2012 IEEE 15th International Conference on Computational Science and Engineering

A Framework for Dynamic Validation of Context-Aware Applications

Achilleas P. Achilleos, Georgia M. Kapitsaki and George A. Papadopoulos
Department of Computer Science
University of Cyprus
Nicosia, Cyprus
Email: [achilleas, gkapi, george]@cs.ucy.ac.cy

***Abstract*—The development of context-aware applications is a complex process that involves the tasks of analysis, design, validation and implementation. This process is typically performed using context modelling approaches that consider context-awareness and apply static context model validation, at the modelling level, using Object Constraint Language rules. This paper proposes a framework that combines the Model Driven Engineering paradigm with the Petri Nets formalism to support the development of context-aware applications. In contrast to existing context modelling approaches, the focus is on dynamic validation of context-aware applications using Petri Nets. Dynamic validation complements the static validation of context models and ensures the validity of the operational semantics of context-aware applications. The applicability of the framework presented in this work is exemplified through the development of a context-aware application prototype.**

***Keywords*-model validation; Petri Nets; context-aware applications; context-awareness; model driven engineering**

I. Introduction

Conventional applications are commonly deployed on a specific device and platform to run specific computing tasks. In contrast, context-aware applications have the capability to run anytime, anywhere and on any device with minimal or no user intervention. The advancements in hardware and software technologies contribute towards the development of mobile devices, which include sophisticated computing and communication capabilities. This enables the implementation of complex and adaptive context-aware applications. However, "traditional" development approaches perform validation at the late stage of software testing. In many cases, in the interest of expediting delivery, changes are performed in the implementation and not in the design and documentation. This introduces a discrepancy and impacts the efficiency of the development process.

The aforementioned issue driven research towards context modelling approaches [1]. These techniques deal with the definition of a context model. The context model defines intelligent information and predefined adaptation rules that allow adapting the application's execution logic and thus its interaction with the user. This information and the predefined rules describe the context-awareness characteristic of software applications [2], captured in the context model. Furthermore, context modelling techniques support the validation of context models using static Object Constraint Language (OCL) [3] rules, prior to the generation of the implementation using the context models.

In particular, OCL constraints ensure model integrity by validating the elements, properties and relationships defined in the model. This guarantees the integrity of the model's structure (i.e. typically defined as a class diagram with context information and reasoning rules) but does not tackle the validation of the behaviour of the context-aware application. Hence, a formal modelling approach is necessary that allows defining and validating both the static structure and the dynamic behaviour of the application. This approach ensures also the consistency of the application's execution logic prior to the generation of the implementation from the model.

This work proposes a framework that builds on existing context modelling approaches but addresses also dynamic modelling and validation to support the development of context-aware applications. It combines for the first time, to the best of our knowledge, the Model Driven Architecture (MDA) paradigm [4] with the Petri Nets (PNs) formalism [5] to support dynamic validation of context-aware applications. The integration of MDA with Petri Nets enables the validation of the models structure using static OCL constraints and their dynamic behaviour using the Petri Nets formalism. Moreover, the MDA paradigm facilitates the application's generation from the defined models.

The contribution of this work lies in the use of the Petri Nets formalism, which allows defining the dynamic behaviour of context-aware applications via a suitable representation that enables easy-comprehension and validation of application's execution logic. Existing context modelling techniques fall short in terms of the representation of the dynamic behaviour since the application's execution steps (i.e. context adaptation and reasoning rules) are concealed within elements defined in the context model. In addition, the application's logic is best to be defined by a dynamic modelling formalism (e.g. Petri Nets, statecharts).

The Petri Net based modelling and validation framework is composed of three modelling components. These are: (i) Presentation Modelling Component (PMC), (ii) Context Modelling Component (CMC) and (iii) Petri Net Process Modelling Component (PN-PMC). These components al-

978-0-7695-4914-9/12 $26.00 © 2012 IEEE
DOI 10.1109/ICCSE.2012.79

low defining presentation, context and Petri Net models, which describe respectively the graphical user interfaces (GUIs), the context-awareness and the dynamic behaviour of context-aware applications. In this paper, the focus is on the modelling and dynamic validation capabilities provided by the PN-PMC, since it allows detecting inconsistencies directly in the application specification. In this way errors are found before the automatic transformation to implementation code. The descriptions of PMC and CMC are out of the scope of this work. Interested readers are referred to relevant publications [6], [7].

