American Journal of Software Engineering and Applications
2012;1(1):10-22
Published online December 30, 2012 (http://www.sciencepublishinggroup.com/j/ajsea)
doi: 10.11648/j.ajsea.20120101.12

A framework for evaluating model-driven architecture

Basel Magableh[1],Butheyna Rawashdeh[2], and Stephen Barrett[3]

[1]School of Computer Science and Informatics,University College Dublin,Ireland

[2]Faculty of Computer Science and Information, Ajloun University College, AL-Balqa Applied University, Ajloun, Jordan

[3]School of Computer Science and Statistics, Trinity College, University of Dublin, Ireland

Email address:
basel.magableh@ucd.ie (B. Magableh), butheyna.rawashdeh@bau.jo (B. Rawashdeh), stephen.barrett@tcd.ie (S. Barrett)

To cite this article:
Basel Magableh,Butheyna Rawashdeh, and Stephen Barrett. A Framework for Evaluating Model-Driven Architecture. *American Journal of Software Engineering and Applications,* Vol. 1, No. 1, 2012, pp. 10-22. doi: 10.11648/j.ajsea.20120101.12

Abstract: In the last few years, Model Driven Development (MDD) has become an interesting alternative for designing the self-adaptive software systems. In general, the ultimate goal of this technology is to be able to reduce development costs and effort, while improving the modularity, flexibility, adaptability, and reliability of software systems. An analysis of model-driven methodologies shows them all to include the principle of the separation of concerns as a key factor for obtaining high-quality and self-adaptable software systems. Each methodology identifies different concerns and deals with them separately in order to specify the design of the self-adaptive applications, and, at the same time, support software with adaptability and context-awareness. This research studies the development methodologies that employ the principles of model-driven architecture in building self-adaptive software systems. To this aim, this article proposes an evaluation framework for analyzing and evaluating the features of those development approaches and their ability to support software with self-adaptability and dependability in highly dynamic contextual environment. Such evaluation framework can facilitate the software developers on selecting a development methodology that suits their software requirements and reduces the development effort of building self-adaptive software systems. This study highlights the major drawbacks of the proposed model-driven approaches in the related works, and emphasize on considering the volatile aspects of self-adaptive software in the analysis, design and implementation phases of the development methodologies. In addition, we argue that the development methodologies should leave the selection of modeling languages and modeling tools to the software developers.

Keywords: Model-Driven Architecture, Self-Adaptive Application, Context Oriented Software Development, Evaluation Framework Of Model-Driven Architecture.

1. Introduction

There is a growing demand for developing applications with aspects such as context awareness and self-adaptive behaviors. Context awareness [1] means that the system is aware of its context, which is its operational environment. Hirschfeld et al. [2] considered context to be any information that is computationally accessible and upon which behavioral variations depend. A self-adaptive application adjusts its behavior according to context conditions arising during execution. A self-adaptive application modifies its own structure and behavior in response to changes in its operating environment [3].

In recent years, a significant number of model-driven architecture approaches were proposed for the construction of context-dependent and self-adaptive applications. The Object Management Group (OMG) presented Model Driven Architecture (MDA) as a set of guidelines for building software systems based on the use of the MDD methodology [4]. MDA focuses primarily on the functionality and behavior of a distributed application or system deployed across many platforms. In MDA, the functionality and behavior are modeled once and only once. Thus, MDA defines the notions of a Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). CIM describes the software requirements in computational free fashion. A PIM describes the components of the software that do not change from one platform to another, and a PSM describes the implementation of the software components independently from the platform configurations [4].

This article contributes to the knowledge by providing an evaluation framework for most popular model-driven

approaches that were proposed to support the development of self-adaptive software systems. Such evaluation framework can facilitate software developers on selecting the best development approach that suits their needs and the software requirements.

This research studies the MDA approaches that used for building self-adaptive mobile software. For this aim, we focus on studying the features of those modeling approaches and comparing their impact on the software development. This can help the software developer in selecting the best methodology that suits their needs.

This article is structured as follows: the self-adaptive software and their self-* properties are defined in Section II. Section III provides a detailed description of model-driven approaches that were proposed for facilitating the development of self-adaptive software systems. Section IV proposes an evaluation framework for analyzing and evaluating those model-driven approaches. Section V illustrates the evaluation results, followed by the conclusions of this study.

2. Self-Adaptive Software

Mobile computing infrastructures make it possible for mobile users to run software systems in heterogeneous and resource-constrained platforms. Mobility, heterogeneity, and device limitations create a challenge for the development and deployment of mobile software. Mobility induces context changes to the computational environment and therefore, changes to the availability of resources and services. To overcome those challenges mobile software can be more dynamic and adaptive by enabling the software to be able to adapt their functionality/behavior to the context changes in the mobile environment [5]. This requires implementing mobile software to be more self-adaptive and reliable.

Self-adaptive applications have the ability to modify their own structure and behavior in response to context changes in the environment where they operate [3]. Self-adaptive software offers the users with context-dependent and context-independent functionality. Context-independent functionality (also called base functionality) refers to software functionality whose implementation is unaffected by the context changes. For example, the map view and user login forms in mobile map application are context-free functionality (i.e. context changes would not change their functionality). The context-dependent functionality refers to software functionality, which exhibits volatile behavior when the context changes. Self-adaptive software can be seen as a collaboration of individual features spanning the software modules in several places [2], and they are sufficient to qualify as heterogeneous crosscutting in the sense that different code fragments are applied to different program parts [6].

Before encapsulating crosscutting context-dependent behaviors into a software module, the developers must first identify the behaviors in the software requirements. This is difficult to achieve because, by their nature, context-dependent behaviors are entangled with other behaviors, and are likely to be included in multiple parts (scattered) of the software modules [7]. Using intuition or even domain knowledge is not necessarily sufficient for identifying their volatile behavior; instead, a formal procedure is needed for analyzing and separating their individual concerns [8].

Implementing a self-adaptive software system in a resource-poor environment faces a wide range of variance in platforms' specifications and Quality of Services (QoS) [9]. Mobile devices have different capabilities in terms of CPU, memory, and network bandwidth [9]. Everything from the devices used and resources available to network bandwidths and user context can change extremely at runtime [11].To face those challenges, the concept of MDA were employed of building self-adaptive software as an attempt to overcome the variability of context changes and reduces the development efforts, besides increasing the software ability to change its architecture and/or behaviors at runtime.

