Pervasive and Mobile Computing 6 (2010) 281–296

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

Context modelling and a context-aware framework for pervasive service creation: A model-driven approach☆

Achilleas Achilleos [a,*], Kun Yang [a], Nektarios Georgalas [b]

[a] Pervasive Systems Research Group, School of Computer Science and Electronic Engineering, University of Essex, Colchester, CO4 3SQ, United Kingdom
[b] Centre for Information and Security Systems, British Telecom Innovate, Ipswich, IP5 3RE, United Kingdom

ARTICLE INFO

Article history:
Received 28 February 2008
Received in revised form 12 April 2009
Accepted 1 July 2009
Available online 19 July 2009

Keywords:
Pervasive service creation
Context-aware framework
Context modelling
Model-driven development
Domain specific modelling

ABSTRACT

Pervasive service creation entails a complex process that involves a diversity of development aspects. Context-awareness is an important facet of pervasive service creation, which deals with the acquisition, rendering, representation and utilisation of context information. In this paper we tackle context-awareness at the application level dealing with the representation and utilisation of context by services. We propose a model-driven approach that facilitates the creation of a context modelling framework and simplifies the design and implementation of pervasive services. To conclude, we demonstrate the benefits of our model-driven approach via the creation of a pervasive museum service and its evaluation using selected software metrics.

1. Introduction

Conventional services represent software applications that can be deployed on a specific device and platform to support the execution of particular computing tasks. In contrast, pervasive services refer to software applications that can operate in a dynamic environment and have the capability to run anytime, anywhere and on any device with minimal user attention [1]. A pervasive service provides users with a specialised and personalised behaviour that allows performing dynamic computing tasks.

One characteristic feature of pervasive services is context-awareness, which depicts the necessity to react in accordance to certain predefined rules or on the basis of intelligent stimulus. This denotes the capability of the service to utilise context information in order to adapt dynamically its behaviour. Context has acquired a variety of meanings over the course of research [2]. In this work, we define context as: "*Any information relevant to the interaction of the user with the service, where both the user and the application's environment are of particular interest*". Understanding which information is termed as context, how to model and utilise them is particularly important in order to simplify the creation of pervasive services.

Service creation is a complex process, which involves a set of activities for the rapid analysis, design, implementation and validation of services [3,4]. The process is usually supported by a service creation framework, which aims to simplify service creation. Many technology-specific frameworks [3,5] have been developed to realise this objective. None of them though provides a clear-cut solution, due to the technology-specific complexities introduced. These frameworks aid technology experts but certainly do not assist novice users [4]. Hence, we argue that a model-driven generic framework [6] is required to provide solutions to these issues.

☆ The work presented in this paper is partly supported by British Telecom under the Model-driven Component-based Systems Engineering (MOSE) project and the UK Engineering and Physical Sciences Research Council (EPSRC) under project PANDA (Policy-based Model-driven Pervasive Service Creation and Adaptation).

* Corresponding author. Tel.: +44 0 78400 65217; fax: +44 0 1206 872900.
E-mail address: aaxilleas79@yahoo.gr (A. Achilleos).

1574-1192/$ – see front matter © 2010 Published by Elsevier B.V.
doi:10.1016/j.pmcj.2009.07.014

When dealing with pervasive services the complexity of the process is augmented due to the diversity of sources from which context information is obtained. In conventional services information comes mainly as input from the user and this manually supplied information drives the service execution. Pervasive services though rely on information that arises from a variety of sources; *e.g. sensors, repositories, users*. Therefore, the capability to effectively represent and manage context must be provided, in order to aid the creation of pervasive services.

Context-aware service creation has been studied during the course of research following two complementary directives. Several approaches [7,8] have been proposed that follow an infrastructure-based solution to the problem. These approaches provide an infrastructure capable of sensing, gathering and processing context information required by the pervasive service [9]. Although the process is simplified, the necessity to tailor the service implementation in accordance to the infrastructure's implementation arises. Consequently, these approaches restrict the developer to a specific implementation technology.

The complementary directive introduces approaches operating at the application level [10]. These approaches do not consider how information is acquired, gathered and processed to obtain an abstract context description. The primary requirement is the representation of context information in a format that can be realised and utilised by context-aware services. In principle these approaches are termed as context modelling techniques [10]. They deal with management tasks such as representation, administration and distribution of context information to services to achieve their adaptation.

Context modelling primarily tackles the representation of context information in the form of an abstract context model. The model is defined via the use of a context modelling framework, which comprises a modelling language and a supporting editor with drag and drop capabilities. Subsequently, the mapping of the modelling language to an implementation technology is defined, to facilitate the transformation of context models. The context model drives the generation of the implementation, which acts as the bridging point (e.g. similar to an API: Application Programming Interface) that allows context to be utilised by services [11]. The generated implementation typically serves tasks for managing a context repository such as querying, administrating and distributing context information to services.

An ideal context model should go head to head with the service creation framework into which it is to be implemented. A common software engineering technology that underpins both context modelling and a pervasive service creation framework can naturally bring context-awareness into pervasive services at the stage of service creation. One such technology is the Model Driven Architecture (MDA) [12,13] paradigm from the Object Management Group (OMG) [14]. In our previous work, a preliminary MDA-based service creation framework has been proposed and verified [15]. The work in this paper follows on our previous research outcomes to bring context-awareness into service creation via a model-driven technology, in particular, OMG's MDA. MDA's many advantageous features such as high-level abstraction and platform independence not only facilitate but also simplify the process of context modelling and its eventual implementation to particular programming languages [16].

The terms pervasive and context-aware are used interchangeably in this work. This is because we exclusively consider context-awareness as the prime characteristic feature of pervasive services. The paper promotes the thought of incorporating context-awareness into pervasive services at the static compile time, i.e., service creation stage. These mechanisms built in at the service creation stage will be triggered at the service execution phase to provide inherent and therefore much enhanced service adaptability. The proposed approach complements the main-stream service adaptation methodology that is largely based on a complicated middleware infrastructure.

In this paper two main technical contributions are introduced. First, we propose a model-driven methodology based on the MDA paradigm that facilitates the service creation process. We utilise MDA for context modelling and consider context modelling as part of the whole process of pervasive service creation. Second, we practise the methodology to design and develop a Context Modelling Framework (CMF) and verify its effectiveness using a pervasive service scenario. Note that the CMF is integrated into the generic framework as one of its components to facilitate the design, validation and implementation of pervasive services.

The remainder of this paper is structured as follows: Section 2 presents related research work on context-aware service creation. Section 3 introduces the model-driven methodology and presents the generic framework's software components and architecture. In Section 4 we perform the necessary requirements analysis of the context domain and introduce the proposed CMF, which is integrated into the generic framework to comprise the Context-Aware Pervasive Service Creation Framework (CA-PSCF). Section 5 demonstrates the applicability of the CA-PSCF for the creation of a pervasive museum service. An evaluation of the approach is also presented in this section using selected software metrics. Finally, in Section 6 we present the conclusions and future work.

2. Related work

Initial efforts on context-aware service creation focused on an infrastructure-based solution to the problem. In their work Dey and Abowd [7] define an architecture and present a Java-based Context Toolkit that simplifies context-aware service creation. The toolkit provides three abstract architectural components namely widgets, interpreters and aggregators. These components are responsible for the acquisition of context information from sensors as raw data and the processing of those data to obtain a high-level representation. As a result, this context information can be utilised by context-aware services to achieve their adaptation.

Table 1
Mapping the MDD process to the service creation process.

MDD process	Service creation process
Domain specific language definition	Service analysis
Domain model definition	Service design
Domain model validation	Service validation/testing
Domain model-to-model transformation	Service implementation/management
Domain model-to-code generation	Service implementation

Chen and Kotz [8] describe a graph abstraction for acquiring context in the form of raw data. These data are then processed to obtain a suitable abstract representation, which enables the distribution of context to services. The aforementioned initiatives help to accomplish low-level tasks that are important to application level approaches. Our work aligns with complementary application level approaches, which assume that this infrastructure level functionality exists and focus principally on context modelling [10] to simplify context-aware service creation.

Bauer in [17] proposes an extension to the Unified Modelling Language (UML), which allows modelling context information relevant to air traffic management in the form of a context model. The strength of the approach lies in the use of graphical models, defined using the widely accepted UML, which makes it easy to realise and transform context models. Despite the benefits of its approach, Bauer's model lacks a formal basis and consequently model evaluation is considered a downside. Furthermore, only a partial validation of models is applicable [10].