The paper is structured as follows: Section 2 presents related research work. Section 3 introduces the framework's architecture and describes the development process, with explicit focus on the Petri Net based validation component presented in Section 4. Section 5 showcases the applicability of the framework by modelling, validation and implementation of a context-aware application. Concluding, Section 6 presents results and proposes future directions of this work.

II. Related Work

This work follows the direction of context modelling approaches, which assume network-level context sensing, aggregation and processing functionalities. Context modelling techniques deal with the representation of context types and their relationships, of high-level context abstractions describing real world situations using context information facts, of histories of context information and of uncertainty of context information [1]. This knowledge is then passed from designers to developers, so as to implement the application or apply code generation techniques to produce "skeleton code" that is subsequently extended.

Different techniques [8] have been proposed that use ontologies for context modelling and reasoning. One of the popular techniques is the SOUPA ontology [9] that offers many advantages in terms of modelling, reasoning and handling context at an abstract level. This approach was adopted in the Context Broker Architecture (CoBrA) for supporting context-awareness [10], but also in more recent works [11]. However, ontology models are rather unsuitable for human structuring purposes since their representation restricts communication between designers and developers. In specific, these models are inherently complex for developers that are not familiar with their description logics [1].

The strengths of graphical modelling approaches are certainly on the structure level [1]. The context modelling technique proposed in [12] defines such an approach. It proposes an infrastructure and a framework for modelling (Context Modelling Language, CML), managing and disseminating context information to context-aware applications for adapting their functionality. Although the approach has many advantages, the lack of a widely-used representation and the low-level abstractions introduced via the extension of the Object-Role Modeling (ORM), limit the understanding of context models. A tool support for modelling context using CML can be found in [13]. Since ORM models are more suitable for use in databases, the model is transformed into a relational schema and SQL scripts. This makes the approach highly suitable for the process of context collection. Additionally, the applied model validation is limited (i.e. restricted to static validation by highlighting model errors) and is indicated by the authors as future work.

The approach defined in [14] conveys well to the concept of graphical context modelling since it exploits a widely-used representation; i.e. the Unified Modelling Language (UML). In specific, a Context Modelling Profile (CMP) was developed that defines a domain-specific language using UML stereotypes. The CMP allows modelling information as UML-based context models that describe context-aware applications. This is a key advantage of the approach because it allows developers to use CMP in various UML tools. The shortcoming is the inability to access CMP stereotypes in many UML tools, so as to define constraints and transformations. Thus, the Eclipse Modelling Framework (EMF) is used instead to define model constraints. Moreover, Simons and Wirtz [14] state that model-to-code transformations are considered as future work.

An approach that is highly-interlinked with the MDA paradigm, proposes the use of the Meta-Object Facility (MOF) specification for defining context modelling concepts [15]. Using the MOF specification a metamodel is defined, which describes a domain specific modelling language for context-awareness and provides common understanding of context information. Although the abstract syntax of the language is defined using MOF, the concrete syntax of the language is defined using a UML profile. Hence, as in the case of the CMP, the approach suffers from the inability to access model stereotypes in various UML tools for defining OCL constraints and model transformations.

A key shortcoming of these techniques is that only a subset considers validation, which is limited to static model validation using OCL rules. The development of context-aware applications involves dynamic and adaptive computing tasks (i.e. application's logic), which cannot be clearly defined in a context model represented as a class diagram. Thus, a dynamic modelling language is required that allows defining behaviour. Moreover, such a formalism should enable validation of the application's behaviour. This complements existing validation techniques that validate the structure of context models using OCL rules.