In general, using MDA for building self-adaptive softwarefaces several challenges such as maintaining the correspondence between architectural models and software implementation in order to ensure that behavioraladaptation is appropriately executed. The second issue is providing the necessary facilities for implementing the software in wide-range of platforms, specifically in mobile computing the same application can be executed in several platforms. The following section focuses on describing MDA approaches for modeling self-adaptive software systems that proposed to suit mobile computing platform.

3. Modeling Self-Adaptive Software

In the classical view of object-oriented software development, the modular structure for software systems has rested on several assumptions. These assumptions mayno longer characterize the challenge of constructing self-adaptive software systems that are to be executed in mobile computing environments [13]. 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.

Supporting the development of self-adaptive software systems raises numerous challenges. These challenges include: 1) the development processes for building them, 2) the design of their building blocks (i.e. component model,

service model or code fragments) and the adaptation engine, which describes the design patterns that can be employed by the middleware designers to support adaptability, 3) the adaptation mechanism that describes the best adaptation action that can be used under the limited resources of the computational environment. This requires the model-driven approach to maintain the correspondence between architectural models and system implementation in order to ensure the adaptation action is appropriately executed. 4) Providing the necessary configurations that suit the deployment platforms.

An appropriate way to study those challenges is to classify them on the basis of adaptation features that they support and how they manage software variability in the architecture level [12]. In the following sections, several model-driven approaches that target engineering self-adaptive software systems are discussed.

3.1. Model Driven Development and AOP

Carton et al. [8] proposed the Theme/UML, a model driven approach supported by Aspect-Oriented Programming (AOP) in an attempt to model various crosscutting concerns of context-aware applications at an early stage of the software development. Crosscutting concerns refers to aspects of a program that affect other concerns. These concerns often cannot be cleanly decomposed from the rest of the system in both the design and implementation. Theme/UML provides a systematic means to analyze the software requirements in order to identify the base and crosscutting concerns, and the relationships between them.

The Theme/UML approach was based on the use of the Meta Object Facility (MOF) extension and the ECORE [14]. The MOF meta model for the development of context-aware mobile applications proposed by de Farias et al. [15] was structured according to the core and service views of thesoftware system. This approach provides a contextual model that is independent from the application domain. However, it does not provide high-level abstraction of the software models, which express conceptual characteristics of the context-dependent behaviors. From a software developer's perspective, it does not take into account the architectural or deployment issues of the target platform;there is no clear process that describes the deployment configurations and the platform specific model of the software. In addition, it has focused on the model-to-model transformation for generating the software composition. Such approach increases the development effort, as the developers have to write and configuremany MOF scripts for the software. The Theme/UML methodology limits the development of self-adaptive applications to a very specific framework that supports the AspectJ and Eclipse Modeling Framework (EMF) [16]. Extending this paradigm for another platform requires a specific compiler that supports Aspect-Oriented Programming (AOP) and toolset that follow the process of EMF.

Plastic is another development approach, which uses the MDD paradigm for developing and deploying adaptable applications, implemented in Java language [5]. The Plastic development process focuses on the model verification and validation and service composition of java service stubs. The methodology shows a very interesting feature of runtime model verification and validation mechanism. Unfortunately, the generated software is tightly coupled with the target deployment platform and cannot be used with a standard development process supported by a standard object-oriented language other than the JAVA and AspectJ languages. However, the two-paradigm Theme/UML and Plastic face challenges with regard to the model manipulation and management. These challenges arise from problems associated with (1) defining, analyzing, and using model transformations, (2) maintaining traceability linksbetween model elements to support model evolution and round-trip engineering, (3) maintaining consistency among viewpoints, (4) tracking versions, and (5) using models during runtime [17].

3.2. A-MUSE

Daniele et al. [18] proposed an MDA-based approach for the refinement andmodeling of software systems, called Architectural Modeling for Service Enabling in freeband (A-MUSE).The A-MUSE approach focuses on the decomposition of the PIM model into three levels; each level is used forautomating a behavioral model transformation process. Daniele et al. [18] applied their approach to a Mobile System Domain Specific Language (DSL) (called M-MUSE). Therefore, the platform independent design phase has been decomposed into service specification, and platform-independent service design. The platform-independent service design model is a refinement of the service specification, which implies correctness and consistency of the originalbehavioral model. Such correctness and consistency are addressed in the transformation process. However,when trying to realize this refinement transformation, there is a gap between the service specification and the platform-independent service model, so that correctness and consistency were hard to guarantee in a single transformation process. Daniele et al. approach this problem by proposing multiple rounds of the transformation between the PIM and PSM, which requires the developers to switch simultaneously between the PIM, PSM and the service specifications several times during the software development.

3.3. CAMEL

Context Awareness modeling Language (CAMEL) is an MDD-based approach proposed by Sindico and Grassi [19]. The approach uses a domain-specific language called JCOOL, which provides a metamodel for context detection;a context model designed using the JCOOL metamodel supports CAMEL. However, Sindico and Grassi implemented the context binding as the associate

relationship between context value and context entity. On the other hand, context-driven adaptation refers to a structure or behavior elements, which are able to modify the behavior based on context values. The structural or behavioral insertion is accomplished whenever a context value changes; it uses AOP inter-type deceleration, where the behavioral insertion is accomplished by means of an AOP advice method to inject a specific code into a specific join point.

The CAMEL paradigm provides insufficient details with regard to the underlying component model or the application architecture. The authors used their former domain-specific language to support the Context-Oriented Programming (COP) approach proposed by Hirschfeld et al. [2]. Moreover, CAMEL has no formal MDD methodology that possesses a generic life cycle that a developer can use. Irrespective of these problems, JCOOL is specific to an AOP framework called the Simple Middleware Independent LayEr (SMILE) [20]. SMILE platform used for distributed mobile applications [20]. The model approach in JCOOL supports only ContextJ, which is an extension of the Java language proposed by Appeltauer et al. [21]. The CAMEL methodology requires the software to be re-engineered whenever a new context provider is introduced into the context model. The developers must build a complete context model for the new values and maintain the underlying JCOOL DSL and the Unified Modeling Language (UML) model. The CAMEL methodology has adapted AOP and the EMF to produce a context-oriented software similar to the layered approach proposed by Hirschfeld et al. [2]. This makes CAMEL limited to the EMF tool support and the ContextJ language [22]. From our point of view CAMEL tightly coupled the software with modeling language, modeling tool and the target deployment platform configurations.

3.4. MUSIC MDD

The MUSIC development methodology [23]–[25] adapts a model-driven approach to construct the application variability model. In MUSIC, applications are built using a component framework, with component types as variation points. The MUSIC middleware is used to resolve the variation points, which involves the election of a concrete component as a realization for the component type. The variability model defines the component types involved in the application's architecture and describes their different realizations. This comprises either a description of collaborating component types and rules for a composite realization, or a reference to a concrete component for an atomic realization. To allow the realization of a component type from an external services, the variability model can include a service description, which is used for service discovery and invocation in the variability model.

