induces context changes to the computational environments and therefore changes to the availability of resources, and continuously evolving requirements require software systems to be able to adapt to context changes [2]. Moreover, because of the software pervasiveness, and in order to make adaptation effective and successful, adaptation processes must be considered in conjunction with dependability and reliability by providing dynamic verification and validation of the adaptation output among the adaptation goals, objectives, and architecture quality attributes [2].

In the classical view of object-oriented software development, the modular structure for software systems has rested on several assumptions. These assumptions may no longer characterize the challenge of constructing self-adaptive software systems that are to be executed in mobile computing environments [3]. The most important assumptions in object-oriented development methodologies are that the decision to use or reuse a particular component/object is made at the time the software is developed. However, the development of a variety of modern self-adaptive software architectures such as mobile/ubiquitous computing, and component-based and context-oriented software has emphasized on deferring these decisions about component selection until runtime. This might increase the software capabilities in terms of variability, adaptability, and maintainability, and increase the anticipatory level of the software by loading a particular component/service that can handle unforeseen context changes dynamically.

The hypothesis presented here is that self-adaptive software engineering requires to consider the context information and context-dependent behaviours in the analysis, design and implementation of self-adaptive software. In particular, software composition must be considered in conjunction with context-dependent behavioural variations and the contextual changes, which provides context-driven adaptation and self-adaptability.

To achieve this target, Context Oriented Software Development (COSD) was proposed as a generic development methodology, which facilitates the development of self-adaptive context-oriented software. The COSD integrates a model-driven architecture approach (Context-Oriented Component-based Applications Model-Driven Architecture (COCA-MDA)) [4] with a behavioural decomposition strategy, based on the observation of context information in requirements analysis and modelling phase. As a result of combining a decomposition strategy with COCA-MDA, a set of behavioural units is produced. Each unit implements several context-dependent functionalities. The context-oriented component model (Context-Oriented Component model (COCA-component)) encapsulates code fragments of the context-

978-0-7695-4803-6/12 $26.00 © 2012 IEEE
DOI 10.1109/NGMAST.2012.50

dependent functionality in distinct components and decouples them from the core-functionality components.

Aspect Oriented Software Development (AOSD) [5] and COSD [4] are the alternatives for the design and construction of self-adaptive software. Their ultimate goal is to support the adaptability and variability of software systems, and to be able to reduce development cost and effort, while improving the software modularity and complexity. This motivates us to evaluate these technologies with respect to their ability to support software adaptability (modifiability) and the performance gain from using these technologies to implement the case study application in a mobile computing environment [4].

The rest of the article is structured as follows. Section II discusses behavioral variability support in context-oriented programming and aspects. Section III describes the Context-Oriented Software Development Paradigm. Section III-A demonstrates a case study designed using the COCA-MDA and implemented with the COCA-middleware. The Context-Oriented Software is evaluated in terms of energy utilisation and adaptation time as discussed in Section IV. The conclusions of this study and future works are discussed in Section V.

## II. Variability Management with Context-oriented Programming and Aspects

Mobile computing infrastructures make it possible for mobile users to run software services on heterogeneous and resource-constrained platforms. Heterogeneity and device limitedness create a challenge for the development and deployment of mobile services that are able to run in the execution context and are able to ensure that users experience the best quality of services according to their needs and specific contexts of use. Thus, it is desirable that self-adaptive software is able to reconfigure and re-optimise itself by recomposing components or services dynamically, according to the operational context [6].