The UML based Context Modelling Profile (CMP) defined by Simons and Wirtz [18] allows one to model context for mobile distributed systems. UML stereotypes have been defined for the context modelling domain and Object Constraint Language (OCL) [19] constraints are enforced to ensure the validity of models. The approach benefits from the use of the widely accepted UML, since the CMP can be used in various UML tools. Despite that fact, these tools do not provide a standard way to access model stereotypes and enforce constraints [18]. Hence, constraints are imposed and enforced in this approach using the Eclipse Modelling Framework (EMF) [20]. Furthermore, mapping models to implementation is considered as future work.

Henricksen and Indulska in [9,21] propose an infrastructure and a framework to gather, manage and disseminate context to services. Context modelling concepts are introduced that facilitate the generation of a context management system from models. These are namely the context modelling language, the situation abstraction, the preference, branching and programming models. In their work formality of models is considered, diverse context sources are addressed and validation capabilities are provided. The approach is slightly hindered by the absence of a context modelling editor and the degree of automation provided for software generation.

In this paper we propose a model-driven approach that provides a higher level of automation in software generation. The approach is strictly based on the MDA paradigm and provides the capability to semi-automatically generate service creation environments (frameworks) for different application domains [15]. Hence, the approach and the generic framework are utilised to define and generate the CMF in the form of an Eclipse plug-in. The plug-in is then integrated into the generic framework, comprising a new software capability. Consequently, merely the modelling, validation and implementation tasks must be carried out for the creation of pervasive services. In addition the capability to generate diverse implementations and deploy pervasive services on different devices is provided. This simplifies the process and enables the rapid creation of pervasive services at the static compile time.

3. Generic model-driven methodology for pervasive service creation

3.1. The proposed model-driven development process

The model-driven development (MDD) process proposed in this paper aims to provide a systematic methodology that facilitates the generation of modelling frameworks and supports the overall service creation process. In this work we make explicit use of the devised methodology to support the creation of pervasive services. Table 1 presents a mapping between the phases of the MDD process and the phases of the service creation process.

The domain specific language (DSL) [22,23] definition phase reflects the service analysis phase. Fig. 1 illustrates the Steps 1–4 required for the definition of a DSL and the generation of a domain specific modelling framework. Initially the semantics of the domain are identified and mapped to modelling language constructs. The language is defined in the form of a metamodel, which comprises the elements, relationships and properties of the modelling domain. Furthermore domain rules are identified, mapped to constraints and imposed onto the language definition. The constraints are most commonly applied explicitly and separately of the domain metamodel definition; some rules are implicitly stated in the metamodel.

The metamodel definition and the successful imposition of constraints provide the capability to define coherent models. In order to design domain models, a graphical modelling framework is required. Implementing a modelling framework from scratch is often cumbersome and costly. Especially its maintenance introduces quite an overhead on the process and subsequently increases development time and costs. The generic framework provides the capability to simplify the creation and the maintenance of the modelling framework. This is performed by mapping the language constructs (domain metamodel) to graphical elements, forming the graphical metamodel. The domain and graphical metamodels define the entire set of artefacts required to enable the automatic generation of the domain specific modelling framework.

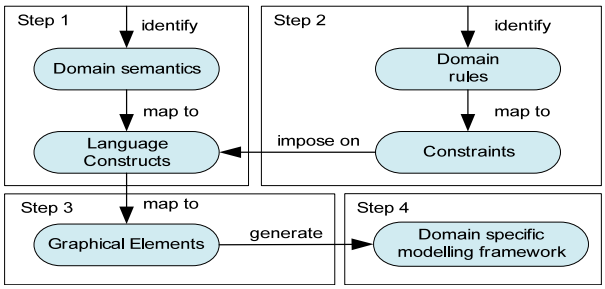


Fig. 1. Domain specific language definition; service analysis.

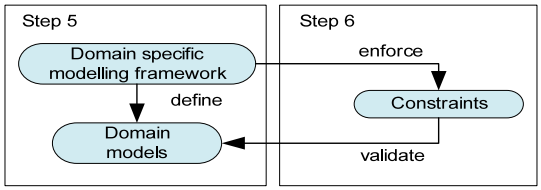


Fig. 2. Domain model definition and validation; service design and validation.

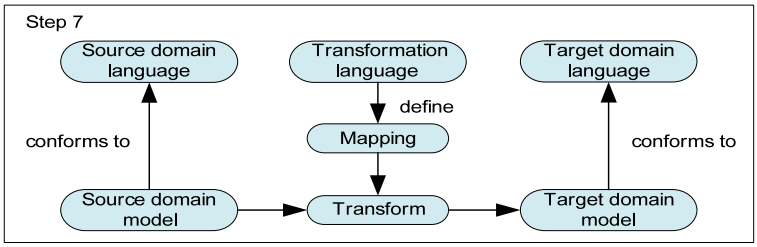


Fig. 3. Model-to-model transformation; service implementation/management.

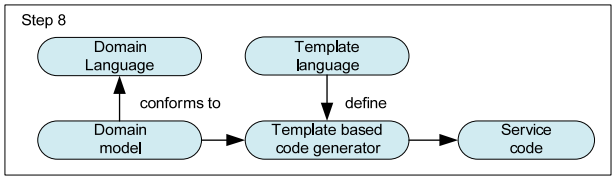


Fig. 4. Model-to-code generation; service implementation.

The generated domain specific modelling framework facilitates primarily the definition of models (Step 5), as illustrated in Fig. 2. Furthermore, the framework provides the capability to enforce the constraints (Step 6) imposed during the language definition phase. In this way the domain models can be validated to ensure their completeness and coherency. Consequently, non-erroneous implementations are automatically generated from the validated domain models.

The model-driven development process entails a model-to-model transformation phase (Step 7), which assists either the service implementation or the service lifecycle management. As illustrated in Fig. 3, via the use of the transformation language the mapping of the source language to the target language is defined. The transformation takes as input a model conforming to a metamodel and produces an output model conforming to another metamodel. In the case of the service implementation phase the transformation accepts as input a platform independent model (PIM) and generates an output platform specific model (PSM). The PSM includes implementation specific details and conforms to a metamodel, which reflects the operational semantics of a programming language. Besides the service implementation phase, transformations can be used for the configuration management of services when porting from one service version to another version; PIM to PIM transformation.

The final phase is model-to-code generation (Step 8), which corresponds to the service implementation phase. In case the intermediary PIM to PSM transformation phase is omitted, implementation specific details are hard-coded within the code generator. The implementation is obtained via a semi-automatic process that transforms models (PIM or PSM) to code. Fig. 4 illustrates the definition of the code generator in the form of templates, via the use of a template language. The code

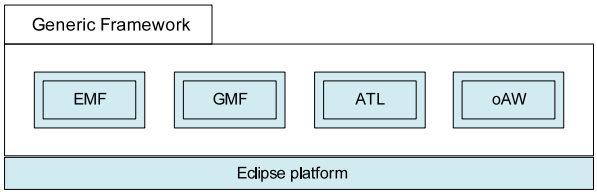


Fig. 5. Generic meta-modelling framework architecture.

generator accepts as an input the domain model defined in accordance to the language and generates the corresponding software application code.

3.2. Generic service creation framework architecture

In order to apply the methodology in practice a generic framework is required. The proposed framework comprises of existing software components integrated together on top of the Eclipse platform. Its extensible component-based architecture facilitates the integration of software components in the form of Eclipse plug-ins. The core components integrated into a common generic framework [6] are namely the *Eclipse Modelling Framework (EMF)* [20], the *Graphical Modelling Framework (GMF)* [24], the *Atlas Transformation Language (ATL)* [25] and *openArchitectureWare (oAW)* [26]. Fig. 5 presents the Eclipse platform as the foundation and the container of the combined modelling frameworks.

The EMF is the core component that interlinks and enables the functionality of the rest of the frameworks. These components are regarded as EMF-based frameworks because they rely on EMF-based metamodels to carry out their individual operations. For instance transformations in the ATL language are defined on the basis of the EMF domain metamodel. Similarly, the definition of template-based code generators is performed according to the EMF domain metamodel. Consequently, EMF acts as the bridge and ensures horizontal integration of the framework's individual software components.

In this work we utilise the capabilities of the generic framework to define the Context Modelling Language (CML) and generate in a semi-automatic manner its supporting Context Modelling Framework (CMF). The CMF is the new software component generated as an Eclipse plug-in, which is integrated with the main frameworks to deliver the Context-Aware Pervasive service creation framework (CA-PSCF). It provides its own modelling and validation software capabilities but additionally makes use of the existing components' capabilities to execute the phases of model-to-model transformation and model-to-code generation.