The software architecture in MUSIC is a pluggable architecture, which means composing the software from multiple plug-ins developed separately. To support dynamic adaptation, a middleware is proposed for facilitating a generic and reusable context management system. The architecture supports context variation and resource utilization by separating low-level platform-specific context from higher-level application-specific context. The resource utilization is improved through intelligent activation and deactivation of context-related plug-ins based on the needs of the active application. The MUSIC middleware architecture defines multiple components that interact with each other to seamlessly enable self-adaptive behavior in the deployed applications. These components include context management, adaptation reasoned, and a plug-in lifecycle management based on the Open Services Gateway initiative framework (OSGI) [26].

At runtime, a utility function is used to select the best application variant; this is the so-called 'adaptation plan'. The utility function is defined as the weighted sum of the different objectives based on user preferences and QoS. Realistically, it is impossible for the developer to predict all possible variations of the application when unanticipated conditions could arise. In addition, mobile computing devices have limited resources for evaluating many application variations at runtime and can consume significant amounts of device resources. Theoutcome of this isthe adaptation benefits aredefeated by the big loss of the allocated resources[12].

3.5. Paspallis MDD

Paspallis [27] introduced a middleware-centric development of context-aware applications with reusable components. Essentially, his work is based on the MUSIC platform [28]. According to Paspallis, an MDA-based context-aware application is built by separating the concerns of the context provider from those of the context consumer. For each context provider, a plug-in or bundle is modeled during the design phase. At runtime, a utility function is used to consider the context state and perform the decision of which plug-in should be loaded. Once the plug-in is selected to be loaded into the application, the MUSICmiddleware performs dynamic runtime loading of the plug-in.

However, it is impossible for the developers to predict all the context providers that might produce context information at runtime. In addition, using this methodology means that the developer is required to design a separate plug-in architecture for each context provider, which is proportional to the available number of context providers. Additionally, this methodology does increases the development effort as each plug-in requires a separate development process.

3.6. U-Music MDD

Khan [29] proposed U-MUSIC methodology. U-MUSIC adapts a model-driven approach to construct self-adaptive applications and uses a component model for achieving unanticipated adaptation. However, the author has modified the MUSIC methodology to support semi-anticipated

adaptation; also called planning-based adaptation, which enables the software to adapt to foreseeable context changes. U-MUSIC enables developers to specify the application variability model, context elements, and data structure. The developers are able to model the component functionalities and QoS properties in an abstract, platform-independent model. In U-MUSIC, dynamic decision-making is supported by the MUSIC middleware mentioned above. However, this approach suffers from a number of drawbacks. First, it is well known that correct identification of the weight for each goal is a major difficulty for the utility function. Second, the approach hides conflicts between multiple goals in its single, aggregate objective function, rather than exposing the conflicts and reasoning about them. Finally, it assumes that the code generation process can produce high quality of code based on the variability models in automatic and unsupervised process by the software developers.

3.7. CAUCE

CAUCE proposed as a model-driven development approach [30]. The authors defined an MDA approach that focuses on three layers of models. The first layer confirms to the computational independent model for capturing the conceptual properties of the applications. The second layer defines three complementary points of view of the software systems. These views include deployment, architecture and communication. The third layer focuses on converting the conceptual representation of the context-aware application into a software representation using a multi model transformation. The Atlas Transformation Language (ATL) is used to interpret the model and convert them into a set of models conforming to the platform independent model. The final model is transformed using the MOF Script language based on the EMF paradigm [14]. The CAUCE methodology focuses more on the CIM by splitting this layer into three layers of abstraction, which confirms to the tasks, social and space meta models. The task model focuses on modeling a set of tasks and the relationships among them that any entityin the system is able to perform. The social metamodel defines the social environment of the entities in the system and is directly related to the entity task and entity information that identify the context-aware application behavior. The space metamodel defines the physical environment of the entities in the system. Therefore, this metamodel is directly related to the physical conditions, infrastructure and location characteristics of the context-aware applications.

However, the CAUCE methodology provides a complete development process for building context-aware applications. Despite that, CAUCE is limited to specific modeling tool and language, in this case the UML is integrated with EMF. The generated application can only be implemented using Java language as the ATL and MOF Script languages support it. However, it is impossible for the developers to adapt CAUCE for building heterogeneous and distributed mobile applications, which might have multiple deployment platforms that require variant numbers of implementation languages.

3.8. ContextUML

Generally, UML profiles and metamodels are used to extend the UML language semantics. ContextUML was one of the first approaches that targeted the modeling of the interaction between context and web service applications [31]. Prezerakos et al. [32] extended ContextUML using aspect-oriented programming and service-oriented architecture to fulfill the needs for dynamic adaptation in service-oriented architecture. However, contextUML used a UML metamodel that extended the regular UML by introducing appropriate artifacts that used to create context-aware applications. The contextUML produces a class diagram, which corresponds to the context class and to specific services. They mitigate the UML relationship and dependency to express the interaction between the context information and the respective services. A means of parameters injection and service manipulation are used to populate specific context-related parameters in the application execution loop.

However, the UML profiles and metamodels lack from several features required for modeling the self-adaptive software system. Ignoring the heterogeneity of the context information, they based their claims on the nature of the context values, which can fluctuate and evolve significantly at runtime. It is not feasible to this study how the behavior is modeled when multiple context values have changed at the same time.

3.9. COCA-MDA

According to the COCA-MDA approach, the software self-adaptability and dependability can be achieved by dynamically composing software from context-oriented modules based on the context changes rather than composing the software from functional-oriented modules. Such composition requires the design of software modules to be more oriented towards the context information rather than being oriented towards the functionality. The principle of context orientation of software modules was proposed in the COCA-MDA [33]. COCA-MDA proposes a decomposition mechanism of software based on the separation between context-dependent and context-independent functionality. Separating the context-dependent functionality from the context-independent functionality enables adaptability and dependability of software systems with the aid of middleware technology. The middleware can adapt the software behavior dynamically by composing interchangeable context-dependent modules based on context changes. COCA-MDA proposes that software self-adaptability is achieved by having both adaptive middleware architecture and a suitable decomposition strategy, which separates the context-dependent functionality from the context-independent functionality of the software systems.