Compositional adaptation enables a software system to adapt a new structure/behaviour for anticipating concerns that were unforeseen during the original design of the software. Normally, compositional adaptation can be achieved using the separation of concerns techniques, computational reflection, component-based design, and adaptive middleware [7]. The separation of concerns enables the software developers to separate the functional behaviour and the crosscutting concerns of self-adaptive applications. Crosscutting concerns are properties or areas of interest such as quality of service, energy consumption, location awareness, users' preferences, and security. The functional behaviour refers to the business logic of an application [7]. Context-dependent behavioural variations are heterogeneous crosscutting concerns and a set of collaborated aspects that extend the application behaviour in several parts of the program and their code have an impact across the whole software. Before encapsulating crosscutting context-dependent behaviours into software modules, the developers must first identify them in the requirements documents. This is difficult to achieve because, by their nature, context-dependent behaviours are tangled with other behaviours, and are likely to be included in multiple parts of the software modules. Using intuition or even domain knowledge is not necessarily sufficient for the developers to identify the context-dependent parts of self-adaptive applications.

Context-Oriented Programming (COP) provides a dynamic fine-grained behavioural adaptation mechanism, which uses a programming-level technique for performing the adaptation [8]. In COP, context can be handled directly at the code level by enriching the business logic of the application with code fragments responsible for performing context manipulation, thus providing the application with a code block, that implements the required behaviour [6]. The assumptions made by the COP approaches proposed in [8], i.e. that the developer knows all the possible software adaptations in advance and designs the application accordingly. As an outcome, the anticipated adjustment is restricted to the amount of code blocks offered by the developers. In addition, COP has no separation between the application's business and adaptation logic. the context model and the adaptation logic are explicitly hard-coded in the application's business code; this often leads to poor scalability and maintainability [6].

However, for a more complex context-dependent software system, the same context information would be triggered in different parts of an application and would trigger the invocation of additional behaviour. In this way, context manipulation becomes a concern that spans several application units, essentially crosscutting into the main application execution [9]. A programming paradigm aiming at handling such crosscutting concerns (referred to as aspects) is Aspect-Oriented Programming (AOP) [10]. Using the AOP paradigm, context information can be handled through aspects that interrupt the main application execution. The idea behind AOP is to implement crosscutting concerns as aspects whereas the core features are implemented as components. Using pointcuts and advice, an aspect weaver glues aspects and components together. Pointcuts specify the join points of aspects and components, whereas advice define which code is applied to these points. In this sense, the aspect-oriented development paradigm can be used to handle homogeneous behavioural variations where the same piece of code can be invoked in several software modules [11]. On the other hand, context-driven adaptation requires a set of collaborated aspects to be executed in several software modules i.e. Executing multiple code fragments in several parts of the program at the same time.

Tanter et al. [12] proposed context-aware aspects, which supports context-driven adaptation by designing pointcuts that depend on different context conditions, so that advices would only be executed in specific context conditions. Current AOP languages have limited support for context condition expression. First, they are not able to consider past context. Second, they are not able to express context-dependencies in aspects. Designing aspects that become active when particular contexts are verified, require the possibility to refer to a context definition in a pointcut construction. That means join points like

BeInContext(Context LocationCtx) should be provided by the framework. Another important ability of a framework should be giving an overview about all actual and past activated contexts, so that pointcuts can be designed on the base of this information. In other words, the AOP framework needs to keep track of past context conditions and their associated states. This is called context snapshotting [12], and the saved state of one context condition at a given point of time is called context snapshot. A global context snapshot is therefore a snapshot of all context conditions at a given point in time. Context snapshots are only made at a special point of time, because otherwise it would lead to high memory usage. Therefore the main problem is to define the right points of time to take such context snapshots. The actual current solution is to take snapshots of context conditions only if necessary as stated in the Reflex framework [13].

The following section proposes the COSD development methodology, that applies to the Model-driven Architecture style. The COCA-MDA methodology's phases and tasks are described in detail and show the process of constructing a case study application.

