Model Driven Development of Context Aware Software Systems

Andrea Sindico
University of Rome “Tor Vergata”
Elettronica S.p.A.
andrea.sindico@gmail.com

Vincenzo Grassi
University of Rome “Tor Vergata”
vgrassi@info.uniroma2.it

ABSTRACT
This paper presents the first results of an ongoing work towards the realization of a model driven development framework for context awareness. Its core element consists of a domain specific modeling language called CAMEL (Context Awareness ModEling Language), and defined as a UML extension. CAMEL can be used to enrich a UML model of an application with elements related to contexts and context dependent behaviors. The resulting UML+CAMEL model is the starting point for model transformation aimed at generating executable code or other artifacts. CAMEL is implemented by an Eclipse plugin.

Categories and Subject Descriptors

General Terms
Design, Languages, Documentation.

Keywords
Context Awareness, Modeling, UML, MDA, Context Oriented Modeling, Context Oriented Programming.

1. INTRODUCTION
Context Oriented Programming [1], [2] is an emerging approach aimed at providing explicit supports for context awareness (CA) [3] in programming languages and runtime environments. Several context oriented programming approaches have been proposed in literature aimed at addressing both the issues of context sensing and context driven adaptation [4], [5]. Close to them approaches aimed at rising the level of abstraction are required that should enable the designer to take into account context awareness concerns also in the design phase. Moreover, due to technological constraints related to the languages adopted in the development of a target system (TS), existing COP approaches can lack a good separation of concerns with respect to the TS where context awareness behaviors have to be introduced. As a consequence the developer has to modify the target system code with constructs related to CA characteristics by strongly coupling the CA concern with other TS concerns. Motivated by the objective of both raising the level of abstraction where context awareness capabilities can be defined and reducing the coupling between CA and TS models, in this paper we present an ongoing work towards the realization of a model driven development framework [11], [12], enabling a designer to handle context awareness concerns at the design phase of a system. The core element of this framework consists of a domain specific modeling language (DSML)[8] called CAMEL (Context Awareness ModEling Language) which enables enriching independently defined UML models with the model of context aware behaviors. Model transformations can be then applied to the defined CAMEL+UML models, aimed at generating executable code for a specific platform (e.g., ContextJ [5], ContextToolkit [4], ContextL), or other artifacts such as metrics or documentation.

2. CAMEL (Context Awareness ModEling Language)
In this section we introduce the CAMEL language through an informal description of its meta-model. Consequently a simple example of the language expressiveness is presented. CAMEL can be considered as an heavy-weight extension of UML, that instantiates the conceptual domain model for context awareness introduced in [6]. An editor for the CAMEL language has been implemented exploiting the Eclipse Modeling Framework (EMF) [7], the Eclipse plugin defining a modeling environment and code generation facility for building tools and other applications based on a structured data model.
Each context state is related to a ContextConstraint representing the condition that has to be verified by the involved contextual information in order to consider the system in the related context state. CAMEL provides three kinds of ContextConstraint, namely StateConstraint, EventConstraint and Operator. State constraints refer to context attributes; event constraints refer to context events; the operator finally enables to compose state and event constraints by defining logical or temporal condition over them.

Figure 2 depicts the concepts of the CAMEL meta-model realizing the context adaptation concern. Context aware adaptation can be defined as the set of adaptation mechanisms that can be triggered and then activated during context monitoring activities in order to properly react to context changes. In the CAMEL modeling language contextual adaptation can be realized by means of two mechanisms: context aware bindings and context aware inserts. A binding associates values to target system’s entities depending on the retrieved contextual information. Inserts are special construct which introduce additional structural or behavioral elements (structural vs behavioral inserts) depending on the perceived context. As in the work of Costanza et al. [5] we call Adaptation Layer the construct as acting for logically related adaptation mechanisms (i.e. binding or inserts).

When an adaptation layer is activated all the adaptation mechanisms having a binding or inserts are activated too and the desired adaptation is introduced. In the CAMEL language the adapters are those entities which act as container for logically related adaptation layers. Adapters receive signals by the monitors and activate/deactivate their adaptation layers depending on the context states which are currently active.

Exploiting the Eclipse Modeling Framework we have defined an ECore description of the CAMEL meta-model. Starting from the developed ECore based meta-model we have generated an editor for the CAMEL language which is seamlessly integrated with the Eclipse UML editor as it enables to introduce models of context awareness capabilities into TS models defined by means of the UML without having to modify them. To give an example of how this can be done we take as reference the ContextJ code example introduced in [5] by Costanza et al. and depicted in Figure 3. Two classes are defined, Person and Employer, with field names address and employer, together with the necessary constructors and a default toString method.

```java
class Person {
    String name, address;
    Person(String newName, String newAddress, Employer newEmployer) {
        this.name = newName;
        this.employer = newEmployer;
    }
    String toString() {
        return proceed() + \"[\"Address\", \"Employer\"] \";
    }
}
class Employer {
    String name, address;
}

class Monitor {
    String proceed(), address;
    String toString() {
        return proceed() + \"Address\";
    }
}
```