COCA-MDA 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 behaviors and provides the software with capabilities of self-adaptability and dependability in mobile computing environment. The context-oriented component model encapsulates the implementation of the context-dependent parts in distinct architectural units, which enables the software to adjust its functionality and/or behavior dynamically. This differs from the majority of existing work, which seeks to embed awareness of context in the functional implementation of applications. The context-oriented software is developed using a COCA-MDA. Afterwards, the context-oriented software is manipulated at runtime by a COCA-middleware that performs a runtime behavioral composition of the context-dependent functionality based on the operational context. The self-adaptive software dependability is achieved through the COCA-middleware capability in considering its own functionality and the adaptation impact/costs. A dynamic decision-making based on a policy framework is used to evaluate the architecture evolution and verifies the fitness of the adaptation output with the application's objectives, goals and the architecture quality attributes.

The COCA-MDA follows the principles of OMG model-driven architecture. In MDA, there are three different views of the software: the Computation Independent View (CIV), the Platform Independent View (PIV), and the Platform Specific View (PSV). The CIV focuses on the environment of the system and the requirements for the system, and hides the details of the software structure and processing. The PIV focuses on the operation of a system and hides the details that are dependent on the deployment platform. The PSV combines the CIV and PIV with an additional focus on the details of the use of a specific platform by a software system [34]. COCA-MDA partitioning the software into three views: the structure, behavior, and enterprise viewpoints. The structure view focuses on the core component of the self-adaptive application and hides the context-driven component. The behavior view focuses on modeling the context-driven behavior of the component, which may be invoked in the application execution at runtime. The enterprise view focuses on remote components or services, which may be invoked from the distributed environment. 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. A state-machine model is used to describe the DPL by specifying a set of internal and external variables, and conditional rules. The rules determine the true action or the else part of the 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.

COCA-MDA was used for building self-adaptive applications for indoor wayfinding for individuals with cognitive impairments as proposed in [35]. Evaluating the COCA-MDA productivity among the development cost and effort using the Constructive Cost Model II (COCOMO II) [36] was demonstrated in [37]. This article focuses on evaluating the features of COCA-MDA against other MDA approaches as shown in the following section.

4. Feature Analyses And Comparative Study Of MDA-Based Approaches

From the software developer's perspective, it is vital to know the features of the development paradigm, which might be used in constructing a self-adaptive application. Feature evaluation of the development methodology can assist the developers in selecting among the proposed methodologies in the literature for achieving adaptability and dependability of the software systems. Improving the development of self-adaptive software systems using model driven approach has attained several research efforts. The target was in general to introduce software with adaptability and variability while focusing on reducing the software complexity and optimizing the development effort.

The examination of software system performance, dependability and availability is of greatest importance for tuning software system in conjunction with several architecture quality attributes. Such performance analysis was considered by the modeling Specification and Evaluation Language (MOSEL) [38]. The system modeling using MOSEL illustrates how easily it can be used for modeling real-life examples from the fields of computer communication and manufacturing systems. However, extending the MOSEL language towards the modeling and performance evaluation of self-adaptive software system can estimate several qualityattributes of model-based architecture and provides early results about how efficient is the adaptation action.

Kitchenham et al. in [39] proposed the DESMET method, which evaluates software development methodologies using an analytical approach. Asadi et al. have adapted the DESMET method to analyze several MDA-approaches. The authors adapted several evaluation criteria that can be used to compare MDA methodologies based on MDA-related features and MDA-based tool features [40].

However, Calic et al. [41] proposed an evaluation framework to evaluate MDA-based approaches in terms of four major criteria groups, as follows: I) MDA-related features: The degree to which the proposed methodologies are compliant with OMG's MDA specification [42]. II) Quality: Evaluation of the overall quality of the MDA-based approaches including their efficiency, robustness, understandability, ease of implementation, completeness, and ability to produce the expected results [39]. III)

Usability: Simplicity of use and ease of implementation by the developer, which covers clear information about the impact of the methodology on the development effort [43], [44]. IV) Productivity: The quality of benefits derived from using the methodology and its impact on the development time, complexity of implementation, code quality, and cost effectiveness [41]. Calic et al. [41] presents the COPE tool, to evaluate the MDA productivity by automating the coupled evaluation of metamodels and model by recording the coupling history in a history model.

Lewis et al. [45] have evaluated the impact of MDA on the development effort and the learning curve of the MDA-based development tools based on their own experiences. The authors concluded that the real potential behind MDA is not completely employed either by current tools or by the proposed MDA approaches in the literature. In addition, the developers have to modify the generated code such that it is suitable for the target platform. The MDA tools can affect the level of maintenances required for the generated codes. In the same way, the developer's level of understanding of MDA tasks and their familiarity with the target platform have direct impacts on MDA productivity.

The COCOMO II [36] emerged as software cost estimation model, which considers the development methodology productivity. The productivity evaluates the quality of benefits derived from using the development methodology, in terms of its impact on the development time, complexity of implementation, code quality, and cost effectiveness [41]. COCOMO II allows estimation of the effort, time, and cost required for software development. The main advantage of this model over its counterparts such as the Software Life-cycle Management (SLIM) model [46] and the System Evaluation and Estimation of Resources Software Estimation Model (SEER-SEM) [47] is that COCOMO II is an open model with various parameters which effect the estimation of the development effort. Moreover, the COCOMO IImodel allows estimation of the development effort in Person-Months (PM) and the Time to Develop (TDEV) a software application. A set of inputs such as software scale factors (SF) and 17 effort multipliers is needed. A full description of these parameters is given in the COCOMO II model definition manual, which can be found in [36]. An example of an evaluation of MDA approaches with COCOMO II can be found in [48].

In this research, we intend to use an evaluation framework that can test and qualify the ability of MDA-based approaches to produce the expected results [39] in terms of dynamic adaptation in general, and self-adaptability, in specific. These features are evaluated in the following sections.