4. Context-aware pervasive service creation framework (CA-PSCF)

4.1. Requirements analysis

Context-awareness refers to the capability of devices to detect changes in context information and react accordingly to adapt the service behaviour. The primary context categories we acknowledge and build upon are explicitly stated by Dey and Abowd in [2]. These are namely the *Identity, Time, Location and Activity*category types. Apart from these we introduce the *Preference* category type. This is of particular importance since it depicts different service behaviours in accordance to individual user preferences. For example a specific user might prefer to contact a particular friend by sending a simple SMS message and another friend via an email message. Therefore, a distinct behaviour should be realised by the service for each of the two recorded cases. Furthermore, recording users' preferences and providing a rating to individual preferences enables the service to undertake the appropriate actions on behalf of the user; with increased probability of correctness.

In addition to the primary category types, secondary context categories are introduced that denote explicit information of the context-aware service. For instance a restaurant's details, *e.g. name, address,* describe a secondary context category that is explicit to a particular service. Furthermore, primary and secondary categories facilitate the derivation of simple pieces of context information [2]. For instance, if the *Identity* of the person is known we can realise primitive context information about that individual such as his *name, email, gender etc.*

Fig. 6 illustrates the different abstraction levels of context information. Each particular entity *(i.e. Person, Device)* is described by different context information. At the top abstraction level context represents information objects *(i.e. Identity, Location)*, which are composed by *complex* datatypes such as *Profile, Address and Message*; at the second level. These *complex* datatypes are composed of primitive datatypes such as *Integers, Strings, and Booleans* residing at the third abstraction level. For instance the *Identity* of a person is defined in the form of a *Profile* complex datatype, which is composed by the *forename* and the *surname String* primitive datatypes.

In conventional services information can be managed in a common way since it is mainly acquired from a single input source; profiled by the user. On the contrary, in context-aware services information needs to be managed differently since it is obtained from diverse input sources (e.g. sensors, repositories). Consequently, a classification of context sources is essential in order to distinguish and manage context information accordingly. In this work we follow the classification defined in [9], which segregates context types as follows:

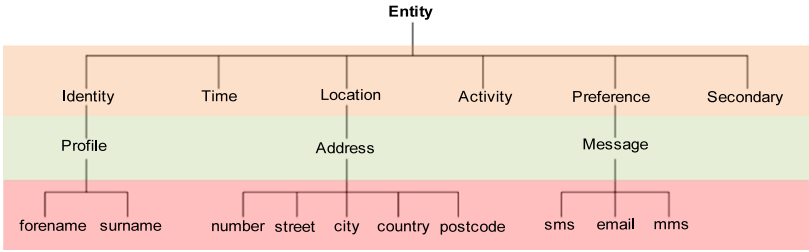


Fig. 6. Context information abstraction levels.

- *Static*: information of high persistence (e.g. date of birth).
- *Profiled*: user-supplied information.
- *Sensed*: information captured from sensors.
- *Derived*: derived on the basis of other context information.

Context validity can be determined by the persistence of context information. The persistence property defines the frequency with which context information are subject to change. Conventionally, static context sources disclose a permanent correlation between the entity and its associated context information. Profiled sources reveal infrequent (seldom) context changes since information remain fixed over long periods of time; unless altered by the user. Conversely, sensed and derived context sources denote information that change frequently and are extremely unstable (frequent or volatile). This is because context obtained from sensors or derived from other information is highly unpredictable.

Moreover, context validity can be determined via the use of temporal constraints. These are defined either as comparative or absolute time constraints. A comparative constraint establishes an expiration time for information obtained from a context source. Alternatively, absolute temporal constraints designate both the starting and expiring interval values. Temporal constraints are of prime importance since they designate when information becomes outdated.

Context quality is also considered on the basis of diverse context sources. Static information contained within a repository is usually of superior quality than profiled information input by the user; assuming correct information is defined. This is because a user might neglect or forget to update the information. Correspondingly, sensed context is of inferior quality than the static and profiled ones. This is because context acquired from sensors can be inaccurate or erroneous and consequently unreliable. Furthermore, context can be derived on the basis of other information. For instance, potential restaurants for dining can be derived from the food preference of the person. Therefore, the quality of derived information relies both on the quality of other context information (e.g. sensed) and the derivation rule.

Context privacy is another major issue that needs to be addressed in order to achieve the acceptance of context-aware services by users. Different context information requires to be treated differently in terms of privacy. For instance a user might want to keep its profile information accessible to all users of the service. Opposed to this, some context information such as credit card details must not be accessible to other users. Hence, different permissions should be set on each context source to restrict accordingly the access to context information.

Apart from categories and input sources, contextual situations are crucial for the modelling and creation of context-aware services. Context situations represent the conditions that must be valid in order to trigger an explicit behaviour in the case a context event occurs. For instance the change of context information related to a person (e.g. *location*) results in the alteration of the person's situation (e.g. *in office or at a meeting*). If the person is currently at a meeting its mobile phone device must be set automatically to silent mode in accordance to the occurring contextual situation. Therefore, it can be realised that contextual situations are imperative for modelling the rules that guide explicit behaviours and are valuable to the interaction between the user and the service.

The categorisation, classification and the situations definition facilitate the mapping of the context model and the generation of the corresponding implementation. Primarily categorisation supports the generation of information classes in accordance to the requirements of each distinct category. Moreover, context classification aids the generation of different context management classes for handling diverse context information as required in a distributed environment. Also the definition of contextual situations supports the generation of distinct situation classes that handle explicit service behaviours. Via the generated classes the developer can query, obtain and distribute context information to the pervasive service to achieve its adaptation. Hence, the generated implementation aids and simplifies the pervasive service creation process.

4.2. The proposed Context Modelling Framework: CMF

The requirements analysis provides the capability to identify the essential artefacts for the definition of the CML and the generation of the CMF. Fig. 7 presents Steps 1–4 (Fig. 1) of the model-driven development process and illustrates how these are accomplished using the software tools and the functionality of the generic framework.

According to Step 1 (Fig. 1) of the process the identified semantics must be mapped to language constructs. This denotes in particular the definition of the CML in the form of a context metamodel. The Meta Object Facility (MOF) [27] formal specification is the foundation of our meta-modelling approach. Although MOF conceptually facilitates the metamodel

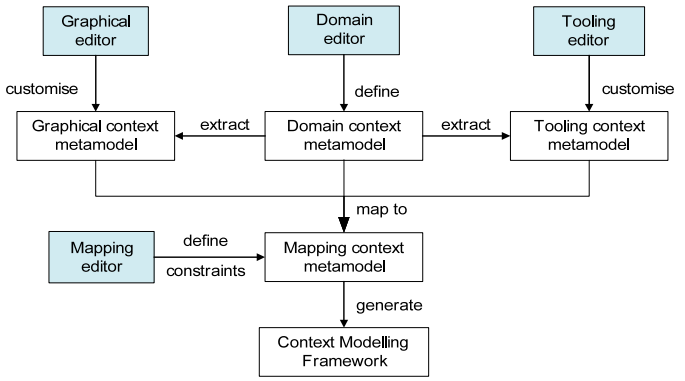


Fig. 7. Creating the Context Modelling Framework.

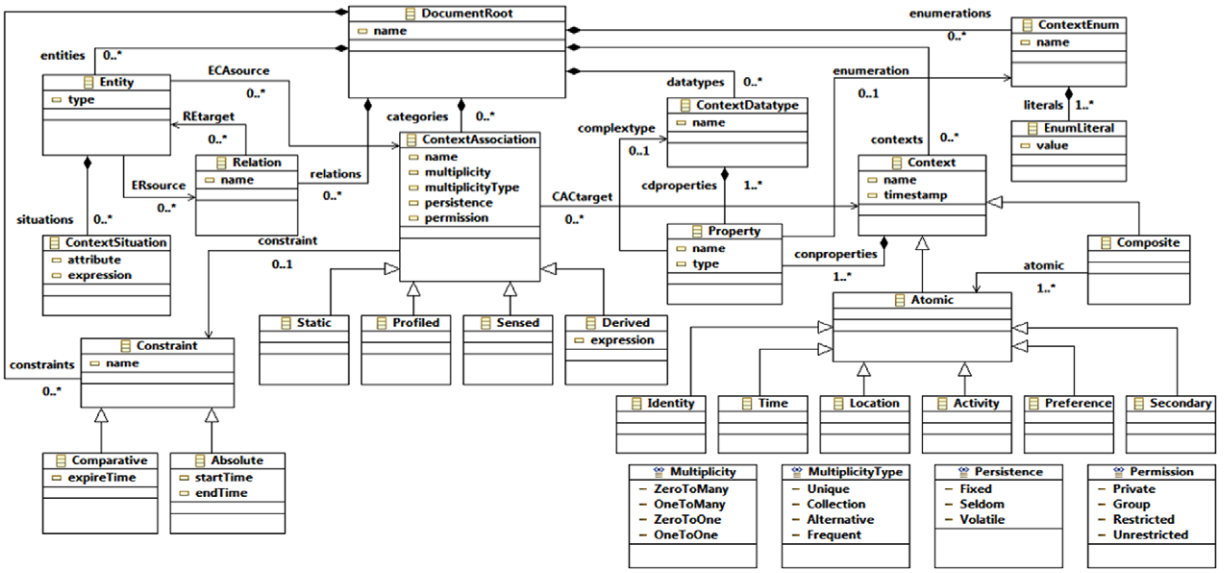


Fig. 8. Context modelling language.

