ISBN: 972-8924-09-7 © 2006 IADIS

MULTI-PARADIGMATIC DEVELOPMENT OF CONTEXT-AWARE APPLICATIONS

Giovanni Degli Antoni
Universita degli Studi di Milano, Dipartimento di Scienze dell'Informazione
Via Comelico 39/41, 20135, Milano, MI, Italy

Hooman Tahayori
Universita degli Studi di Milano, Dipartimento di Scienze dell'Informazione
Via Comelico 39/41, 20135, Milano, MI, Italy

ABSTRACT

In this article we are to explore different roles that Context-Oriented Programming (COP), Layered-Base Context Aware Programming (LBCAP) and Run-Time Subject Oriented Programming (RTSOP) can play in developing context-aware applications. Since the process of developing a context aware application is a very complex procedure, we have tried to separate complexities regarding different aspects of context-aware programs and based on this separation, have investigated the roles of the paradigms.

KEYWORDS

Context-Awareness, COP, LBCAP, RTSOP.

1. INTRODUCTION

Respecting the current advancements in the field of hardware, that has led to the faster and of course cheaper computers, and software engineering that has enabled application developers to develop more complex and capable software systems, yet some important factors like simplicity, portability, adoptability and productivity are not developed as expected.

It is argued that the reason lies in the nature of the structure of the current applications. In fact most of the current applications would be considered as a black box that process their explicit inputs and produce explicit outputs. An application, functions without considering its surrounding. However, there exists some systems that can, more or less, be matched to their environment but the request of change must be explicitly provided by the user, i.e. an explicit input is required for the system to be informed about necessary changes.

Some subfields in computer science have tried to alleviate the problems. For example artificial intelligence, human-computer-interaction, neural networks and fuzzy logic are some fields that have tried to make computers more adaptable, flexible and simpler to use. All the fields have common aim that is getting or at least facilitating more use of the surrounding environment.

A proposed high level solution for the above problem is to develop applications that besides accepting explicit inputs accept implicit inputs and also in added to providing explicit outputs, provide implicit outputs. Those implicit input and output would be regarded as the environment of the system or simply the context. In other words, the future applications, in order to be more adaptable, flexible, portable and productive must be able to consider their surrounding and contextual information and surely affect the environment too.

Developing such applications, regarding the necessity to the use of implicit input and providing implicit output is not that easy. Gathering, presenting, processing and applying the information, all are challenging tasks. On the other hand affecting the environment in a proper way, implicitly, is another complex subject.

In this paper, we have discussed multi paradigmatic development of context-aware applications , in effect we have focused on some, more or less, art of the day paradigms, to explore how to get use of them to

develop context-aware applications. However we have considered mainly applying the processed information, i.e. how to change or maintain the functionality of a system in different contexts, given the required contextual information.

2. CONTEXT AWARENESS

The meaning that is mentioned for *context* in Cambridge Advanced Learner's Dictionary is “the situation within which something exists or happens, and that can help explain it”, however the definition proposed in [6] is more applicable to our field, it states "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 application themselves". By this definition contextual information can range from identity, spatial information, temporal information to physiological measurements, social situation and history of interactions and more.

Context-awareness is defined as the ability of getting use of contextual information and consequently, context-aware applications are those that can gather, present, process and apply the contextual information to adopt or maintain their behavior.

3. CONTEXT-ORIENTED PROGRAMMING (COP)

COP that firstly proposed in [8] and later we refined, as follows enables you to design an incomplete program. COP lets you design a program but leave those parts that at design time you are not sure about them empty, i.e. let you introduce some *Holes* or *Gaps* in the program that should be filled later in run time, when all the required information to complete the incomplete program became known. *Gaps* will be filled with one of the subcode sections that we call it *Tenon*. This is the responsibility of sensors to provide required information.