4.1. Existence of MDA-related Features

Table 1. *MDA-related features.*

Group criteria	Features	Criterion Type	Description of level	UML-based meta model	CAMEL	A-MUSE	MUSIC	Paspalis	U-MUSIC	CAUCE	COCA-MDA
Existence of MDA-related Features	Tool suit and implementation:	Scale form	**A.** The methodology does not provide a specific tool and there are no explicit guidelines as to how to select an appropriate alternative. **B.** The methodology does not provide a complete toolset, or only general guidelines are provided for selecting alternative tools. **C.** The methodology provides a complete toolset, or provides precise guidelines for selecting appropriate alternative tools.	A	A	A	C	B	B	A	A
	Computational independent model	Scale form	**A.** Production of the model are not addressed by the methodology.	A	A	A	B	B	B	C	C
	Platform independent model	Scale form	**B.** The methodology provides general guidelines for creating the model; creation steps are not determined precisely. **C.** The methodology explicitly describes steps and techniques for creating the model.	B	B	B	C	B	C	C	C
	Platform specific model	Scale form		B	B		C	B	C	C	C
	Verification and validation	Scale form	**A.** The activity is not defined and is devolved to the developers. **B.** The activity is defined by the methodology, but not in detail. **C.** The methodology provides explicit and detailed guidelines and techniques for performing the activity.	B	A	B	C	A	A	A	B
	Source model. Target model synchronization	Scale form		A	A	B	C	A	B	A	A
	Use of UML profiles	Narrative	**Involved**: the Methodology depend on the UML Profile **Devolved**: Methodology is not using UML profile	Involved	Involved	Devolved	Devolved	Devolved	Devolved	Devolved	Devolved

MDA features refers to the degree to which the proposed methodologies are compliant with the OMG's MDA specifications; these specifications can be divided into the support of CIM, PIM, PSM, model validation, and transformation [42]. In terms of MDA features, we adapt the criteria proposed by Asadi and Ramsin [40], which highlights the methodology's conformance to the original OMG standard, as shown in Table 1. Feature analysis can

be performed in two ways: scale form and narrative form. The scale form attaches the methodology compliant to a specific feature, which is divided into three ranks, from A to C, as shown in each table. The narrative form captures whether the methodology covers a specific feature based on the level of involvement.

4.2. Tool-related Feature Analysis

The major challenges arise from concerns associated with providing support for creating and using an appropriate modeling abstraction for analyzing and designing the software [17]. A second challenge posed by Asadi and Ramsin [40], that each development methodology generates more specific technical details that suit the underlying modeling language or modeling tool they used, as each tool requires a learning curve, and it might have some limitation with regard to the platform and the number of implementation languages they support [45]. This implies that a MDD approach should be decoupled from using a specific tool or modeling language. The developers have to be free on selecting the tool(s) that fits their needs and the software under development.

On the other hand, MDD approaches should focus more on describing standard development processes without relaying on a specific technology or platforms like EMF and ECORE. In terms of the tools the methodology used, the features that highlight the methodology dependency on the modeling languages and tools are shown in Table 2.

Table 2. *Modeling tools-related features.*

Criterion Name	Feature	Description of level	UML-based meta model	CAMEL	A-MUSE	MUSIC	Paspalis	U-MUSIC	CAUCE	COCA-MDA
Existence of tool-related	Model To Model transformation	Involved : The methodology explicitly participates in the activity and provides precise techniques/ guidelines Devolved: The activity is developed to the tools and the methodology does not prescribe the steps that should be performed by the tools	Devolved	Involved	Involved	Involved	Involved	Involved	Involved	Involved
	Model to code transformation		Devolved	Devolved	Involved	Involved	Involved	Involved	Involved	Involved
	Meta-model maintainability		Devolved	Involved	Involved	Involved	Devolved	Involved	Devolved	Involved
	Verification of the generated model and code		Devolved	Devolved	Devolved	Involved	Involved	Involved	Devolved	Involved
	Traceability between models		Devolved	Devolved	Involved	Involved	Involved	Involved	Devolved	Involved

4.3. Quality of the MDA-based Approaches

Quality refers to the overall quality of the MDA-based approaches, including their efficiency, robustness, and understandability, In addition the ease of implementation, completeness, and the software ability to produce the expected results are included [39]. However, in this research, we have focused on the ability of the MDA-based approaches to provide the expected results that support the adaptability of the generated software, whether these results are derived from the code or the architecture. Moreover, we have split these criteria into four groups: requirements engineering, unanticipated awareness, context model, and modeling context-dependent behavioral variations.

4.3.1. Requirements Engineering of Context-dependent Behavioral Variations

Requirements engineering refers to the causes of adaptation, other than the functional behavior of the self-adaptive system. Whenever the system captures a change in the context, it has to decide whether it needs to adapt. The MDA-based approaches in the related work were evaluated regarding whether they support the modeling of context requirements as a specific feature and whether they support the requirements engineering in general, as shown in Table 3.

In addition, the methodology's ability to analyze and models the context-dependent behavior variations requires the MDD supports at three levels. The first is the requirement analysis at the computational independent model. The second is the representation of these requirements by means of UML objects at the platform independent model and platform specific model. The third is the representation of the context-dependent behavior as runtime objects, which are a code representation of these requirements [49,50]. However, the evaluation of these criteria is shown in Table 3.

4.3.2 Unanticipated Awareness

This feature captures whether a context change can be predicted ahead of time [51]. Anticipation can be classified into three degrees: foreseen, foreseeable, and unforeseen changes. Foreseen refers to the changes that are handled in the implementation code. Foreseeable refer to the context changes that were predicted at the software design.

Unforeseen refers to the changes that are not modeled at the design or the implementation stage, but are to be handled at runtime [52]. The evaluation criteria are shown in Table 4 with their related scale form.

Table 3. *Supporting context-dependent behavior variations on the analysis, design and implementation.*

Criterion Name	Features	Criterion Type	Description of level	UML metmodel	CAMEL	A-MUSE	MUSIC	Paspallis	U-MUSIC	CAUCE	COCA-MDA
Context-dependent variations management	Requirements analysis in the CIM	Narrative	Involved : The methodology supports context-dependent behaviour concerns. Devolved : The methodology does not support context-dependent behaviour concerns.	Devolved	Involved	Devolved	Devolved	Involved	Devolved	Devolved	Involved
	Modelling Context-dependent behaviour at PIM	Narrative	Involved : The methodology has supports for modelling the context-dependent behaviours at the PIM. Devolved : The methodology has no supports for modelling the context-dependent behaviours at the PIM.	Devolved	Involved	Devolved	Devolved	Devolved	Devolved	Devolved	Involved
	Modelling Context-dependent behaviour at PSM.	Narrative	Involved : The methodology has supports for modelling the context-dependent behaviours at the PSM. Devolved : The methodology has no supports for modelling the context-dependent behaviours at the PSM.	Devolved	Involved	Devolved	Devolved	Devolved	Devolved	Involved	Involved

Table 4. *Anticipation of context change-related criteria and evaluation results.*