definition, it does not contribute any software tools to support the metamodel definition and the concrete syntax definition. Hence, the Ecore meta-modelling language of the EMF component is used for the definition of the context metamodel. The motive for selecting EMF as the core of the generic framework is its one-to-one mapping with MOF [28,29]. Furthermore, both EMF and GMF influenced heavily the MOF 2.0 specification towards the critical direction of tools integration to achieve the overall objective of model-driven development. The context metamodel definition can be performed either using the EMF or the GMF based editor. The GMF domain editor is preferred, as shown in Fig. 7, since it facilitates the metamodel definition using a comprehensible graphical notation; on the basis of the ECore language.

Fig. 8 illustrates the defined metamodel, which describes context elements (e.g. *Entity*), properties (e.g. *multiplicity*) and relationships (e.g. *ECAsource*). The metamodel includes the artefacts required for the definition of a context model; an instance of the metamodel. The *DocumentRoot* metaclass is the container of the elements of the context model. Its aggregation associations (e.g. *contexts*) define the containment relationships with the rest of the elements.

The *Entity* metaclass represents objects that can be associated via the *ContextAssociation* metaclass to a variety of context information. The object type (*e.g. Person, Device*) can be defined via the *type* property designated in the metaclass. Entities can contain one or more situations defined by the *ContextSituation*metaclass. Contextual situations are defined via the *attribute* and *expression* properties of the *ContextSituation* metaclass. The *attribute* property describes a *Boolean* variable. In accordance to the expression defined as an OCL constraint the variable value is evaluated either to *true or false*, which denotes the occurrence or not of the contextual situation. Finally an entity can be associated to other entities via the *Relation* metaclass and the *ERsource* and *Retarget* associations. Conceptually the *Relation* metaclass denotes a relationship between two entities and it is interpreted as a specific behaviour bound to the two entities; e.g. *Person* → *owns* → *Device*.

ContextAssociation is defined as an abstract metaclass from which the *Static, Profiled, Sensed and Derived* metaclasses inherit their properties. The first property denotes the name of the context source. The *multiplicity* property designates

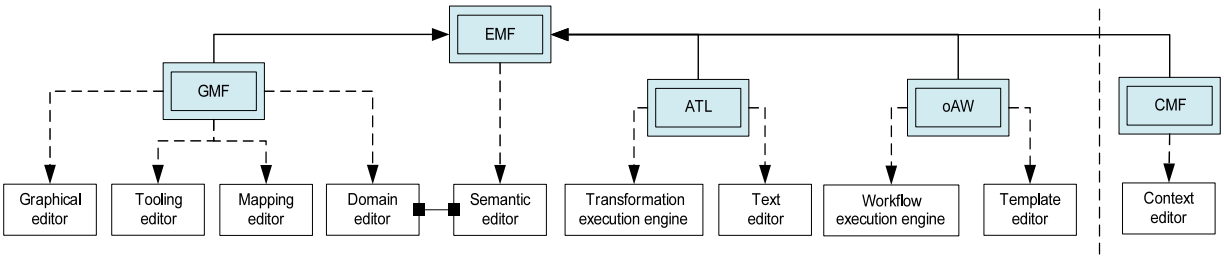


Fig. 9. CA-PSCF components and tools.

a collection of information and can be assigned the values defined by the *Multiplicity* enumeration. Moreover, the *multiplicityType* property determines the number of simultaneous valid occurrences of context information and obtains its values from the *MultiplicityType* enumeration. The *persistence* property is bound to the *Persistence* enumeration, which describes the frequency with which context is subject to change. Finally the *permission* property discloses the access restrictions imposed upon context information. The values defined via the *Permission* enumeration show the access restrictions that can be imposed on context information in order to safeguard the privacy of the user.

Additionally the *Derived* metaclass includes an *expression* property, which is used to define the dependence of context information on other context information via an OCL constraint. Furthermore each context source is associated via the *constraint* relationship to the abstract *Constraint* metaclass, which defines a time constraint for the specific context source. This temporal constraint can be defined either by establishing a valid expiration time for the context type or by setting an absolute time interval designating both the starting and expiration time.

Context information is defined as an instance of the *Context* abstract metaclass and can be either *Atomic* or *Composite* according to the inheritance relationship. The *Atomic* context contains simple properties, which represent the lowest level of context information. The *Composite* context contains both simple properties and atomic context. Moreover, the *Atomic* context is also defined as an abstract metaclass since it is extended by the context categories. Properties can be defined as primitive datatypes or even complex datatypes via the *complextype* association. Additionally a property can be associated to an enumeration (instead of a context datatype) as depicted by the *enumeration* relationship.

Although the metamodel definition includes the elements, relationships and properties required to define a context model, invalid metamodel instances can be still defined by the designer. Consequently, domain rules need to be identified and imposed onto the modelling language definition; according to Step 2 of the process (Fig. 1). A portion of the defined OCL constraints is illustrated next to showcase their importance.

Domain Element Target: Entity::EClass

i. Entity.allInstances()->forAll(e1, e2 | e1 <> e2 implies e1.type <> e2.type)
ii. Entity.allInstances()->forAll(e1, e2 | e1<>e2 implies e1.ERsource <> e2.ERsource)

Domain Element Target: Context::EClass

i. self.conproperties->forAll(p: Property | p.type = 'char' or p.type = 'String' or p.type = 'boolean' or p.type = 'Integer' or p.type = 'double' or p.type = 'float' or p.type = 'long' or p.type = 'short' or p.type = p.complextype.name or p.type = p.enumeration.name)
ii. self.conproperties->forAll(p1: Property, p2: Property | p1 <> p2 implies p1.name <> p2.name)

The first group of OCL expressions targets the *Entity* metaclass. The primary constraint restricts the model definition so that duplicate entity instances cannot be defined. In addition the second rule applied onto the metaclass prohibits the definition of cyclic relationships between entities. Following, the second group of OCL expressions targets the *Context* metaclass. The first constraint restricts the context model definition and ensures that the *type* property of the *Property* metaclass is set to one of the following: (i) primitive datatype, (ii) complex datatype, (iii) enumeration. Moreover, the second rule complements the first since it prevents the definition of the same *name* property for distinct context properties. Hence, the definition of context properties is restricted via these two constraints to valid *Property* instances.

The definition of constraints provides a coherent abstract syntax for the CML. Subsequently, according to Step 3 of the process (Fig. 1) the metamodel must be mapped to a corresponding concrete syntax. Fig. 7 illustrates the automatic interpretation of the metamodel and the extraction of the graphical and tooling context metamodels; concrete syntax. The graphical metamodel defines the graphical components that will be used to define visually the context model within the modelling editor. Likewise, the tooling metamodel defines the palette components that enable the drag-and-drop functionality of the modelling editor. Both metamodels can be customised using the graphical and tooling editors to optimise the appearance of the CMF.