Figure 3: ContextJ Example [5]
Within the classes code two layers of context driven adaptation behaviors are also introduced named Address and Employment. These layers define behavioral variations on the toString method. In the Address layer, address information is returned for instances of Person and Employer in addition to the default behavior of toString. The call to the special method proceed ensures that the original definition of toString is called. The toString method in layer Employment returns additional information about the employer of a person in the Person class. None of the defined layers is activated by default. Instead a client program must explicitly choose to activate them when desired. To this end ContextJ provides with and without constructs for activation and deactivation of layers with dynamic scope. The purpose of this example is to present different views of the same program where each client can decide to have access to just the name of persons, their employment status, or the addresses of persons, or employers, or both. For example, when a client chooses to activate the Address layer but not the Employment layer, address information of persons will be printed in addition to their names. When the Employment layer is activated on top, a request for displaying a person object will result in printing that person’s name, its address, its employer, and its employer’s address, in that particular order. A code fragment showing the activation of these two layers is given in Figure 4. ContextJ does not modularize the source code along the layers but keeps the object-oriented modularization along classes. That is to say that: instead of grouping partial classes definitions inside layers the layer definitions are grouped inside classes. A possible advantage of this approach is the possibility for a layer to refer private fields of the core class definition.

```java
public class Person {
    private String name;
    private String address;

    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }

    public String getName() {
        return name;
    }

    public String getAddress() {
        return address;
    }

    @Override
    public String toString() {
        return name + "\n" + address;
    }
}
```

However it has the drawback of requiring the developers to modify the code of possibly already developed components. Moreover it leads to a coupling between the target components and the ContextJ code implementing CA concerns possibly reducing the reusability of the involved components.

A possible solution to these issues is to model both the system and the desired context aware behavior using UML and CAMEL. In this way we raise the level of abstraction, and preserve the separation of concerns at the modeling level. Model driven transformation can then be applied to the obtained platform independent models (e.g., based on Open Architecture Ware workflows [9]) to produce the ContextJ code already woven with the target system code (or other alternative implementations e.g., AspectJ [10], ContextToolKit, etc.).

A class diagram representing the classes involved in this example can be modeled through the UML Eclipse editor as depicted in Figure 5. We have modified the example making the Employer class inheriting from Person and introducing a type attribute in the Person class which is a String representing the related instance’s type, namely Person or Employer.

Figure 6 depicts the CAMEL editor where a new model called CamelTest has been instantiated and the defined class diagram (the targetSystem.umlClass file) has been loaded as an external resource.

First of all we start modeling those constructs representing the contextual information we are interested in. In this example we are interested in changing the way a Tester instance perceives a Person instance depending on the testing method which has been invoked (test1 or test2). When the test1 is called the toString method of the Person object passed as parameter has to be invoked with the Address layer active. When the test2 is called also the Employment layer has to be activated. To this end we have defined an event based context called TestingContext consisting of two context events, Testing1 and Testing2, which respectively refer to the test1 and test2 methods. These two elements also pick up the reference to the Tester object having invoked the related method by means of the testerInstance alias (Figure 7).

Once the desired contextual information has been properly modeled in well defined constructs (state- and event-based context) isolated from the target system, it is necessary to model those conditions that, if verified, trigger the activation of adaptation mechanisms. Figure 8 depicts an example of monitor called TesterMonitor which is aimed at detecting when the test1 and test2 methods are invoked. It consists of two context states named addressNeeded and addressAndEmployerNeeded. The former goes active as soon as the Testing1 event occurs, that is to say as soon as the test1 method is invoked. On the contrary the latter context state is activated as soon as the Testing2 context event occurs that is to say the test2 method is invoked.
Figure 9 depicts an example of CAMEL adapter named TesterAdapter consisting of two adaptation layers named Address and AddressAndEmployment. The former is triggered by the activation of the addressNeeded context state while the latter by the activation of the addressAndEmploymentNeeded context state.

Both the layers contain a binding aimed at substituting the default implementation of the toString method of a Tester object. Figure 10 and Figure 11 depict how the binding can be modeled by means of the CAMEL editor. First of all an activity diagram of the new desired behavior has to be defined by means of the Eclipse UML editor (Figure 10). The defined model can then be loaded into the camel editor and referenced by the binding (Figure 11 value parameter) in order to substitute it as the new implementation of a method specified by means of the pointcut parameter. As any adaptation mechanisms the binding remains until the related layer is active. Once the binding is removed the affected method returns to its original implementation.

3. CONCLUSIONS AND FUTURE WORKS

In this paper we have presented CAMEL, a domain specific modeling language enabling software engineers to handle context awareness concerns at the design phase of a system. We have first introduced the CAMEL meta-model as a possible instantiation of the conceptual model for context awareness described in [6]. Then, exploiting an appositely realized Eclipse based editor for the CAMEL language, we have tried to demonstrate how CAMEL can be used to introduce context awareness capabilities within already existing and independently modeled applications. Because of the platform independency of CAMEL models, transformation
workflows can be defined aimed at automatically generating executable code or other artifacts (i.e. metrics, documentation, etc.).

What presented is a first step of an ongoing work aimed at realizing a complete MDD framework for context awareness. Further steps will consist of the realization of an enriched graphical representation for the CAMEL models; the modeling of inference rules exploitable to derive complex contextual information. CAMEL is actually tailored for the modeling of context dependent behaviors in component based system; we would also investigate about the possibility to extend it to the needs of web services or real time systems which are both typically affected by context aware requirements. To this end we are evaluating the possibility to define a CAMEL refinement integrated with the SysML [13] by the introduction of the CAMEL meta-model’s ECore implementation in the Topcased environment [14], an Eclipse based editor for SysML.

4. REFERENCES