## III. Context-Oriented Software Development Paradigm

In order to overcome the problem and the challenges of engineering self-adaptive software, this article contributes to the knowledge by evaluating the impact of COSD over software adaptability. COSD was proposed as a generic and standard development paradigm towards constructing self-adaptive software from context-oriented components, which enables a complete runtime composition of the context-dependent behaviours and provides the software with capabilities of self-adaptability and dependability in mobile computing environment. Our model is based on a decomposition strategy of self-adaptive software based on context, which provides flexible mechanism for modularising the software into several composable units of behaviour and decouples the context-dependent from the context-free parts. Because each context-dependent functionality realises multiple volatile context-dependent behaviour. The context-oriented component model (COCA-component) encapsulates their implementation in distinct architectural units and provides several benefits for the software. This differs from the majority of contemporary works, which seek to embed awareness of context in the functional implementation of applications.

The adaptive software operates through a series of substates. The substates are represented by j, and j might represent a known or unknown conditional state *k*. Examples of known states in the generic form include detecting context changes in a reactive or proactive manner, so the developers are able to specify decision policy (k), which controls the adaptation in the associated state *($S_i$)*. Each decision policy (*k*) is attached to a decision point *(DP)j*, which controls the transformation *T(jk)* of the self-adaptive software form $state_i$ into $state_{i+1}$, when the application receives context changes *(Ci)* from the computational environment, as shown in Figure 1.

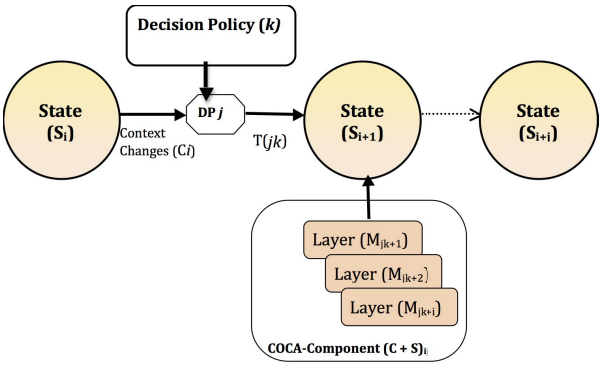


Fig. 1: Behavioural Decomposition Model

In the presence of uncertainty and unforeseen context changes, a self-adaptive application might be notified about an unknown condition prior to the software design. Such adaptation is reflected in a series of context-system states. $(C + S)_{ji}$ denotes the $i^{th}$ combination of context-dependent behaviour, which is related to the *Decision Point (DP)j* by the notion mode $M_{jk}$. In this way, the development methodology decomposes the software into a set of context-driven and context-free states. At runtime, the middleware transforms the self-adaptive software form $state_i$ into $state_{i+1}$, considering a specific context condition $T_{jk}$, as shown in Figure 1. This enables the developer to clearly decide which part of the architecture should respond to the context changes $T_{jk}$, and provides the middleware with sufficient information to consider a subset of the architecture during the adaptation. This enhances the adaptation process, impact, and cost and reduces the computation overhead from implementing this class of applications in mobile devices.

Context-driven adaptation requires dynamic composition of context-dependent parts, which enables the middleware to add, remove, or reconfigure components within an application at runtime. Each component embeds a specific context-dependent functionality $(C + S)_{ji}$, realized by a COCA-component. Each COCA-component realizes several layers that encapsulate a fragment of code related to a specific software mode $layer(M_{jk})$, as shown in Figure 1. The developers have the option to provide a decision policy $(k)$ for each *(DP)j* for a specific context-related condition. Hereafter, the COCA-components are dynamically managed by Context-Oriented Component-based Applications Middleware (COCA-middleware) and their internal parts to modify the application behaviour. The COCA-middleware performs context monitoring, dynamic decision-making, and adaptation, based on policy evaluation.