Finally, as shown in Fig. 7, the three distinct metamodels are merged automatically into a mapping metamodel that enables the generation of the CMF in the form of an Eclipse plug-in. The generation of the CMF is driven by the existing EMF Java Emitter Templates (JET) and the GMF XPand templates, which transform the mapping metamodel to the required implementation. The CMF comprises of the Context Modelling Language as its core constituent and a context modelling editor with drag and drop capabilities for the definition and validation of context models. Fig. 9 illustrates the architecture of the Context-Aware Pervasive Service Creation Framework, which comprises of the generated CMF component integrated

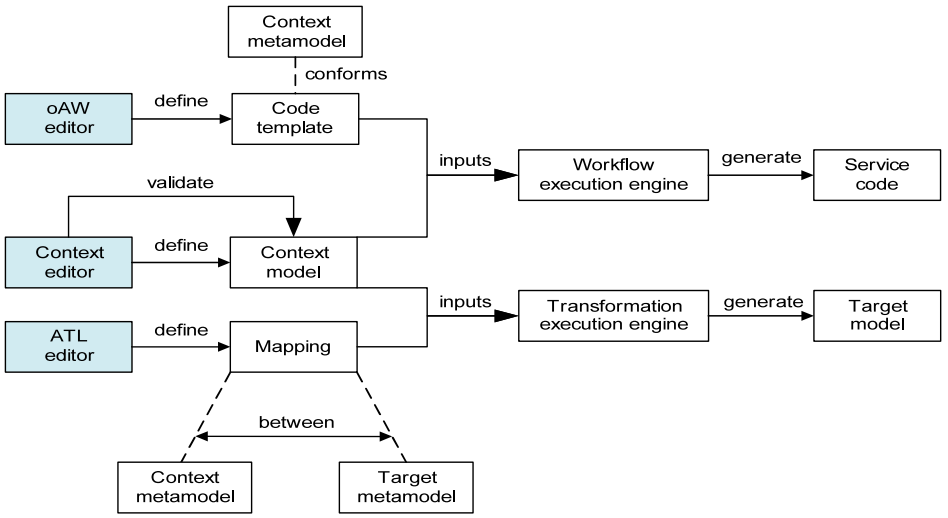


Fig. 10. Context model definition, validation, transformation and code generation.

with the core components of the generic framework. Moreover, the software tools of each component that support the pervasive service creation process are also presented in the figure.

4.3. Context model definition and validation

The definition of the context model is of prime importance for the subsequent steps of the pervasive service creation process. Therefore the designer needs to ensure that the context model defined is precise and coherent. Fig. 10 presents the context model as the key input for the model-to-model transformation and model-to-code generation phases. Initially via the use of the context editor the model definition is performed (Step 5— Fig. 2). Moreover, the modelling editor restricts the designer from defining an invalid model and supports the validation of the context model in accordance to the imposed OCL constraints (Step 6— Fig. 2). The validation reveals any inconsistencies detected and presents accordingly descriptive messages to the designer in order to revise the model definition.

4.4. Context model-to-model transformation

The model-to-model transformation phase is performed using the ATL component of the integrated framework (Step 7–Fig. 3). The core constituent of the component is the Atlas Transformation Language that allows writing transformations in the form of a mapping. Fig. 10 illustrates that via the use of the ATL editor the mapping is defined between the CML semantics and the target metamodel. The context model and the mapping are accepted as the inputs of the transformation execution engine that drives the translation of the context model, e.g. to a relational model. Transformations are important since they provide the capability to translate the context model to a platform specific model to ease the code generation phase. Another form of transformation supports porting context models to extended or improved versions. This is performed by defining a mapping between the current context metamodel and an extended version of the context metamodel. The procedure is known as software configuration management and supports the service evolution.

4.5. Context model-to-code generation

The model-to-code generation phase aims to complement and simplify the implementation of pervasive services (Step 8–Fig. 4). This is performed by mapping context models using a template-based approach to a corresponding programming language and automatically generating the implementation. Although the implementation is not generated fully, a considerable portion of the pervasive service implementation is obtained from the context models. This includes information classes, context management classes and contextual situations classes. In this work we assume that low-level architectural components (e.g. widgets, interpreters) for acquiring, processing and distributing raw data from sensors either exist or require to be implemented manually. Moreover, graphical user interfaces and complex computations must be also implemented manually.

Model-to-code generation is realised via the oAW component of the context-aware pervasive service creation framework. Fig. 10 illustrates the use of the oAW editor for the definition of code templates in accordance to the component's template language. The language provides the capability to define advanced code generators for transforming context models to any implementation technology. As can be realised from the figure, the code generators are defined on the basis of the context

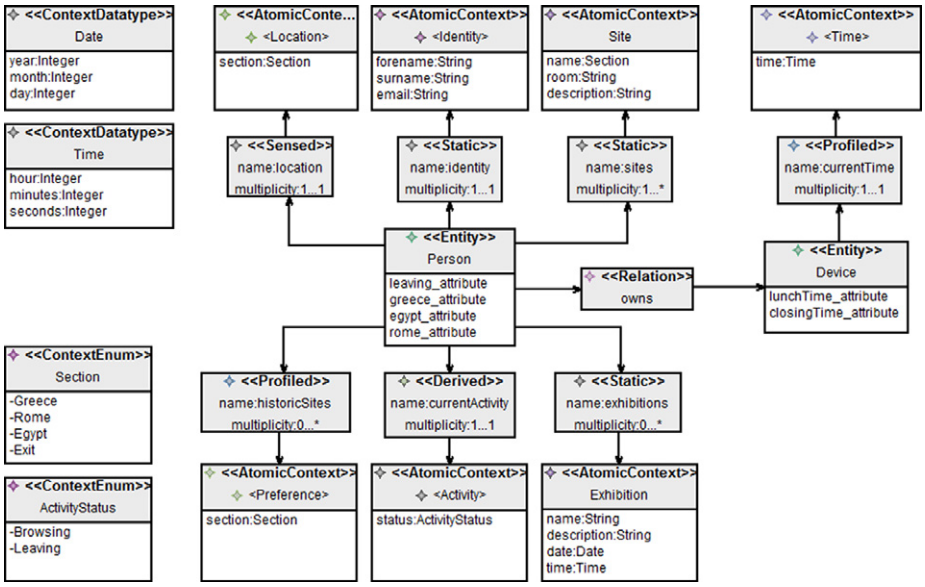


Fig. 11. Pervasive museum service context model.

metamodel definition. This means in particular that the artefacts of the CML are mapped to the operational semantics of the corresponding programming language. Subsequently the context model and the code templates are accepted as inputs of the workflow execution engine, which drives the transformation of models to service implementation code.

5. Case study: Creation of a museum tourist guiding service

5.1. Museum context model definition and validation

The case study detailed in this section presents the execution of Steps 5–8 (Figs. 2–4) of the model-driven development process for the creation of the pervasive museum service. The museum guiding service aims to ease the visitors' touring experience by providing information concerning historic sites, exhibitions and facilities available in the museum. This information is delivered to the user either due to the occurrence of a contextual situation or because the user has explicitly requested the information. For instance, when a user enters a museum virtual zone a proximity sensor detects his presence. The occurrence of this contextual situation denotes that the user should be presented with information on historic sites available within this virtual zone.

Fig. 11 presents the context model designed via the use of the CMF, which defines the entities, situations, context sources and context information required for the creation of the pervasive museum service. The core element of the context model is the *Person* entity that identifies any particular user of the service. Each person is associated to relevant context information via the context source elements, which are defined as instances of the *ContextAssociation* metaclass. For instance each user is associated to the *Identity* context information, which defines a simple user profile.

The *identity ContextAssociation* denotes the necessary properties that characterise the context source from which profile information can be obtained. Primarily the type of the *identity* context source is termed as *Static*. This designates that the user's profile information is stored within a context repository. Moreover the persistence of this profile information is set as *fixed* and the *multiplicity* property is set as 1 . . . 1. The *persistence* property denotes that profile information remains unaltered for large periods of time. In addition the *multiplicity* property designates that only a single profile exists for each user. Furthermore, the *multiplicityType* property is defined as *unique* and the *permission* property is set as *private*. These properties determine correspondingly that a distinct profile exists for each user and only the user can access this context information. The profile properties are defined as *String* primitive datatypes and are named accordingly as *forename, surname and email.*

The *sites* and *exhibitions* associations denote two additional static sources defined in the context model. These designate static repositories that contain correspondingly information on historic sites of the museum and exhibitions taking place at the museum within the current calendar month. Each historic site is defined as a *Site* secondary context, which comprises of the *name, room* and *description* properties. Both the room and description properties are defined as *String* primitive datatypes. In contrast the name property is defined as an enumeration and can be assigned the literal values defined by the *Section* enumeration (ContextEnum). Furthermore, the *Exhibition* secondary context comprises the *name* and *description String* primitive datatypes and the *date* and *time* complex context datatypes (*ContextDatatype*).