The need for dynamic validation motivated us towards the PNs formalism, through the study of its use in the domain of Web Services (WSs). Dumez et al. state in [16] that static validation is complemented by checking the dynamic structure of WSs defined using a UML extension [17]. In specific, UML-WS models are transformed to high-level PNs. This allows formalising UML and applying mathematical analysis on the models. A similar approach uses the Business Process

Execution Language (BPEL) language to define WSs [18]. The authors state that BPEL can experience inconsistency and incompleteness, when modelling behaviour. Hence, transformation rules are defined that allow transforming, analysing and validating WSs using Synchronisation Nets (i.e. Petri Net extension).

This work addresses dynamic validation by bridging the MDA paradigm with the PNs formalism. MDA provides a model-driven approach that expedites the design, static validation using OCL rules and implementation of context-aware applications. Complementing MDA, PNs provide a formal method that enables definition and validation of the dynamic behaviour of context-aware applications by means of model execution. The applicability of the framework is demonstrated through the design, validation and implementation of a context-aware application.

III. Overview of the Petri Net based Modelling and Validation Framework

The formulation of a consistent development process is a crucial activity since it guides designers and developers in a uniform way. This enables them to fully comprehend their responsibilities and assigned tasks, so as to smoothly accomplish the development of the necessary applications. Moreover, a supporting development environment is important because it provides the necessary software tools (e.g. modelling, code generators) that support actors in the development process. In this work, a model-driven Petri Net based process and a development environment are defined that support the development of context-aware applications.

The modelling components of the development environment support the definition of the presentation, context and Petri Net based modelling languages and the generation of the supporting modelling components (Figure 1). These components have been generated and integrated into the model-driven environment as Eclipse plug-ins. They facilitate the definition of presentation, context and Petri Net models that describe respectively the GUIs, the context information and the dynamic behaviour of context-aware applications. Note that the focus is on the PN-PMC, since it supports integration of the modelling languages and dynamic validation of the application's execution logic.

The environment allows accessing stereotypes of the modelling languages, in order to impose design rules (i.e. OCL constraints) and enforce static model validation. This is performed using the model-driven capabilities of the Eclipse Modelling Framework (EMF) and the Graphical Modelling Framework (GMF). In addition, the Atlas Transformation Language (ATL) and the openArchitectureWare (oAW) provide respectively model-to-model and model-to-text transformation capabilities. These components enable model transformation, so as to support validation of the dynamic behaviour and code generation. In overall, the environment includes all necessary software tools that support

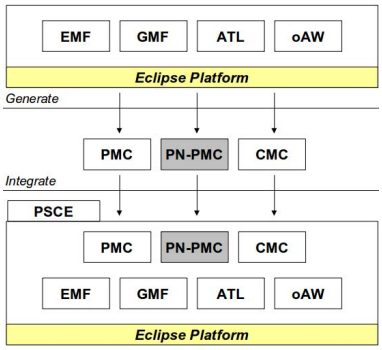

Figure 1. Architecture of the Model-Driven Engineering Environment.

the development of context-aware applications.

Figure 2 illustrates the development process supported by the environment. First, the modelling components support the design of the application's models and serve their validation using static OCL constraints. The Petri Net process model defines the execution logic and includes references to entities defined in the presentation and context models. In this way the validation of the dynamic behaviour is possible through the transformation of the PN process model to the standardised Petri Net Markup Language (PNML) ISO/IEC 15909-2 International Standard [19]. The PNML format is supported by various PN simulator tools that enable the execution of the model so as to guarantee the validity of its operational semantics. In case inconsistencies are detected in the PN model, the corresponding errors (e.g. execution deadlock) are raised and presented to the designer, which is responsible to rectify them prior to code generation.

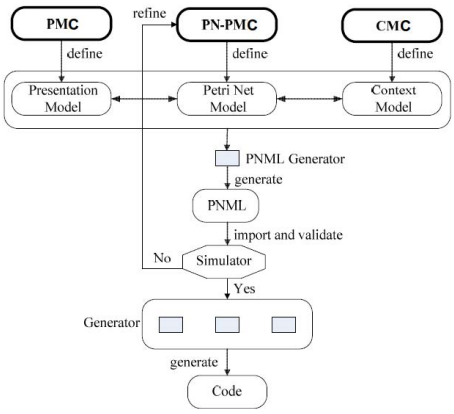

Figure 2. Model-driven Petri Net based process.