The COCA-middleware shown in Figure 2 performs the adaptation processes, including context monitoring and detecting and dynamic decision-making, and maintains the architecture quality attributes during the adaptation. The context manager performs context monitoring and detecting. It employs the observer design pattern for binding the context

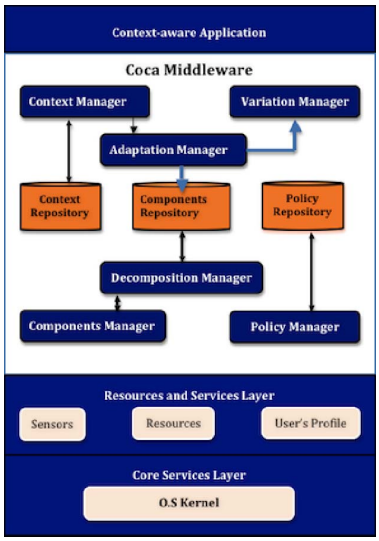


Fig. 2: COCA-platform architecture.

provider with the context consumer [14]. The adaptation manager performs dynamic decision-making, and adaptation, based on policy evaluation. The policy manager evaluates the decision policies that were predefined in the design phase. The verification manager verifies the adaptation among the underlying requirements, adaptation goals and decision polices; and it can verify the adaptation output among the available resources and the trade-off between the quality attributes of the architecture. The component manager instantiate the component implementation using the bundle design pattern [15]. Developing the application using COCA-MDA enables the COCA-middleware to determine which parts need to be changed and how to change them to achieve the best output and enable the component model to employ the delegation design pattern [15]. Developing the application using COCA-MDA enables the COCA-middleware to determine which parts need to be changed and how to change them to achieve the best output and enable the component model to employ the delegation design pattern [15]. The COCA-middleware design principles and the adaptation mechanism were described by Magableh and Barrett [14].

The COCA-MDA provides the developers with the ability to specify the adaptation goals, actions, and causes associated with several context conditions using a policy-based framework. For each COCA-component, the developers can embed one or more Decision PoLicys (DPLs) that specify the architecture properties. The DPL is described by a state-machine model based on a set of internal and external variables and conditional rules. The rules determine the true action or else an action based on the variable values. The action part of the state diagrams usually involves invoking one or more of the component's layers. A single layer is activated if a specific context condition is found, or deactivated if the condition is not found.

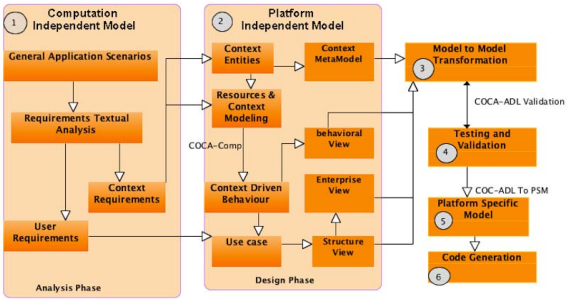


Fig. 3: Context-oriented component-based application model-driven architecture (COCA-MDA)

### A. *Self-adaptive Context-Oriented Component-based Application Example*

The Context-Oriented Component-based Architecture, and the development methodology were described in [4], [14]. This article focuses on the validation and evaluation of the context-oriented software. To this aim, we have considered Computational and Sensory Detection of Dementia (CaSDD) application as a case study described in the following scenarios.