Apart from the static context sources, sensed and profiled sources are defined in the context model. The *location* source depicts the association to the *Location* context, which describes the current position of the user within the museum. For

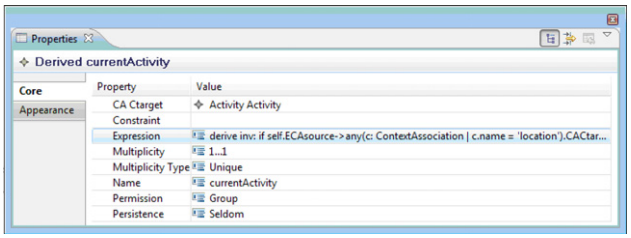


Fig. 12. Activity context association properties view.

this example case study we have separated the museum premises into four virtual zones, which are defined by the *Section* context enumeration. These virtual zones define the possible locations of the user within the museum, which are obtained and processed via the use of proximity sensors. Moreover, the *historicSites* profiled association determines a context source that obtains input information directly from the users. This input context information denotes preferences on historic sites of the museum, which are of particular interest to the user. These preferences are defined via the *section* property of the *Preference* context, which derives its values also on the basis of the *Section* enumeration.

The example model includes also the *currentActivity* context source, which determines the derived *Activity* context information on the basis of the *Location* context. This denotes in particular that the current activity of the user is directly related to his current location. The derivation rule is expressed in the model as an OCL constraint, which is defined using the *expression* property of the context association. In specific the OCL expression presented next defines the condition for deriving the *status* context information of the user, which depicts his current activity derived from the relevant *section* context information. The value of the *status* property is derived by evaluating the conditional part of the logical expression. In the case the user is located at the "*Exit*" of the museum the expression returns the result "*Leaving*", which denotes that the user is currently leaving the museum premises.

context Entity
derive inv: **if self**.ECAsource->any(c: ContextAssociation | c.name = 'location').CACtarget.conproperties->any(p: Property | p.name = 'section').enumeration.literals->any(l: EnumLiteral | l.value = 'Exit').value = 'Exit'
then self.ECAsource->any(c: ContextAssociation | c.name = 'currentActivity').CACtarget.conproperties->any(p: Property | p.name = 'status').enumeration.literals->any(l: EnumLiteral | l.value = 'Leaving').value **else** '' **endif**

Moreover, supplementary OCL expressions are defined in the context model for the three virtual zones *(e.g. Greece, Rome)*, to derive accordingly the current activity status of the user. In the example case study the presence of the user in any of the virtual zones designates that he is still *"Browsing"* the museum historic sites. The OCL constraint illustrated next presents the condition that allows deriving the activity status of the user, when he is located at the historic site of *"Greece"*. In contrast to the aforementioned constraint the evaluation of the logical expression returns the (derived) value *"Browsing"*, which denotes the presence of the user in one of historic sites; *i.e. Greece*. Furthermore, the derivation rules defined for the other two virtual zones are analogous to the constraint designated below. Note that the OCL constraints that characterise the *status* derived context are defined in the form of textual constraints. Consequently the specified constraints are evaluated merely using the OCL engine of the EMF component; *i.e. Interactive OCL console*. The utilisation of the EMF-based implementation of the OMG OCL specification provides the capability to define well-formed invariant constraints, derived attribute and reference constraints and operation constraints.

context Entity
derive inv: **if self**.ECAsource->any(c: ContextAssociation | c.name = 'location').CACtarget.conproperties->any(p: Property | p.name = 'section') .enumeration.literals->any(l: EnumLiteral | l.value = 'Greece').value = 'Greece'
then self.ECAsource->any(c: ContextAssociation | c.name = 'currentActivity').CACtarget.conproperties->any(p: Property | p.name = 'status') .enumeration.literals->any(l: EnumLiteral | l.value = 'Browsing').value **else** '' **endif**

Fig. 12 illustrates the properties view of the *currentActivity* derived source, which comprises of the defined properties that guide the generation of the required functionality (implementation). The properties view defines the target of the context source, which is the *Activity* context information. Accordingly the multiplicity is set to 1…1 and the multiplicityType is set as *Unique* to denote that a person can be engaged in one distinct activity at any given time. Moreover the permission property is set to *Group,* something that defines that only a group of people can access this context information. For instance the security personnel of the museum might require having access and being able to identify the current activity (or location) of the user. The persistence of the source is defined as *Seldom* to depict the infrequent change of the activity context information; user is either *Browsing* or *Leaving*. Finally the expression property defines the context derivation rules that can be validated for consistency, as aforementioned, using the Interactive OCL console of the CA-PSCF.

The final context source depicted in the model associates the user's *Device* to the corresponding *Time* context information. This information is profiled by the user on its own device (e.g. mobile, laptop) and can be acquired and managed in accordance to the properties defined for the *currentTime* context source. The set of context associations introduced in the model and their individual properties, realise the main prerequisite to distinguish between diverse classes of context information and manage this information differently.

In addition to entities, sources and context information the model comprises of contextual situations, which are defined for each entity. Contextual situations designate in essence the necessary condition(s) that must be valid in order to undertake a corresponding action. One contextual situation defined in the context model describes the following: "As soon as the user leaves the museum premises details of forthcoming exhibition tours within the current calendar month should be presented to the user". This condition is expressed by the OCL constraint illustrated next, which describes a contextual situation defined for the *Person* entity. The evaluation of the logical expression determines accordingly the occurrence or not of the behaviour that is directly associated to this particular contextual situation. Note that the specified constraint defines the contextual condition and does not describe the actual action or event that must be triggered as a result.

```
context Entity
init inv: self.situations->any(c: ContextSituation | c.attribute = 'leaving_attribute').attribute > derive inv: if self.
ECAsource->any(c: ContextAssociation | c.name = 'currentActivity').CACtarget.conproperties->any(p: Property |
p.name = 'status').enumeration.literals->any(l: EnumLiteral | l.value = 'Leaving').value = 'Leaving' then true else false
endif
```

In particular the OCL expression defines that if the person activity status changes from *Browsing* to *Leaving* this denotes that the user is currently leaving the museum. Therefore, in accordance to the evaluated logical expression the value of the *Boolean* variable *leaving_attribute* becomes true. Consequently, monitoring the state of the attribute using different instances of the contextual situation provides the capability to detect the context event and react accordingly. In this case the action triggered as a result of the change in the context condition will present to the user the exhibition tours within the current calendar month.

An additional contextual situation expressed in the model depicts the following: "When the user enters the historic site of *"Egypt"*, he should be presented with information concerning this particular site". The subsequent constraint describes this location-specific condition, which allows detecting the context change and reacting accordingly, so as to present to the user the necessary historical information. Similarly, each situation attribute defined in the context model is accompanied by an OCL expression, which describes every contextual situation that must be depicted in the model.

```
context Entity
init inv: self.situations->any(c: ContextSituation | c.attribute = 'egypt_attribute').attribute > derive inv: if self.
ECAsource -> any(c: ContextAssociation | c.name = 'location').CACtarget.conproperties->any(p: Property | p.name =
'section') .enumeration.literals->any(l: EnumLiteral | l.value = 'Egypt').value = 'Egypt' then true else false endif
```

Following the model definition phase, the validation of the museum context model is performed using the CMF capabilities in accordance to the imposed metamodel level constraints. This prevents the developer from attempting to transform or generate implementations out of erroneous context model definitions. In the case that an inconsistency is detected in the context model a corresponding error message is displayed suggesting possible resolutions using an informative description. Subsequently, the designer can undertake the necessary actions to rectify the problems discovered in the model definition.

5.2. Pervasive museum service implementation and evaluation

The implementation phase is carried out primarily by means of context model-to-code generation. For this purpose we have defined the mapping between the context metamodel and the operational semantics of two programming languages. The mapping was defined in the form of code templates to facilitate the transformation of context models (e.g. museum context model) to the Java and J2ME implementation technologies.

Appendix A illustrates an extract of the template mapping defined for transforming context associations to J2ME implementation code. From the mapping (lines 6–8), we can observe that for each context source defined in the context model a corresponding class is being generated. Furthermore, the conditional statements defined at lines 15 and 17 drive the generation of the required implementation in accordance to the multiplicity property depicted in the context source. Another important section of the mapping is presented in lines 10–13 where each property of the context source is mapped accordingly to a corresponding class variable. Accordingly, utility functions are defined that allow accessing these variables from other classes to facilitate and simplify the implementation of the required functionality.

Appendix B illustrates the utility functions defined for the association properties, using the extension language of the oAW component. The extension language facilitates the definition of rich libraries of utility functions, which can be accessed within code templates to aid the code generation process. These helper functions are imported in the code template definition via the *Extensions* statement defined at line 1 and accessed via the ≪ *ecas.SourceHelperFunctions*() ≫ statement defined at line 19 of Appendix A.

The implementation generated from the context model eradicates the requirement to manually implement repetitious code such as information and context management classes [9]. This simplifies and enables the rapid creation of context-aware services. Despite that fact, complex computations and graphical user interfaces require to be manually implemented by extending or modifying the generated service code. Table 2 demonstrates an evaluation of the developed pervasive museum service according to selected software metrics. The analysis results are obtained via the use of the CCCC analysis tool [30]. The selected metrics for the evaluation are: (i) Lines of Code (LOC) and (ii) McCabe's cyclomatic complexity [31]. The analysis shows the effectiveness of the approach in minimising the cost and effort required to implement the pervasive service.