Criterion Name	Features	Criterion Type	Description of level	UML metamodle	CAMEL	A-MUSE	MUSIC	Paspallis	U-MUSIC	CAUCE	COCA-MDA
unanticipated awareness	Foreseen changes	Scale form	A. The methodology takes care of the context changes implicitly by the code. B. The methodology takes care of the context changes explicitly by means of UML model C. The methodology takes care of the context changes explicitly by enabling the developer to model them in abstract level.	B	A	B	C	B	C	C	C
	foreseeable changes	Scale form	A. The methodology enables the developer to plane for the context changes implicitly by maintaining the code. B. The methodology planned for the context changes explicitly by the variation model supported by planning-based adaptation at runtime. C. The methodology takes care of the context changes and enables the developer to model them in an abstract level.	B	A	B	C	B	C	B	C
	Unforeseen Changes	Scale form	A. The methodology anticipates the context changes implicitly by maintaining the code to handle them , static adaptation. B. The methodology anticipates the context changes explicitly and enables the developer to specify several application variations models supported by refining the base model. C. The methodology anticipates the context changes at runtime by means of requirements' reflection and allows the developers to represent them as runtime objects.	A	A	B	B	B	B	B	C

4.3.3 Context Model

This captures the ability of the methodology to incorporate the context information using the 'separation of concerns' technique between the context information model and the business logic. The first criterion focuses whether the methodology supports/uses the separation of concerns in the development processes. The second criterion refers to the ability to bind the context source to the context provider,

as proposed by Sen and Roman [53] and Broens et al. [54] and Paspallis [55]. The binding mechanism enables the developers to map each context cause to the affected architectural units. The binding mechanism also enables the application to determine which part has to manage the context changes, by means of the adaptation mechanism.

The evaluation criteria for the context model are shown in Table 5.

Table 5. *Context model-related criteria and evaluation results.*

Criterion Name	Features	Criterion Type	Description of level	UML metmodel	CAMEL	A-MUSE	MUSIC	Paspallis	U-MUSIC	CAUCE	COCA-MDA
Context model	Separation of concerns between the context model and the business logic code.	Narrative	Implicitly: The context model is implicitly handled by the generated code. Explicitly: The context model is explicit separated from the generated code.	Implicitly	Implicitly	Implicitly	Explicitly	Explicitly	Implicitly	Implicitly	Explicitly
	Context Binding	Narrative	Involved : The methodology binding the context information to architectural units Devolved: The methodology is does not support context binding	Involved	Involved	Devolved	Devolved	Involved	Devolved	Devolved	Involved

4.3.4 Modeling Context-dependent Behavior

These criteria refer to the ability of the model to capture the impact of context changes on the self-adaptive application's behavior. However, Hirschfeld et al. [2] classified these changes into three kinds of variations: actor dependent, system dependent, and environment dependent behavioral variations. This behavioral variation requires a separation between their concerns, by separating the context handling from the concern of the application business logic. In addition, a separation between the application-dependent parts from the application-independent parts can support behavioral modularization of the application, thereby simplifying the selection of the appropriate parts to be invoked in the execution, whenever a specific context condition is found. The behavioral modeling criteria are shown in Table 6.

Table 6. *Modeling context-dependent behaviors.*

Criterion Name	Features	Criterion Type	Description of level	UML metmodel	CAMEL	A-MUSE	MUSIC	Paspallis	U-MUSIC	CAUCE	COCA-MDA
Modelling Context-dependent Behaviour	Actor-dependent behaviour	Scale form	A. The methodology does not capture the actor-dependent behaviour B. The methodology capture actor-dependent, implicitly by means code weaving and advise. C. The methodology capture actor-dependent behaviour in abstract level, manipulating the behaviour performed at runtime by means of compositional reflection	A	A	A	A	A	A	A	C
	System-dependent behaviour	Scale form	A. The methodology does not capture the system-dependent behaviour. B. The methodology capture system-dependent, implicitly by means of code weaving and advise methods. C. The methodology capture system-dependent behaviour in abstract level, manipulating the behaviour performed at runtime by means of compositional reflection	A	A	A	B	A	B	B	C
	Environment-dependent behaviour	Scale form	A. The methodology does not capture the environment-dependent behaviour B. The methodology capture environment-dependent, implicitly by means of code weaving and advise methods. C. The methodology capture environment-dependent behaviour in abstract level, manipulating the behaviour performed at runtime by means of compositional reflection	A	A	A	A	A	A	A	C

5. Evaluation Results

Based on the analysis results shown in Tables 1, 2, 3, 4, 5, and 6, we find that the discussed methodologies in the related work suffer from several critical failings in terms of their conformance to the OMG's guidelines for MDA methodology [4].

First, it is well known that correct identification of the weight of each goal is a major difficulty for the utility functions as shown in the MUSIC, U-MUSIC and Paspallis

methodologies.

Second, these approaches hide conflicts among multiple adaptation goals by combining them into a single, aggregate objective function, rather than exposing the conflicts and reasoning about them. On the other hand, it would be optimistic to assert that the process of code generation from models can become completely automatic or that the developer's role lies only in application design, as discussed in the above with regard to CAMEL and A-MUSE.

Third, it is impossible for the developers to predict the possible application variations, which will extend the application behavior when unanticipated conditions arise, this applied to all methodologies mentioned in the above.

In addition, mobile devices have limited resources for evaluating many application variations at runtime, which might consume significant amounts of the allocated resources. As a result, the benefits gained from the adaptationprocess areachieved by drained the allocated resources because theadaptation processneedlong processing time to achieve the adaptation. Fourth, the previously mentioned methodologies produce an architecture with a tight coupling between the context provider and the context consumer, which may cause the middleware to notify multiple components about multiple context changes. Finally, all the methodologies seem to generate an architecture that is tightly coupled with the target platform for deployment and the modeling tools they used.

In addition, the developers have to explicitly predict the final composition of the software and the possible variations of the application, whether at the platform independent model or through the model transformation. Moreover, the developers have to modify the generated code to be suitable for deployment on the target platform and to be integrated with the middleware implementation, which is in the best case made a huge gap between the middleware designer and the application developer. Understandings the modeling tasks and the target platform configurations have limited software developers from employing MDA-approaches in several platforms.

6. Conclusions

The MDD-based approaches evaluated in thisstudy suffer from a number of drawbacks. First, there is no guaranty that the code generation process and model transformation of the software models can become completely automatic and unsupervised by the software developer's. Alternatively model-checking techniques are used to check the validity of the generated code or software models. Second, it is impossible for the developer to predict all possible variations of the application when unanticipated conditions will arise. In addition, mobile devices have limited resources for evaluating many application variations at runtime and can consume significant amounts of the allocated resources. As result of that, the advantages from the adaptation processareoverwhelmed by the overhead required to achieve the adaptation output. Third, each development methodology generates more specific technical details that suit the underlying implementation language or modeling tool they used. Fourth, the impact of MDA-approach over the development cost must be consider by the software developers for selecting the most appropriate methodology for their project. We found that COCA-MDA is more capable to meet the requirements of self-adaptive systems and allows the software developers to adapt it in wide range of execution platforms.