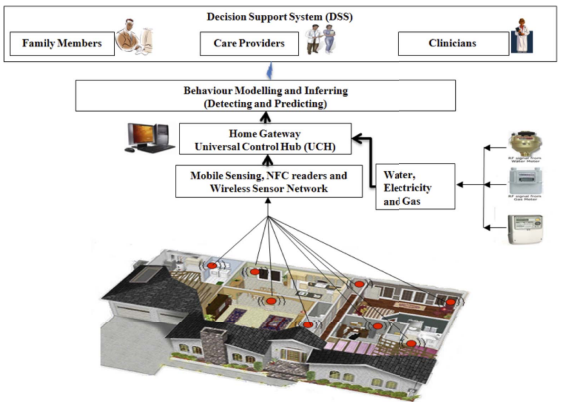


Fig. 4: CaSDD Architecture

The paradigm of Ambient Assisted Living (AAL) [16] has been gaining more interest recently. AAL paradigm mainly originates as a result of the merger between two streams which are assisted living and ambient intelligence. AAL solutions leverage the concept of aging-in-place which has proven to be very valuable especially under the current austerity measures all over the world and because of the increase in the ratio of care receivers to the number of social care providers [17]. As part of the big paradigm, elderly monitoring, in order to collect data about their medical or non-medial status, utilizes the vigorous technological leap in the communication, hardware and software arenas especially in the field of Wireless Sensor Networks (WSNs) and human behaviour modelling algorithms. The solution depicted in Figure 4 is aimed at detecting and predicting the onset of dementia by monitoring activities

of daily life of elderly people within their home environments. Abnormal behaviour is a very early symptom of the onset of dementia. Sensors can be deployed in each room within the home environment as illustrated in Figure 4. Simple sensors such as Passive Infra Red (PIR), temperature, luminosity and utilities' meters sensors are preferred to complex and invasive sensors such as video [17]. Data collected from those sensors will form a raw context that will be gathered on a centralized home gateway within the household. This raw context needs to be processed to remove any uncertainties and extract necessary features that will enable modelling the human behaviour for the elderly person who is living within that house. After building a statistical model and setting up thresholds, any abnormal behaviour can be detected and therefore may herald the onset of dementia. However, the system will integrate with third-parties as indicated in the figure and this underlines the important factor that the framework is not supposed to take decisions but to instigate actions.

The deployed sensors will generate valuable context data that can be used for multiple purposes, specially predicting and detecting abnormal behaviour of the subject (elderly person). This can only be achieved, if the software system is able to provide a reasonable conclusions about the subject's context data, which includes:

- The location/room
- The activity level within a room which is the number of firings of the PIR sensor:
  - Elapsed time between sensor' readings:
    - Two consecutive readings from the same sensor mean that the subject is in the same room (inactivity time).
    - Two consecutive readings from different sensors mean that the subject is moving from one room to another.
  - The frequency of movement firings within the same room (activity time).
- No sensors' firings at all can be considered a feature where it means that the subject is not available.
- Temperature, luminosity and utilities' meter readings can have their own features or can be used in other stages to mitigate uncertainty.

The contextual data in the above list provides multiple variations of context-dependent behaviours, which presents a challenge for decision support members to analysis behavioural variation according to a specific context and activity level. The field of assisted living and ambient intelligence can take advantage of the growing technology of self-adaptability and context-awareness, which provides a dynamic decision-making and provides the software with autonomic attributes such as self-organizing and self-healing. To this aim, this article focuses on demonstration how context-oriented software development can be used for building assisted living application. However, adapting the context-dependent functionality according to the current subject living within the household will enable inferring a more accurate behavioural model. In addition, it can protect those resource constrained sensors from power depletion and thus missing important data.

## IV. COSD Vs. AOSD Experiments

This section focuses on evaluating the performance and modifiability quality attributes of context-oriented software, including the COCA-middleware and the case study implementation of the CaSDD application. AOSD [5] and COSD are the alternatives for the design and construction of self-adaptive software. Their ultimate goal is to support the adaptability and variability of software systems, and to be able to reduce development cost and effort, while improving the software modularity and complexity. This motivates this study to evaluate these technologies with respect to their ability to support software adaptability (modifiability) and the performance gain from using these technologies to implement the case study application in a mobile computing environment. This article claimed that COSD is better suited to dynamic context-driven adaptation in the mobile computing domain. To this end, an evaluation of the two major paradigms (AOSD and COSD) is required to find out which one is better suited to developing self-adaptive applications.

