

# Towards Aspect-Oriented Programming for Context-Aware Systems: A Comparative Study

<sup>1</sup>Francisco Dantas, <sup>2</sup>Thaís Batista, <sup>3</sup>Nélio Cacho, <sup>3</sup>Alessandro Garcia

<sup>1</sup>State University of Rio Grande do Norte, <sup>2</sup>Federal University of Rio Grande do Norte

<sup>3</sup>Lancaster University

[franciscodantas@uern.br](mailto:franciscodantas@uern.br), [thais@dimap.ufrn.br](mailto:thais@dimap.ufrn.br), [n.cacho, a.garcia}@lancaster.ac.uk](mailto:{n.cacho, a.garcia}@lancaster.ac.uk)

## Abstract

*Development of modular context-aware applications has been a deep challenge to software engineers. One of the main reasons is the crosscutting nature of certain context-awareness concerns. Specific distributed aspect-oriented programming (AOP) techniques have recently emerged as a promising candidate to address these shortcomings. This paper reports our ongoing effort on the definition of relevant criteria to perform a comparative analysis of five emerging AOP approaches for context-aware systems. We evaluate to what extent their specialized linguistic mechanisms scale to distributed systems, in particular context-aware mobile systems.*

## 1. Introduction

Context-aware systems are mainly characterized by sensing the environment and adapting its behaviour according to some contextual conditions. However, such context-awareness concerns are usually implemented by using traditional OO-based mechanisms, which hardwire sensor's drivers and spread *if* statements out all over the application to achieve context-dependent behaviour [1]. As context acquisition and adaptation concerns are very often distributed, the utilization of usual programming techniques leads to a complex distributed design where high reusability and maintainability are impracticable.

AOP has recently been proposed as a means to modularize several parts of a context-aware application [11], such as context acquisition, location/proximity context and context-dependent behaviour. This approach allows creating aspects capable to perform context fusion/fission and adapting modelled context into non-modelled contextual information that may be required by application to perform the context-dependent behaviour adaptation. To support composition of concerns in distributed environments, many AO distributed strategies have emerged, such as EJB [15], JBoss AOP [6] and Spring AOP [14].

However, most of the solutions are limited by the framework's capabilities as the lack of explicit distributed constructors. Besides distributed middleware, there are frameworks and specific languages that address distributed AOP. Nevertheless there is no work that compares them in the light of some important criteria for context-aware applications, such as architectural organization, transparency, synchronization and exception handling issues.

This paper reports initial results of a systematic comparison of distributed AOP techniques for developing modular context-aware applications. We described (Section 2) 5 emerging distributed AOP proposals: AWED [7], JAC [10], ReflexD [12], DyMAC [8] and CSLogicAJ [13]. The comparison of these approaches is based on five important criteria for context-aware applications (Section 3). Naturally some of the criteria are relevant to distributed systems in general, while others are specific to context-awareness support (Section 4). The conclusions (Section 5) of this exploratory comparative analysis are a first step stone on fostering the evolution of distribution-specific pointcut languages and aspect-oriented middleware systems towards the support for modular context-aware applications.

## 2. Outline of the Investigated Approaches

**AWED** [7] is an AOP language for distributed aspects with syntax based on AspectJ. The main proposal is to provide mechanisms for distribution as part of the aspect language. According to [7], the AWED language has three main characteristics: (i) it offers remote pointcuts that can match events on remote hosts, including support for remote sequences; (ii) it allows the distribution of advice execution and (iii) it provides a notion of distributed aspects including models for the deployment, instantiation and state sharing of aspects. The implementation of AWED is built on top of JasCo [5].

**JAC** (Java Aspect Components)[10] is a dynamic AOP-framework in Java. An aspect program in JAC is a set of aspect objects that can be dynamically deployed and undeployed on top of running application objects. Besides joinpoints and pointcuts core concepts, JAC has a concept named *wrapping methods* that is a code that runs on joinpoints upon meeting certain conditions specified in the pointcuts. An aspect-oriented program in JAC is composed of four main parts [10]: the base program, the aspect program, the weaver and the composition aspect.

**ReflexD** [16] is a platform for distributed AOP in Java. It was developed on top of the Reflex AOP Kernel [12]. ReflexD defines three different concepts in order to support the definition of an aspect's anatomy [16]: cut, action and binding. The *cut* of an aspect refers to the execution points of a program where such an aspect applies. The effect of aspect is the action and the binding specifies issues between a cut and an action when the latter is executed (*before, after or around*).

**DyMAC** [8] is a component and an aspect-based middleware framework that use aspect-oriented composition to connect the application logic to the middleware services. It deals with some three important challenges: *remote pointcuts, remote advice*, which can be executed in a transparent way, and *semantic components* since an aspect must be part of a mature middleware.

**CSLogicAJ** (Context-aware Service-oriented LogicAJ) [13] is a service aspect language that provides context-sensitive aspects for Ditrios [3]. CSLogicAJ is an extension of the LogicAJ language and thus support the advice concept, the call pointcut and logic meta-variables.