Acknowledgement

Research presented in this paper was funded by a Strategic Research Cluster grant (07/SRC/I1168) by Science Foundation Ireland under the National Development Plan. The authors gratefully acknowledge this support.

References

[1] Parashar, M., Hariri, S.: Autonomic computing: An overview. Unconventional Programming Paradigms (2005) 257–269.

[2] Hirschfeld, R., Costanza, P., Nierstrasz, O.: Context-oriented programming. Journal of Object Technology 7(3) (March 2008) 125–151.

[3] Oreizy, P., Gorlick, M., Taylor, R., Heimhigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D., Wolf, A.: An architecture-based approach to self-adaptive software. Intelligent Systems and Their Applications 14(3) (1999) 54–62.

[4] Kleppe, A.G., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing, Boston, MA, USA (2003).

[5] Inverardi, P., Tivoli, M.: The future of software: Adaptation and dependability. In Lucia, A., Ferrucci, F., eds.: Software Engineering. (2009) 1–31.

[6] Apel, S., Leich, T., Saake, G.: Aspectual mixing layers: aspects and features in concert. In: Proceedings of the 28th international conference on Software engineering. (ICSE '06), Shanghai, China, ACM (2006) 122–131.

[7] Lincke,J.,Appeltauer,M.,Steinert,B.,Hirschfeld,R.:Anopeni mplementationfor context-oriented layer composition in contextjs. Science of Computer Programming 76 (December 2011) 1194–1209.

[8] Carton, A., Clarke, S., Senart, A., Cahill, V.: Aspect-oriented model-driven development for mobile context-aware computing. In: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments. (SEPCASE '07), Washington, DC, USA (2007) 5–10.

[9] Kuwadekar, A., Joshi, A., Al-Begain, K.: Real time video adaptation in next generation networks. In: Proceedings of Fourth International Conference on Next Generation Mobile

Applications, Services and Technologies (NGMAST 2010). Volume 1 of LNCS., Amman, Jordan (july 2010) 54 –60.

[10] A. Kuwadekar, C. Balakrishna, and K. Al-Begain, "Genxfone: design and implementation of next-generation ubiquitous sip client," in Proceedings of the Second International Conference on Next Generation Mobile Applications, Services and Technologies, ser. (NGMAST '08), Cardiff, UK, sept. 2008.

[11] Belaramani, N.M., Wang, C.L., Lau, F.C.M.: Dynamic component composition for functionality adaptation in pervasive environments. In: Proceedings of the Ninth IEEE Workshop on Future Trends of Distributed Computing Systems. (FTDCS '03), San Juan, Puerto Rico (May 2003) 226–232.

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

[13] Harrison, W.: Modularity for the changing meaning of changing. In: Proceedings of the tenth international conference on Aspect-oriented software development. (AOSD '11), Porto de Galinhas, Brazil (2011) 301–312.

[14] Eclipse: Eclipse modeling framework. http://www.eclipse.org/modeling/emf/ (November 2012) [Online; accessed 1-November-2012].

[15] de Farias, C.R.G., Leite, M.M., Calvi, C.Z., Pessoa, R.M., Filho, J.G.P.: A mof metamodel for the development of context-aware mobile applications. In: Proceedings of the 22nd Annual ACM Symposium on Applied Computing. (SAC '07), Seoul, Korea (2007) 947–952.

[16] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.: An overview of aspectj. In: Proceedings of the 15th European Conference on Object-Oriented Programming, (ECOOP 2001). Volume 2072 of LNCS., Budapest, Hungary (2001) 327–354.

[17] France, R., Rumpe, B.: Model-driven development of complex software: A research roadmap. In: Proceedings of the Future Of Software Engineering. (FOSE '07), Washington, DC, USA (2007) 37–54.

[18] Daniele, L.M., Ferreira Pires, L., Sinderen, M.: An mda-based approach for behaviormodeling of context-aware mobile applications. In: Proceedings of the 5th European Conference on Model Driven Architecture Foundations and Applications. (ECMDA-FA '09), Birmingham, UK (2009) 206–220.

[19] Sindico, A., Grassi, V.: Model driven development of context aware software systems. In: Proceedings of International Workshop on Context-Oriented Programming. (COP '09), Genova, Italy (2009) 7:1–7:5.

[20] Bartolomeo, G., Salsano, S., Melazzi, N., Trubiani, C.: Smile: simple middleware independent layer for distributed mobile applications. In: Proceedings of the Wireless Communications and Networking Conference. (WCNC 2008), Las Vegas, USA (31 2008-april 3 2008) 3039–3044.

[21] Appeltauer, M., Hirschfeld, R., Masuhara, H.: Improving the development of context-dependent java applications with contextj. In: Proceedings of the International Workshop on Context-Oriented Programming. (COP '09), Genova, Italy (2009) 5:1–5:5.

[22] Appeltauer, M., Hirschfeld, R., Haupt, M., Masuhara, H.: Contextj: Context-oriented programming with java. Information and Media Technologies 6(2) (2011) 399–419.

[23] Floch, J., Hallsteinsen, S., Stav, E., Eliassen, F., Lund, K., Gjorven, E.: Using architecture models for runtime adaptability. IEEE software 23(2) (2006) 62–70.

[24] Rouvoy, R., Beauvois, M., Lozano, L., Lorenzo, J., Eliassen, F.: Music: an autonomous platform supporting self-adaptive mobile applications. In: Proceedings of the 1st workshop on Mobile middleware: embracing the personal communication device. (MobMid '08), Leuven, Belgium (2008) 6:1–6:6.

[25] Rouvoy, R., Barone, P., Ding, Y., Eliassen, F., Hallsteinsen, S., Lorenzo, J., Mamelli, A., Scholz, U.: Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In Cheng, B.H., Lemos, R., Giese, H., Inverardi, P., Magee, J., eds.: Software Engineering for Self-Adaptive Systems. (2009) 164–182.

[26] Osgi: the dynamic module system for java. http://www.osgi.org/Main/HomePage (November 2010) [Online; accessed 1-November-2010].

[27] Paspallis, N.: Middleware-based development of context-aware applications with reusable components. PhD thesis, University of Cyprus, Department of Computer Science (Nov 2009).