The assumption made by the AOSD communities is that dynamic aspect weaving can be used to adjust the software behaviour dynamically, regardless of the complexity involved in implementing Aspect-Oriented Programming (AOP) applications. Existing Dynamic AOP techniques tend to add a substantial overhead in both execution time and code size [18]. The CaSDD implementation was re-engineered to be integrated with the Objective-C AOP framework [19]. As a result, several aspects were implemented which implement context monitoring and detecting. In addition, the context-dependent behaviours for the location service, activity level, and the frequency of moments were implemented. However, for the location service, there are three nested aspects implemented to provide behavioural variation of the battery level. These aspects are the GPS-based, WiFi-based, and IP-based location services. In COSD, these aspects are implemented using three COCA-components, as demonstrated in Section III-A.

### A. Experiment 1: Context Monitoring and Detection

For the context detection process, both implementations were evaluated based on the above criteria. The evaluation results for energy usage are shown in Figure 5. The evaluation results show that DAOP-CaSDD consumes more energy to notify the application components about multiple context changes which were detected in short frequency. This requires more CPU activity to process the context changes and evaluate them with the passive context values stored in the joinpoints. The CPU activity for both applications is demonstrated in Figure 6. In addition, the Dynamic Aspect Oriented Programming (DAOP) application requires more memory for allocating the aspect contexts and notifying them because each aspect must be allocated and executed. The AOP framework then notifies the aspects about the context changes. Later, the decision is

left to the aspect methods implementation to decide whether to adapt or not. Such implementation of the context detection process using DAOP intensively consumes the allocated resources to notify multiple aspects about multiple events. In some cases, the aspect implementation was independent of the execution context, but it was executed and notified.

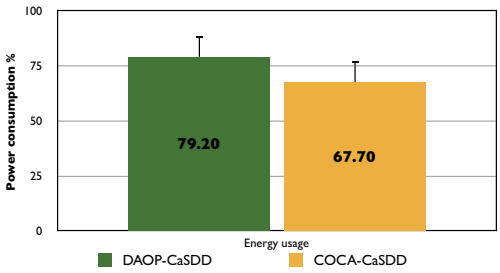


Fig. 5: Context Monitoring and Detection Battery Usage

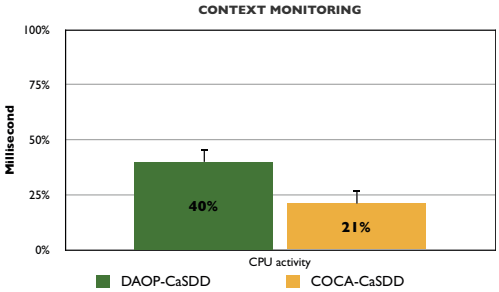


Fig. 6: Context Monitoring and Detection CPU Activity

### B. *Experiment 2: Collaborated Aspect Activation*

It is claimed that in AOSD, dynamic aspect weaving can inject tangle-free code in the program execution; as explained before, context-dependent behaviours are collaborated aspects entangled with each other. It is claimed that in COSD, COCA-components can be activated dynamically to adjust the application behaviour, with affordable costs, during the adaptation. Designing context-dependent behaviour using an aspect-oriented programming paradigm requires platform support for activating aspects driven by the context state; such an implementation requires the AOP platform to evaluate each joinpoint in conjunction with the associated context state and the passive context values. In addition, once the decision has been made, the AOP platform must search for the associated method implementation which implements the required context-dependent behaviour. Moreover, from our own experience, it is very complex to decide which aspect should be woven first, because of the implicit dependence among the aspect implementations. For example, the platform should decide when the battery level is low, and which aspects must be activated. On the other hand, when activating the location aspect, the platform must consider the battery level before deciding the frequency of sampling for 30 minutes; such processes provide cyclic dependence among the aspects implementations and lead to unguaranteed adaptation outputs.

Figure 7 shows the battery usage when multiple contextual aspects are activated and executed compared with the composition of multiple COCA-components. The figure shows that the DAOP-CaSDD consumes more energy to perform the adaptation as it requires more energy to process the context state in each joinpoint. In addition, it requires the AOP framework to resolve the dependence between several aspects before and after the advice methods execution. The CPU activity is shown in Figure 8.