Gap would be simply regarded as a place holder that must be filled in rum time according to some contextual information. The defined gap must be accessible by context filler even if it is filled. Each gap must be associated with at least one *intention* and one *general context;* actuating context for gap is meaningless. In this case *gap-filler* can decide what the gap is reserved for and when it should be filled. Gaps can be nested; however this nesting would be indirectly, i.e. there cannot be any gap directly defined in another gap. Instead, any tenon that would fill the gap can contain gap(s) and this indirect nesting can continue till any desired level. By the way the system must avoid circular nesting that never ends.

Tenon –or Stub as in [8]- is a subcode provided by the application developer or any user of the program to be used for filling a gap. Tenon should be associated with at least one intention and one actuating context. In the case that the tenon itself contains gaps, general context should also be associated with it.

Context is a construct reserved to denote the required contextual information. In fact two different contexts are required, *general* and *actuating*. General context is used to be associated with gaps and is to denote what kind of contextual information will affect the gap. On the other hand actuating context, which is to be used by tenons, denotes in which environment the tenon would be active.

Intention- or Goal as in [8]- is a construct that defines what a gap is reserved for in general and what the general usage of a tenon is. In fact, intention specifies any tenon’s usage and a gap’s role and the functionality of gap filler is primarily based on it. By intention, tenons would be classified in different categories; this will restrict the gap filler domain in any given context.

The most challenging part of COP is gap filler. The inherent complexity of gap filler lies in the fact that it has to deal with different kinds of information and knowledge. It has to handle the information gathered by sensors and make them usable. On the other hand it has to deal with the context and intention constructs. Comparisons between gaps and tenons – i.e. their intentions and contexts respectively- would be considered a kind of knowledge handling. All the aforementioned activities must be done lively, in run time. However it is the responsibility of gap filler to bind all the parameters in tenon with the skeleton.

4. SUBJECT ORIENTED PRORAMMING

Subject oriented programming is a post object oriented methodology and is to enhance the object orientation toward enabling simpler implementing of integrated suits. Subject would be defined as a collection of classes or even a fragment of a class that reflects a perception of part of a world in a specific view, or more precisely, as mentioned in [7] subject is "a collection of state and behavior specifications reflecting a particular gestalt, a perception of the world at large, such as is seen by a particular application or tool".

Subject Oriented Programming (SOP) enables developing independent applications separately and then composing them. It means that any application that is to be composed with others would not need to be aware of their internal structure. It is possible to compose multiple subjects to yield a complete suite of applications, this implies combining class definitions in each subject such that the needs of each subject would be satisfied without any intervention in other subjects' needs and functionality but some restrictions or expansions in the views due to the collaboration of different applications is unavoidable. So extending an existing application, by developing the required extensions and then composing it with the main application is feasible with SOP. It is important to say that each application should respect SOP laws to be composable with others.

As with COP, the most challenging part of the SOP is the composition process, this would be regarded as the core process in SOP. For more details on the composition process and required rules and structures refer to [10].

Composition usually takes place before generating binary code. Usually, COP compilers, transform each subject that is to be composed with others, to an intermediate form, that composer can handle. Composer deals with these intermediate forms to produce the final intermediate code of the application suite, then the result would be translated to machine language. However, as mentioned in [9] dynamic composition is feasible, i.e. subject composition takes place in run time. We call it Run-Time Subject Oriented Programming (RTSOP). It means that those applications developed with SOP would be expanded or condensed in run time due to the needs.

5. LAYERED-BASE CONTEXT AWARE PROGRAMMING (LBCAP)

The idea is presented in [5], however similar approaches would be found, for example, in [1], [2], [11] and [4]. The main idea is that some layers that contain subcodes, are associated with partially defined classes, such that in the control flow of the running program, the layers tagged with some contextual information, would be activated or deactivated. Activating a layer in effect causes its code to be added to the class and vice versa.