[28] Reichle, R., Wagner, M., Khan, M.U., Geihs, K., Lorenzo, J., Valla, M., Fra, C., Paspallis, N., Papadopoulos, G.A.: A comprehensive context-modeling framework for pervasive computing systems. In: Proceedings of the 8th international conference on Distributed applications and interoperable systems. (DAIS '08), Oslo, Norway (2008) 281–295.

[29] Khan, M.U.: Unanticipated Dynamic Adaptation of Mobile Applications. PhD thesis, University of Kassel, Distributed Systems Group, Kassel, Germany (may 2010).

[30] Tesoriero, R., Gallud, J., Lozano, M., Penichet, V.: Cauce: Model-driven development of context-aware applications for ubiquitous computing environments. Journal of Universal Computer Science 16(15) (July 2010) 2111–2138.

[31] Sheng, Q., Benatallah, B.: Contextuml: a uml-based modeling language for model-driven development of context-aware web services. In: Proceeding of the 4th International Conference on Mobile Business. (ICMB '05), Sydney, Australia (2003) 11–13.

[32] Prezerakos, G., Tselikas, N., Cortese, G.: Model-driven composition of context-aware web services using contextual and aspects. In: Proceedings of the IEEE International Conference on Web Services. (ICWS 2007), Utah, USA (July 2007) 320–329.

[33] Magableh, B., Barrett, S.: Context oriented software development. Journal of Emerging Technologies in Web Intelligence (JETWI) 3(4) (June 2012) 206–216.

[34] Miller, J., Mukerji, J.: Mda guide version 1.0.1. Technical report, Object Management Group (OMG) (2003).

[35] Magableh, B., Barrett, S.: Self-adaptive application for indoor wayfinding for individuals with cognitive impairments. In: Proceedings of the 24th International Symposium on Computer-Based Medical Systems. Number 1 in (CBMS '11), Bristol, United Kingdom (june 2011) 1 –6.

[36] Boehm, B.W., Clark, Horowitz, Brown, Reifer, Chulani, Madachy, R., Steece, B.: Software Cost Estimation with Cocomo II. 1st edn. Prentice Hall PTR (2000).

[37] Magableh, B., Barrett, S.: Model-Driven productivity evaluation for self-adaptive Context-Oriented software development. In: Proceedings of the 5th International Conference and Exhibition on Next Generation Mobile Applications, Services, and Technologies. (NGMAST'11), Cardiff, Wales, United Kingdom (September 2011) 158–167.

[38] Begain, K., Bolch, G., Herold, H.: Practical Performance Modeling: Application of the Mosel Language. Kluwer Academic Publishers, Norwell, MA, USA (2001).

[39] Kitchenham, B., Linkman, S., Law, D.: Desmet: a methodology for evaluating software engineering methods and tools. Computing & Control Engineering Journal 8(3) (2002) 120–126.

[40] Asadi, M., Ramsin, R.: Mda-based methodologies: An analytical survey. In: Proceedings of the Euro Conference on Model Driven Architecture Foundations and Applications. (ECMDA-FA 2008), Berlin, Germany (2008) 419–431.

[41] Calic, T., Dascalu, S., Egbert, D.: Tools for mda software development: Evaluation criteria and set of desirable features. In: Proceedings of the Fifth International Conference on Information Technology. (ITNG 2008), Istanbul, Turkey (2008) 44–50.

[42] OMG, O.M.G.: Software & systems process engineering meta-model specification. Technical report, Object Management Group (2010).

[43] Norman, D.A.: The Design of Everyday Things. Reprint paperback edition. Basic Books (September 2002)

[44] Preece, J., Rogers, Y., Sharp, H., eds.: Interaction Design: Beyond Human-Computer Interaction. John Wiley and Sons (January 2002).

[45] Lewis, G., Wrage, L.: Model problems in technologies for interoperability: Model-driven architecture. Technical report, Software Engineering Institute (2005).

[46] Estell, R.G.: Software life cycle management. International Journal of Management Reviews 5 (August 1976) 2–15.

[47] Galorath, D.D., Evans, M.W.: Software Sizing, Estimation, and Risk Management. Auerbach Publications (2006).

[48] Achilleas: Model-Driven Petri Netbased Framework for Pervasive Service Creation. PhD thesis, University of Essex (2010).

[49] Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A., Letier, E.: Requirements reflection: requirements as runtime entities. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, (ICSE '10). Volume 2 of LNCS., CAPE TOWN, South Africa (2010) 199–202.

[50] delmos,R.,Giese,H.,Mu¨ller,H.,Shaw,M.,Andersson,J.,Baresi ,L.,Becker,B., Bencomo, N., Brun, Y., Cikic, B., Desmarais, R., Dustdar, S., Engels, G., Geihs, K., Goeschka, K.M., Gorla, A., Grassi, V., Inverardi, P., Karsai, G., Kramer, J., Litoiu, M., Lopes, A., Magee, J., Malek, S., Mankovskii, S., Mirandola, R., Mylopoulos, J., Nierstrasz, O., Pezz`e, M., Prehofer, C., Scha¨fer, W., Schlichting, W., Schmerl, B., Smith, D.B., Sousa, J.P., Tamura, G., Tahvildari, L., Villegas, N.M., Vogel, T., Weyns, D., Wong, K., Wuttke, J.: Software engineering for self-adaptive systems: A second research roadmap. In: Proceedings of Dagstuhl Seminar, Software Engineering for Self-Adaptive Systems. (Dagstuhl Seminar '11), Dagstuhl, Germany (2011) 1–26.

[51] Cheng, B.H., Giese, H., Inverardi, P., Magee, J., de Lemos, R., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Serugendo, G.D.M., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Mu¨ller, H., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., Whittle, J.: Software engineering for self-adaptive systems: A research road map. In: Proceedings of Dagstuhl Seminar, Software Engineering for Self-Adaptive Systems. (Dagstuhl Seminar '08), Dagstuhl, Germany (2008) 1– 26.

[52] Laprie, J.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 1 (Apr 2004) 11–33.

[53] Sen, R., Roman, G.: Context-sensitive binding, flexible programming using transparent context maintenance. Technical report, Department of Computer Science and Engineering Washington University in St. Louis (2003).

[54] Broens, T., Quartel, D., Van Sinderen, M.: Capturing context requirements. In: Proceedings of the 2nd European conference on Smart sensing and context. (EuroSSC '07), Kendal, England (2007) 223–238.

[55] Paspallis, N.: Software engineering support for the development of context-aware, adaptive applications for mobile and ubiquitous computing environments. http://www.cs.ucy.ac.cy/paspalli/phd/thesis-proposal.pdf (2010) "[online accessed 1-December-2010]".