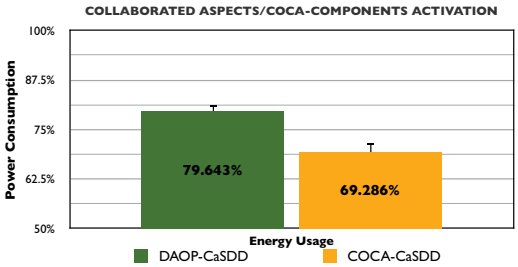


Fig. 7: Activating Collaborated Aspects/COCA-components Battery Usage

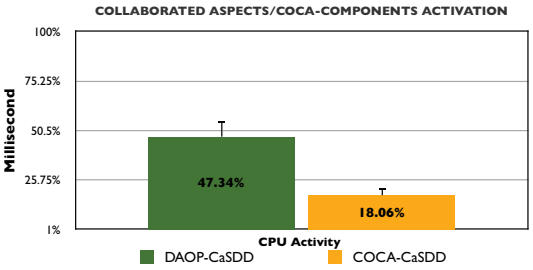


Fig. 8: Activating Collaborated Aspects/COCA-components CPU Activity

The aspects composition needs to keep track of past context conditions and their associated states; more CPU activity and memory allocation are needed to perform this functionality. This experiment describes how each platform responds to multiple events detected at the same time. The adaptation/reconfiguration time for composing aspects/components is shown in Figure 9. The values were taken every 2 minutes from the Instruments tool while executing the application for 30 min continuously. As shown in Figure 9, the COCA-CaSDD requires less CPU time for composing the components, but DAOP requires more time for activating and executing the contextual aspects. The evaluation of aspects activation and execution shows an increased adaptation time because each aspect requires more memory allocation and CPU time to resolve the execution context with the context snapshot. On the other hand, the COCA-middleware requires more adaptation time for loading and executing the bundle implementation, but it can switch between weak/strong adaptation actions

based on the execution context and the allocated resources. As shown in the figure, COCA-components composition requires less adaptation/reconfiguration, based on the adaptation mechanism. Such variations in the adaptation time provided by COCA-middleware can make use of the adaptation process and increase the device durability. The adaptation time in DAOP, as shown in the figure, may increase over the execution time, which leads to poor performance and lower efficiency.

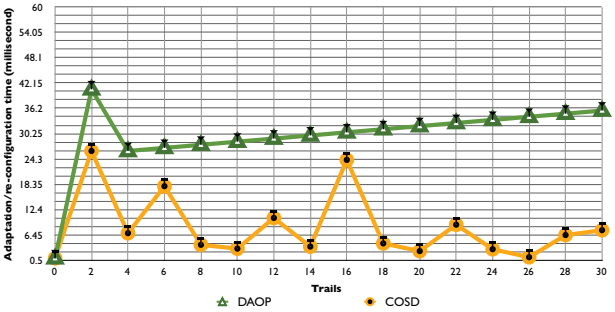


Fig. 9: Aspects/COCA-components Composition

## V. Conclusions and Future Works

The evaluation of the COSD paradigm in comparison to AOSD shows that COSD is better suited to implementing context-dependent and self-adaptive applications. The performance and energy usage in COCA-applications are better than in DAOP-applications. There is no doubt that Aspect-oriented frameworks can be used for developing and implementing self-adaptive applications, but their performance is very poor in comparison to that of COSD, as demonstrated in the implementation of the case study of Computational and Sensory Detection of Dementia application.

The COCA-MDA needs to be improved with respect to support for both requirement reflection and modelling requirements as runtime entities. The requirement reflection mechanism requires support at the modelling level and at the architecture level. Reflection can be used to anticipate the evolution of both functional and non-functional requirements. The decision policies require more development with respect to policy mismatch and resolution. This is in line with an improvement in terms of self-assurance and dynamic evaluation of the adaptation output.