### 3. Criteria for comparison

In a way or another, the approaches previously presented can deal with the aspectization of context-aware applications. However, some important differences among them were detected and became criteria for comparison. The five criteria chosen are as follows.

1. *Synchronization issues* are about the correct management of multiples execution process, which accesses the same resource.
2. *Transparency* is concerned with the capacity to hide tasks and resources, which in a context-aware system are physically distributed in many devices. In this paper, transparency is about the way that the distributed code is written.
3. *Joinpoint models* define, through regular expressions, the execution points that the program can

be intercepted and how context information can be supported.

4. *Exception handling* introduces new functionalities to detect context-sensitive exceptional conditions and change the exceptional control flow according to contextual information.

5. *Implementation Availability* presents which approaches have implementation available.

A number of criteria could be defined to compare distributed AOP approaches for implementing context-aware systems. However, this is a work in progress and we are aware the above criteria are sufficiently broad to support an initial interesting comparison.

### 4. Evaluation

Table 1 summarizes the evaluation per criteria (Section 3) of the strategies presented in this work (Section 2) in a numeric classification. Each criterion can be classified as -1 (lack of support), 0 (basic support) and 1 (full support). The total score of each approach is totalized on the last column. On the criterion 5 the numbers 1 and -1 mean, respectively, the availability and unavailability of an implementation.

**Table 1.** Summary of Criteria

Approaches	Criteria					$\Sigma$
	1	2	3	4	5	
<b>AWED</b>	1	0	0	-1	1	<b>1</b>
<b>JAC</b>	0	0	0	-1	1	<b>0</b>
<b>ReflexD</b>	0	0	0	-1	1	<b>0</b>
<b>DyMAC</b>	0	1	0	-1	1	<b>1</b>
<b>CSAspectAJ</b>	1	0	1	0	1	<b>3</b>

**Synchronization issues:** The order in which the events occur in different process is critical in many context-aware applications. The lack of synchronization can compromise the functionalities of the system. The concurrency control in a distributed system is directly associated with a synchronization mechanism. Otherwise, certain accesses to the system can lead to data inconsistencies. On the other hand, asynchronous communication support is essential to pervasive mobile applications due to connectivity issues. In mobile applications, where clients are constantly connecting and disconnecting to different hosts, asynchrony support is essential to manage the client flow and the advice executions.

In JAC and ReflexD, advice execution is synchronous. However, AWED and CSLogicAJ support synchronous and asynchronous advice execution. In the DyMAC documentation [7] there is

no mention to synchronous or asynchronous support.

**Transparency of the code:** In a context-aware system, the transparency must be guaranteed.

JAC and AWED represent hosts as plain strings. AWED supports explicit constructs in its pointcut language to specify on which host an advice should be executed. Also, execution in groups of hosts is possible. However, [8] states that the AWED does not support the transparency of advice execution due to its constructor `on()`, which provide a mechanism to define where the advice should be executed. We do not agree with this affirmation because AWED allows the use of wildcards to avoid the explicit denomination of the places where the advice must be executed.

When a system offers a good level of transparency it allows that, in an implicit and automatic way, all the aspects are deployed on all hosts. Even when a new host arises, the aspect is added automatically. This feature appear tightly in both JAC, AWED and CSLogicAJ. On the latter the syntax is a mix of Java and logical notation. It requires that programmers need to know logic programming concepts. However, the underlying proxy architecture provides a transparent rebinding and recomposition of services.

**Joinpoint model and pointcut designators:** In a context-aware scenario, the joinpoint model is the basic element to support the capture of contextual information. Joinpoint models defined the points in the program executed that can be associated with advice. Pointcut designators are defined on the top of the underlying joinpoint model.

Considering the CSLogicAJ, all the services can be queried by the pointcut `service(?Service)` and the pointcut `service-attr()` resolve the attributes of the service. The pointcut `service-attr()` gives the developer access to the properties associated with the called service.

The joinpoint model of AWED provides a mechanism where, upon the occurrence of a joinpoint, pointcuts are evaluated on all hosts where the aspects are deployed. Pointcuts allow triggering aspects based on joinpoints of remote hosts. In this case, pointcuts are evaluated in all hosts where the corresponding aspects are deployed and advice may be executed on other hosts using the `on` pointcut constructor both in an asynchronous and synchronous way.

In DyMAC remote pointcuts can be evaluated over the contextual properties considering both the components and the location [8]. Transparent remote advice with remote semantics is supported. The semantic feature allows capturing remote behaviours that are associated with calling and executing remote

invocations.

Pointcut expressions in JAC are based on regular expressions but can include specific keywords or constructors and can be specified through three sub-expressions (*class*, *object* and *methods*) besides a wrapping method. Logical operators can be used.

Considering the control flow of distributed applications, all the approaches offer a distributed control flow mechanism that aims at identifying and capturing certain “patterns” during the communications among the hosts. ReflexD, DyMAC and AWED implement control flow in the same way. In addition, AWED supports distributed sequences of events for stateful aspects [7]. However, because the AWED implementation does not have the same challenging aspects of distributed time, when matching event sequence, inconsistencies can occur.