Table 2
Pervasive museum service evaluation.

Implementation	Metric	Generated service code		Overall service code	
		Overall	Per module	Overall	Per module
J2ME	Number of modules	40	–	53	–
	Lines of code	1768	44.200	2330	43.962
	Cyclomatic number	128	3.200	181	3.415
	Lines of comment	560	14.000	648	12.226
Java	Number of modules	33	–	53	–
	Lines of code	1282	38.848	1859	35.075
	Cyclomatic number	49	1.485	101	1.906
	Lines of comment	543	16.455	564	10.642

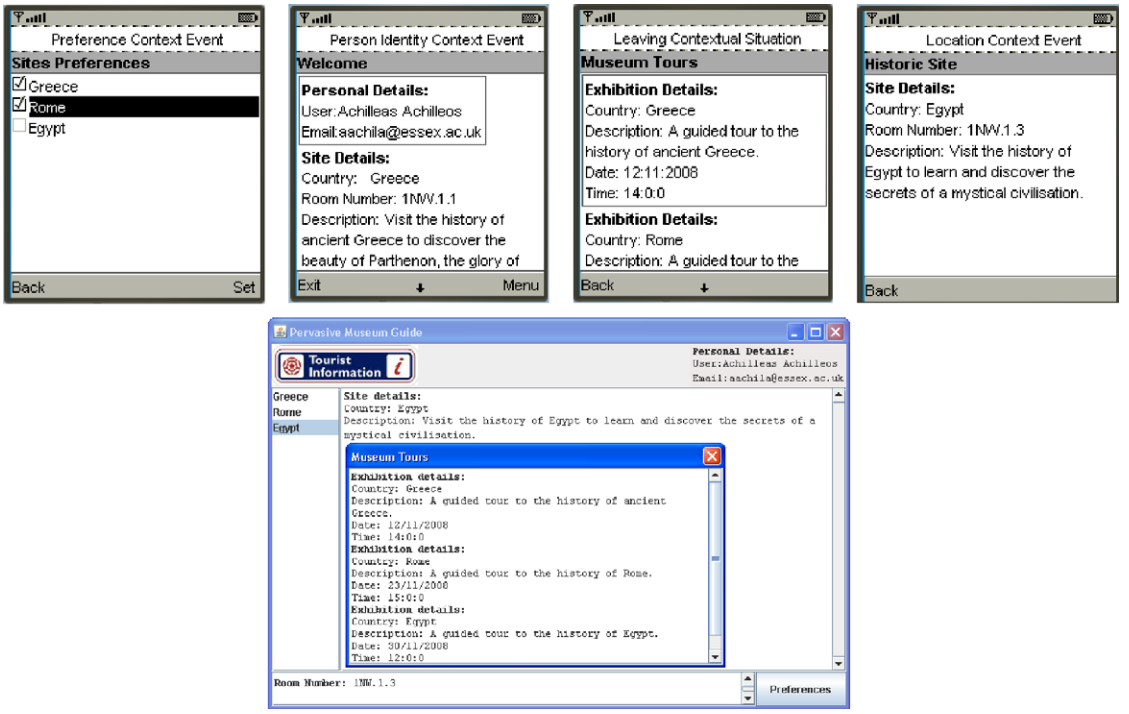


Fig. 13. Pervasive museum service running on a J2ME device emulator.

Table 2 illustrates the analysis results for both the J2ME and Java implementation of the museum context-aware service. Primarily, the LOC metric shows the number of non-comment and non-blank lines of code. From the table it is calculated that 75.879% of the J2ME service code and 68.96% of the Java service code is generated from the context model. The percentages calculated in this case study provide the capability to derive the necessary conclusions but do not serve in any case as an explicit baseline for future case studies. These analysis results indicate simply that the effort required for the implementation of the pervasive museum guide service has been significantly reduced.

Code complexity is another valuable metric, which denotes the degree of code understandability and indicates the code amenability to modification. Moreover it is a dominant indicator of the code testability [31]. From the table it is clear that the complexity of the generated code (indicated by the cyclomatic number) is lower than that of the overall service code. This indicates that the generated code can be easily modified and be subjected to testing. Furthermore, it is realised that the mapping can be optimised further to decrease the generated service code complexity. Conversely, it is very difficult to achieve optimisation and reduce the code complexity when manual implementation is involved.

Fig. 13 illustrates the pervasive museum service running on a J2ME device emulator and a Java enabled laptop device. The screen capture on the top left of the figure shows the historic sites preference selection list. In accordance to the user's preferences the service retrieves and loads the appropriate information on historic sites, as illustrated onto the second screen capture. Following, onto the next screen capture we can observe the occurrence of the leaving contextual situation, which causes a list of upcoming exhibitions tours to be displayed to the user. Moreover, we have the occurrence of the

location contextual situation, which signifies that the user has entered the virtual zone of Egypt. Consequently, the user is presented with information on historic sites located in this virtual zone, as illustrated onto the fourth screen capture. Finally, the screenshot shown at the bottom of the figure presents respectively the occurrence of the leaving contextual situation, while the service is running on the Java enabled laptop device.

6. Conclusions and future work

In this paper we proposed a model-driven development process that is purely based on the MDA paradigm and supports exclusively every phase of the service creation process. On the basis of the approach we have defined the Context Modelling Language and generated the Context Modelling Framework. The CMF was then integrated into the generic service creation framework to deliver the CA-PSCF and simplify the design, validation and implementation of pervasive services.

The usage of the CA-PSCF enables the designer to define context models at an abstract level using a comprehensible visual representation. Furthermore, it supports the validation of context models to determine invalid definitions and rectify them prior to the implementation phase. Subsequently from these context models a significant part of the implementation can be generated, reducing in overall the effort and cost required for the creation of the pervasive service. In this work the effectiveness of the framework has been showcased via the creation of the pervasive museum service and evaluated on the basis of selected software metrics.

The definition of proactive rules for modelling context-aware behaviours is also important and therefore has been considered by a dynamic Petri net model introduced in [32]. The Petri net model facilitates the definition of proactive actions and events that depict the overall pervasive service behaviour. Moreover, the dynamic model supports the formal validation and implementation of the pervasive service. The work presented in this paper focuses though on the context modelling domain, which tackles the structural and more static part of pervasive service creation. This structural part involves the definition of context categories, context sources (i.e. sensed, derived), temporal constraints and contextual situations (i.e. conditions).

Our aim as part of future work is to utilise or develop an OCL generator to facilitate the transformation of the textual constraints to implementation code. Furthermore, we intend to carry out additional case studies that will aid in the identification of any further requirements that must be expressed in the context modelling language. In addition supplementary case studies will assist in the optimisation of the existing context modelling and implementation capabilities of the CMF. This will provide the capability to enhance both the CML and the CA-PSCF for the benefit of pervasive service creation.

Appendix A

```
1:  «EXTENSION extensions::functions»
2:  «DEFINE Root FOR cml::DocumentRoot»
3:  «EXPAND Entity FOREACH entities»
4:  «ENDDEFINE»
5:  «DEFINE Entity FOR cml::Entity»
6:  «FOREACH this.ECAsource AS ecas-»
7:  «FILE ecas.ext1()+".java"»
8:  public class «ecas.ext1()»{
9:  private static final String atomic_context = "« this.ext2()+""+ecas.CACtarget.first().ext3()»";
10: private static final String multiplicity = "«ecas.multiplicity»";
11: private static final String multiplicityType = "«ecas.multiplicityType»";
12: private static final String persistence = "«ecas.persistence»";
13: private static final String permission = "«ecas.permission»";
14:
15: «IF ecas.multiplicity == "0...1" || ecas.multiplicity == "1...1"»
16:
17: «ELSEIF ecas.multiplicity == "0...*" || ecas.multiplicity == "1...*"»
18:
19: «ecas.SourceHelperFunctions()»
20:
21: «FOREACH ecas.CACtarget.conproperties AS cp-»
22: public static String «cp.get1()»() {
23: return «cp.name»;  }
24: «ENDFOREACH» }
25: «ENDFILE»
26: «ENDFOREACH»
27: «ENDDEFINE»
```

Appendix B

```
1:  String SourceHelperFunctions(ContextAssociation ca) :
2:  'public static String getMultiplicity() {
3:  return multiplicity; }
4:  public static String getMultiplicityType() {
5:  return multiplicityType;  }
6:  public static String getPersistence() {
7:  return persistence;  }
8:  public static String getPermission() {
9:  return permission;  }';
```

References

[1] K. Yang, S. Ou, M. Azmoodeh, N. Georgalas, Policy-based model-driven engineering of pervasive services and the associated OSS, British Telecom Technical Journal (BTTJ) 23 (3) (2005) 162–174.