## References


[1] P. Oreizy, M. Gorlick, R. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf, "An architecture-based approach to self-adaptive software," *Intelligent Systems and Their Applications*, vol. 14, no. 3, pp. 54–62, 1999.

[2] P. Inverardi and M. Tivoli, "The future of software: Adaptation and dependability," in *Software Engineering*, A. Lucia and F. Ferrucci, Eds., 2009, pp. 1–31.

[3] W. Harrison, "Modularity for the changing meaning of changing," in *Proceedings of the tenth international conference on Aspect-oriented software development*, ser. (AOSD '11), Porto de Galinhas, Brazil, 2011, pp. 301–312.

[4] B. Magableh and S. Barrett, "Context oriented software development [special issue]," *Journal of Emerging Technologies in Web Intelligence (JETWI)*, vol. 3, no. 4, pp. 206–216, June 2011.

[5] R. E. Filman, T. Elrad, S. Clarke, and M. Akşit, Eds., *Aspect-Oriented Software Development*. Addison-Wesley, 2004.

[6] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, pp. 14:1–14:42, May 2009.

[7] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "Composing adaptive software," *Journal of Computer*, vol. 37, pp. 56–64, July 2004.

[8] R. Hirschfeld, P. Costanza, and O. Nierstrasz, "Context-oriented programming," *Journal of Object Technology*, vol. 7, no. 3, pp. 125–151, March 2008.

[9] G. Kapitsaki, G. Prezerakos, N. Tselikas, and I. Venieris, "Context-aware service engineering: A survey," *Journal of Systems and Software*, vol. 82, no. 8, pp. 1285–1297, 2009.

[10] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *Proceedings of the European Conference of Object-Oriented Programming, (ECOOP '01)*, ser. LNCS, Budapest, Hungary, 1997, vol. 1241, pp. 220–242.

[11] M. Mezini and K. Ostermann, "Variability management with feature-oriented programming and aspects," in *Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. (SIGSOFT '04), Newport Beach, CA, USA, 2004, pp. 127–136.

[12] É. Tanter, K. Gybels, M. Denker, and A. Bergel, "Context-aware aspects," in *Proceedings of the 5th International Symposium on Software Composition*, ser. (SC 2006), Vienna, Autriche, 2006, pp. 227–242.

[13] É. Tanter, "Aspects of composition in the reflex aop kernel," in *Proceedings of the 5th International Symposium on Software Composition*, ser. (SC 2006), Vienna, Autriche, 2006, pp. 99–114.

[14] B. Magableh and S. Barrett, "Adaptive context oriented component-based application middleware (coca-middleware)," in *Proceedings of the 8th International Conference of Ubiquitous Intelligence and Computing, (UIC 2011)*, ser. Lecture Notes in Computer Science, vol. 6905, Banff, Canada, September 2011, pp. 137–151.

[15] E. Buck and D. Yacktman, *Cocoa design patterns*, 2nd ed. Developer's Library, 2010.

[16] M. D. Mulvenna, W. Carswell, P. J. McCullagh, J. C. Augusto, H. Zheng, W. P. Jeffers, H. Wang, and S. Martin, "Visualization of data for ambient assisted living services," *IEEE Communications Magazine*, vol. 49, no. 1, pp. 110–117, 2011.

[17] N. AlBeiruti and K. Al-Begain, "A survey on home-based technologies for detecting behavioural abnormalities and cognitive decline in elderly people," in *Proceedings of IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies,(AEECT '11)*, 2011, pp. 366–369.

[18] C. Hundt, D. Stöhr, and S. Glesner, "Optimizing aspect-oriented mechanisms for embedded applications," in *Proceedings of the 48th international conference on Objects, models, components, patterns*, ser. (TOOLS'10), Malaga, Spain, 2010, pp. 137–153.

[19] "Aspect oriented programming framework for cocoa and objective-c." http://www.cocoadev.com/index.pl?AspectCocoa, May 2011, "[Online; accessed 1-June-2011]".