6. DISCUSSION

Context-aware behavior of systems would be seen in two sides, one user side and the other, system side. On user side which is mainly related to adaptability, flexibility and productivity, the system's reaction toward context, would be divided in three categories:

1- *Presenting related information or offering related services to the user according to the context change*. For example, showing information related to the object the user is closed to while he is in a museum or changing the user interface according to the time of the day. This can be considered as adaptability and productivity, by the way, theoretically, to be applied should not impose a huge change in the program. It implies that the current system should change or modify its behavior which can be done by either changing the systems' objects' methods or adding some new methods to some objects or removing some methods and fields or even modifying them. It is clear that LBCAP is the best choice for such changes since it allows the developer to devise some layers over any object. By activating each layer either a method is modified-if previously exists- or is added to the object and removed as the layer becomes inactive. According to the nature of LBCAP it is possible even to devise some layers that have overlapping and resolve the inconsistencies. But in LBCAP developer has to forecast all the possible cases and design specific layers for them. This can be considered as a negative point in using LBCAP. It seems that it is difficult to make LBCAP, float, i.e. some layers would be defined later even by other persons and later in run time be attached

to objects. It is due to the nature of LBCAP that layer developer must be aware of the existing layers and classes and associate his extensions to the proper layers. By the way in effect, LBCAP provides clear mechanisms for programmers to locate and accurately manipulate those parts of the source code to enable the programs to adapt to the new context. But if the runtime modification for any layer is essential, LBCAP is not a suitable paradigm.

2- *Automatic execution of tasks*. For example, running media player while, entering a Bar. There exists two scenarios for this part, either the task is a part of the application and must be executed, which is regular, or the task must be composed with the application and then be executed. In the second scenario, the applicable paradigm is RTSOP, since the goal of RTSOP is to overcome the tyranny of centralized development of application and providing run-time composition. By the way, the overhead of applying this paradigm is considerable, because each running context-aware application designed in RTSOP must be accompanied with its intermediate code and while the application is running, the run-time composer must be available. If a context change forces a subject to be added to the application, run-time composer should take place, compose the intermediate form of the applications with the intermediate form of the subject and provide a new intermediate form, then the compiler should generate the new binary code. On the other hand if the context change, forces a subject to be removed from the application, run-time decomposer should take place, and using the bookmarks in the application's intermediate code, pull out the codes related to the subject and then the compiler, should produce a new binary code.

These scenarios, besides levying considerable overhead to the system, arise some crucial problems, like terminating a running program, and running a new version of that, but usually not from the beginning but from a logical point regarding the current context and with respect to the point where the former execution had stopped. The problem of recompiling and re-running of the application would be alleviated by using an interpreter for the intermediate code, but the problem of where the interpreter should start the new execution remains, and yet the overhead of the composition remains and overheads of the interpreter arise.

It is worth mentioning that each subject, itself would be developed using LBCAP and COP- we will describe why COP may be required, while exploring the system side, in the following.

3- *Automatically attaching current context to some information for future uses*. For example, while you are driving and falling into a hole, the locational information should be attached to the fact of existing a hole to be used to inform the driver of the hole in future in the same context. This part, despite its illusive simplicity, is complex. The simple part is feasible by providing and adding an extension –a layer- containing the information, tagged with the contextual information, but the hard part is finding to which object and even in which subject it should be attached and how to be used in future. This part requires techniques of knowledge representation and processing. Also some tags should be attached to each subject and object to denote their usages, this will guides where to attach the extensions.

Partially, part 3 would be fulfilled with LBCAP, but some more changes to LBCAP are needed. Each subject and object should be tagged such that its role become evident, this enables the extensions to find the right object(s) to match with. In between a process for locating the extensions is required.

On the system side, we can expect context aware system to preserve its maximum functionality with respect to the changes in the platform. Usually required changes are limited to a single or some consecutive commands. However, we should expect considerable amount of such consecutive commands scattered in all source program. According to the discussions in the former sections, COP would be the best choice for the case. This is due to the metrics mentioned in [13] that categorized paradigms into two categories, large scale and small scale. According to the metrics mentioned, COP is in the small scale's. So we can, at the best, expect COP to be suitable for manipulating a limited number of consecutive commands which induce an intention.