The *before*, *after* or *around* advice can be used in the CSLogicAJ language. Its pointcut language, based on a logic language, supports remote evaluation offering mechanisms to select service level joinpoints. Besides, importing and querying context allow the use of context information in an aspect. The aspect imports the namespace of the context defines remote pointcuts. Furthermore, a service composition mechanism allow the replacement of services on the architectural level.

**Exception handling:** One of the general motivations for employing exception handling in the development of resilient applications is to improve software modularity in the presence of erroneous system states [9]. The code devoted to exception detection and handling should not be intermingled with the implementation of normal system activities. In this sense, AOP can be used [4] to separate the normal system activity from the handling code by using *around* and *after throwing* advice in AspectJ. However, in context-aware applications the situation is potentially more complicated since exception handling mechanisms should properly deal with errors in the presence of contextual information changes. In this way, exception handling mechanisms should take into account contextual information to (i) detect exceptional contexts, (ii) to define protected regions, (iii) to propagate exceptions, and (iv) to select handlers [2].

CSLogicAJ is the unique approach which provides a minimal support to the characteristics described above. Exceptional context can be detected by using asynchronous advice which take into account contextual information to detect context changes. However, such advice do not support transparent exception propagation. This implies that the application should provide its own mechanism to

define protected regions, to propagate the exception and finally to select the handlers according to contextual conditions.

## 5. Summary and Conclusions

This paper presents an initial comparative analysis that is useful for: (i) developers interested in tailoring and/or extending existing distributed AOP mechanisms for context-aware mobile applications, and (ii) programmers choose which platform is better to the implementation of a specific context-aware application or crosscutting functionality. For example, the cache problem [7], is a functionality that involves a huge number of distributed processes. However, some languages as AWED have some concepts that make the implementation code simpler to understand. On the other hand, if modularization requirements for such concerns are not essential, the price paid for working with this language is potentially high.

The AWED joinpoint model uses a proceed statement to execute around advice in a distributed scenario. Since this statement has local semantics, they do not work as a completely distributed advice. Besides this, the transparency of AWED application is questioned [8] since the use of the construct *on()* on the remote advice execution ask for the explicit host name as a parameter. The JAC joinpoint model does not provide remote pointcuts and has a complicated network processing. Unfortunately, most implementations are in progress and some implementation-specific questions are open to be answered in a future paper.

## 6. References

- [1] A. K. Dey, D. Salber, and G. D. Abowd. "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications". Human-Computer Interaction, 16. 2001.
- [2] Damasceno, K., Cacho, N., Garcia, A., Romanovsky, A., and Lucena, C. "Context-aware exception handling in mobile agent systems: the MoCA case". In Proc. SELMAS'06, ACM Press, Shanghai, China, May, 2006), pp. 37-44.
- [3] Ditrios framework. [online]<http://www.ditrios.org/>
- [4] F. Filho, N. Cacho, E. Figueiredo, A. Garcia, C. Rubira. "Exceptions and Aspects: The Devil is in the Details". In Proc. of the FSE.06, 2006.
- [5] JAsCo website. [online] <http://ssel.vub.ac.be/jasco/>.
- [6] JBoss AOP. [online] <http://jboss.com/products/aop>
- [7] L.D.Benavides Navarro, M.Sudholt, W. Vanderperren, B. Fraire, and D. Suvee. "Explicitly distributed AOP using AWED". In Proc. of the 5th International Conference on AOSD'06, ACM Press, Germany, 2006, pp. 51-62,
- [8] Largaïsse, B and Joosen, W. "True and Transparent Distributed Composition of Aspect-Components". Middleware, 2006. pp. 42-61.
- [9] Parnas, D., Würges, H. "Response to Undesired Events in Software Systems". Proc. of the 2nd International Conference on Software Engineering.IEEE-CS, USA, 1976, pp. 437 - 446.
- [10] R.Pawlak, L.Seinturier, L.Duchien, G.Florin, F.Legond-Aubry, and L.Martelli. "JAC: an aspect-oriented distributed dynamic framework". Software Practice and Experience, 2004, pp. 1119-1148.
- [11] Rashid, A., Kortuem, G., "Adaptation as an aspect in pervasive computing". In Proc. of OOPSLA 2004, Vancouver, Canada, 2004.
- [12] ReflexD website. [online] <http://reflex.dcc.uchile.cl>
- [13] Rho, T., Schmatz, M. and Cremers, A. B. "Towards Context-Sensitive Service Aspects". Workshop on Object Technology for Ambient Intelligence and Pervasive Computing, in conjunction ECOOP 06, Nantes, France, July 3-7
- [14] Spring AOP . [online] <http://www.springframework.org>
- [15] Sun Microsystems. [online] <http://java.sun.com/>
- [16] Tander, E and Noyé, J. "A Versatile Kernel for Multi-Language AOP". In Proc. of the 4th ACM SIGPLAN/SIGSOFT, Springer-Verlag, Tallinn, Estonia, 2005, pp 173-188.