[2] A.K. Dey, G.D. Abowd, Towards a better understanding of context and context-awareness, in: Conference on Human Factors in Computing Systems, 2000.

[3] D.X. Adamopoulos, G. Pavlou, C.A Papandreou, Advanced service creation using distributed object technology, IEEE Communications Magazine 40 (3) (2002) 146–154.

[4] R.H. Glitho, F. Khendek, A. De Marco, Creating value added services in internet telephony: An overview and a case study on a high-level service creation environment, IEEE Transactions on System, Man and Cybernetics- Part C: Applications and Reviews 33 (4) (2003) 446–457.

[5] J. Lennox, H. Schulzrinne, Call processing language framework and requirements, RFC 2824, Internet Engineering Task Force (2000) [Online] http://www.ietf.org/rfc/rfc2824.txt.

[6] A. Achilleos, N. Georgalas, K. Yang, An open source domain-specific tools framework to support model driven development of OSS, in: Proc. ECMDA-FA '07, in: Lecture Notes in Computer Science, vol. 4530, Springer–Verlag, Berlin, 2007, pp. 1–16.

[7] A.K. Dey, G.D. Abowd, The context toolkit: Aiding the development of context-aware applications, in: ICSE workshop on software engineering for wearable and pervasive computing, 2000.

[8] G. Chen, D. Kotz, Context aggregation and dissemination in ubiquitous computing systems, in: 4th IEEE Workshop on Mobile Computing Systems and Applications, 2002.

[9] K. Henricksen, J. Indulska, Developing context-aware pervasive computing applications: Models and approach, Pervasive and Mobile Computing 2 (1) (2006) 37–64.

[10] T. Strang, C. Linnhoff-Popien, A context modelling survey, In: UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management, 2004, pp. 34–41.

[11] T. McFadden, K. Henricksen, J. Indulska, Automating context-aware application development, in: UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management, 2004, pp. 90–95.

[12] A. Kleppe, J. Warmer, W. Bast, MDA Explained: The Model Driven Architecture: Practice and Promise, Addison-Wesley, Boston, 2005.

[13] D.S. Frankel, Model Driven Architecture: Applying MDA to Enterprise Computing, Wiley Publishing Inc, Indianapolis, 2003.

[14] Object Management Group (OMG), Model Driven Architecture (MDA) Specification Guide v1.0.1 [Online]. http://www.omg.org/docs/omg/03-06-01.pdf, 2003.

[15] A. Achilleos, K. Yang, N. Georgalas, A model-driven approach to generate service creation environments, in IEEE Globecom 2008, in: Proc. of the IEEE Globecom, 2008, pp. 1–6.

[16] G. Ortiz, B. Bordbar, J. Hernandez, Evaluating the use of AOP and MDA in web service development, in: Proc. of Conference on Internet and Web Applications and Services, 2008, pp. 78–83.

[17] J. Bauer, Identification and modelling of contexts for different information scenarios in air traffic, Diplomarbeit, Faculty of electrical engineering and computer sciences, Technische universitat Berlin, 2003.

[18] C. Simons, G. Wirtz, Modelling context in mobile distributed systems with the UML, Journal of Visual Languages and Computing 18 (2007) 420–439.

[19] Object Management Group (OMG), Object Constraint Language (OCL) Specification version 2.0 [Online]. http://www.omg.org/docs/formal/06-05-01.pdf, 2005.

[20] Eclipse Foundation Inc., Eclipse Modelling Framework (EMF) [Online]. http://www.eclipse.org/modelling/emf/, 2008.

[21] K. Henricksen, J. Indulska, A software engineering framework for context-aware pervasive computing, in: Proc. 2nd IEEE International Conference on Pervasive Computing and Communications, 2004, pp. 77–86.

[22] J. Evermann, Y. Wand, Towards formalizing domain modelling semantics in language syntax, IEEE Transactions on Software Engineering 31 (1) (2005) 21–37.

[23] S.A. Thibault, R. Marlet, C. Consel, Domain specific languages: From design to implementation application to video device drivers generation, IEEE Transactions on Software Engineering 25 (3) (1999) 363–377.

[24] Eclipse foundation inc., Graphical Modelling Framework (GMF) [Online]. http://www.eclipse.org/gmf/, 2008.

[25] INRIA Research Institution, Atlas Transformation Language (ATL) [Online]. http://www.eclipse.org/m2m/atl, 2008.

[26] openArchitectureWare.org, openArchitectureWare (oAW) [Online]. http://www.eclipse.org/gmt/oaw, 2008.

[27] Object Management Group (OMG), Meta Object Facility (MOF) Core Specification version 2.0 [Online]. http://www.omg.org/docs/formal/06-01-01.pdf, 2005.

[28] A. Gerber, K. Raymond, MOF to EMF: There and back again, in: Proc. of the OOPSLA Workshop on Eclipse Technology eXchange, 2003, pp. 60–64.

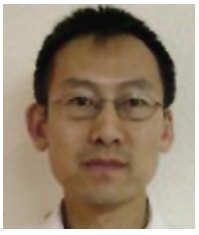
[29] M. Mohamed, M. Romdhani, K. Ghedira, EMF-MOF alignment, in: Proc. 3rd International Conference on Autonomic and Autonomous Systems, 2007, pp. 1–6.

[30] T. Littlefair, An investigation into the use of software code metrics in the industrial software development environment, Ph.D. Thesis, Faculty of Communications, Health and Science, Edith Cowan University, 2001.

[31] P.V. Bhansali, Complexity measurement of data and control flow, ACM SIGSOFT Software Engineering Notes 30 (1) (2005) 1–2.

[32] A. Achilleos, K. Yang, N. Georgalas, M. Azmoodech, Pervasive service creation using a model-driven Petri Net based approach, in: Proc. 3rd IWCMC (2008) 309–304.

Achilleas Achilleos is currently a Ph.D. student in the School of Computer Science and Electronic Engineering at the University of Essex. He received his M.Sc. with distinction from the same department and a B.Sc. with excellence from the Budapest University of Technology and Economics in Hungary. His research interests include model-driven development, pervasive service creation, context-modelling and mobile computing. Currently he is also employed by BT as a research contractor and his research work is co-funded by BT and the UK Engineering and Physical Sciences Research Council (EPSRC). He has recently published his research work in several conference papers, a book chapter and a journal. He served as a referee and a TPC member in several conferences related to his research area. He is a member of the Institute of Electrical and Electronic Engineers (IEEE).



Kun Yang received his Ph.D. from the Department of Electronic & Electrical Engineering of University College London (UCL), UK, and M.Sc. and B.Sc. from the Department of Computer Science of Jilin University, China. He is currently a Reader in the School of Computer Science and Electronic Engineering, University of Essex, UK. Before joining in University of Essex at 2003, he worked at UCL on several European Union research projects such as FAIN, MANTRIP, CONTEXT. His main research interests include heterogeneous wireless networks, fixed mobile convergence, pervasive service engineering, and IP network management. He has published more than 100 peer-reviewed technical papers in journals and major international conference. At Essex, he principally or collaboratively investigates projects such as PANDA, MOSE, HIPnet, PAL, EU project GEYSER, etc. He serves on the editorial boards of both IEEE and non-IEEE journals (such as Wiley Wireless Communications and Mobile Computing, Springer Telecommunication Systems, etc.). He is a Senior Member of IEEE, a Member of IET and ACM.

Nektarios Georgalas holds a Diploma in Electrical and Computer Engineering from the University of Patras, Greece, an MPhil in Computation from University of Manchester (UMIST) and a PhD in Computer Science from the University of London. He joined British Telecom (BT) in 1998 and is now a principal researcher in the company's Centre for Information and Security Systems Research. During his career with BT, he has participated and managed research projects in areas including active networks, market-driven data management systems, policy-based management, distributed information systems, service-oriented architectures and web services. His research is currently focused on product lifecycle management, particularly migration planning and concept-to-market, and rapid service assembly. Nektarios has led numerous international collaborations on the application of model-driven architecture and New Generation Operations Systems and Software (NGOSS) standards in telecoms operational support systems and has both led and contributed to the work of the TeleManagement Forum. He holds five patents, has authored more than 30 papers and has frequently been invited to speak at international conferences.