Here, we implicitly mentioned the fact that a context change may cause changes in different part of the program. [12] has tried to deal with the fact as cross-cutting aspect. There, authors have proposed a framework on context-aware aspect. But according to the above discussion it is clearly evident that context-aware aspect is mostly applicable on the system side, i.e. while we are to deal with COP and rarely on the user side, because on user side contextual changes, impose the system to be expanded or condensed.

In summary we can claim that for applying minor changes to a program that is to be context-aware, COP paradigm is recommended –especially on system side and rarely on user side-, for medium changes, LBCAP is preferable and for more major changes, RTSOP is more applicable.

7. CONCLUSION

As mentioned in [3] almost all of the existing context-aware applications are developed by highly skilled people and used in research laboratories. It implicitly means that, yet not any comprehensive framework and powerful tool to support context-aware programming exists. This backs to the nature of such systems that demands for some complex activities, like gathering, presenting, processing and applying contextual information. We mainly focused on using different art of the day methodologies to develop those parts of the context-aware applications that are to apply the contextual information.

It is clearly seen that it is not possible to develop a real context-aware application relying on just one paradigm. On the other hand it is difficult to draw exact boundary where to use different methodologies and paradigms, which is due to the complexity of developing context-aware applications, by the way, as we discussed in this paper, it is possible to recommend where each paradigm is more applicable.

REFERENCES

[1] Batory D.,1999, Product-Line architectures, generators and reuse, *Proceeding of GCSE'99*, Efurt, Germany.

[2] Batory D., Geraci B. J., 1997, Composition validation and subjectivity in GenVoca generators, *IEEE Trans. On software engineering (Special issue on Software Reuse),* pages 67-82.

[3] Brown, P.J.et al, 1997, Context-Aware Applications: from the Laboratory to the Marketplace, IEEE Personal Communications, 4(5), pp. 58-64.

[4] Bobrow D., Goldstein I.., 1980, Representing Design Alternatives. *Proceedings of the Conference on Artificial Intelligence and the Simulation of Behavior*, Amsterdam.

[5] Costanza P., Hirschfeld R., 2005, Language Constructs for Context-oriented Programming, an overview of ContextL, *In Proceedings of the ACM Dynamic Languages Symposium*.

[6] Dey, A.K. and Abowd, G.D., 1999, Toward a better understanding of context and context-awareness. *GVU Technical Report GIT-GVU-99-22*, College of Computing, Georgia Institute of Technology.

[7] Harrison W., Ossher H., 1993, Subject Oriented Programming (A Critique of Pure Objects), *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, pages 411-428.

[8] Keays R., Rekotonirainy A., 2003, Context-Oriented Prgramming, *International workshop on Data Engineering for Wireless and Mobile Access*, San Diego, USA, ACM Press, pp.9-16.

[9] Ossher H.,et al, 1994, Subject-Oriented Programming: Supporting Decentralized Development of Objects ,*Proceedings of the 7th IBM Conference on Object-Oriented Technology,* Santa Clara, CA, July 1994. IBM.

[10] Ossher H., et al., 1995, Subject-Oriented Composition Rules. *Proceedings of OOPSLA '95,* Austin, Texas, United States pp. 235 - 250

[11] Smith R., Ungar D., 1996, A Simple and Unifying Approach to Subjective Objects. *Theory and Practice of Object Systems*, Vol. 2(3), pp. 161-178.

[12] Tanter E., et al , 2005, Context-Aware Aspects, *University of Chile, Number TR/DCC-2005-12*.

[13] Vranic V., 2002, Toward Multi-Paradigm Software Development, *Journal of Computing and Information Technology* 10(2), pp. 133-147.