The validation of the static structure and dynamic behaviour, provides an unambiguous specification prior to code generation. This ensures that the generated implementation will not contain erroneous semantics. Figure 2 illustrates the defined code generators, which support models transformation to different implementations. In this work, J2ME

and Java code generators have been defined. Each code generator is composed of three sets of templates that support respectively the transformation of presentation, context and PN models. This allows generating a large part of the implementation, which is subsequently extended by the developer. In effect, automatic code generation reduces the implementation effort and speeds up the development of context-aware applications.

IV. Petri Net Process Modelling Component

This work addresses the development of context-aware applications with explicit focus on dynamic validation. Thus, the PN-PMC is introduced in detail since it enables the validation of the dynamic behaviour of these applications. Further details on the PMC and CMC are out of the scope of this paper but can be found in relevant work [6], [7]. The PN-PMC integrates the different modelling languages since it allows designing PN process models that include references to the entities defined in the presentation and context models.

The Petri Net Process Modelling Language (PN-PML) of the PN-PMC is defined in the form of an EMF-based metamodel. It extends the PNML standard that is also defined in this work as an EMF-based metamodel. The PNML standard defines a universal XML-based transfer syntax for PNs and provides an exchange format, which enables the compatibility and interoperability among heterogeneous PN modelling and simulator tools. The definition of the PN-PML by extending the PNML standard enables transformation of the designed PN models to the PNML format. This allows importing and validating models using various existing PN simulator tools.

The environment supports the definition of the PN-PML metamodel by extending (using the EMF import and inheritance mechanism) the PNML metamodel not presented in this paper due to space limitations. Figure 3 illustrates the metamodel that defines the elements, associations and properties that support modelling the application's logic.

The *DocumentRoot* metaclass is the container of model elements and represents the PN model. Containment relationships are depicted through the aggregation associations of the root metaclass. The *places* aggregation defines that each model can include zero to many instances of the *Place* metaclass, which extends respectively the *Node* metaclass of the PNML metamodel and inherits its properties. Each place represents a state in the application's execution and can be marked using inscriptions, which can be initialisation tokens "[]", primitive datatypes or tuples of primitive datatypes. Tokens depict the pre-conditions that enable the firing of a transition or the post-conditions when firing an enabled transition. Tokens are defined using the *Token* element, which extends the *Inscription* metaclass that allows defining syntactically correct conditions.

Six subclasses define different types of transitions that enhance the dynamic nature of the language in terms of model hierarchy and concurrency. The *Transition* element is the parent metaclass of the *Basic, Object, Downlink, Uplink, Action* and *Guard* transitions. The basic transition does not carry any expressions or inscriptions. It is defined using the *Basic* metaclass and it is enabled when the required tokens reside in its input places. The second subclass refers to the object transition and is defined using the *ObjectNet* metaclass. This allows creating instances of the entities (i.e. objects) defined in the presentation and context models. In this way a hierarchical structure is provided since the PN model contains objects that refer to instances of the entities included in the presentation and context models. Therefore, once an object transition fires, it creates the necessary objects that are deposited as tokens into the respective output places.

The concept of synchronous communication is introduced also in the modelling language. This notion of synchronous channels defines that two transitions can synchronise and fire atomically, provided that they initially agree on the name of the channel and the parameters involved in the synchronisation. The metamodel defines the *Downlink* and *Uplink* metaclasses, which describe the two transitions that allow establishing a synchronisation channel. The downlink transition is the initiator of the synchronisation channel and defines the invocation of a method that may carry one or more arguments. Respectively, the uplink transition is the terminating element of the synchronous channel, which serves requests delegated by the downlink transition. In terms of object-oriented programming this denotes a method defined in a class that implements the needed functionality and accepts one or more input parameters.

The *Action* metaclass allows defining action transitions that resemble user-generated actions; e.g. clicking a button. These actions have side effects that influence the state of objects circulating within the PN model. The guard is the final transition defined in the metamodel, which describes a basic transition extended by a logical expression. The *Guard* metaclass inherits its properties from the *Inscription* metaclass, which allows defining logical expressions associated to the guard transition.

Finally, the metamodel defines arc elements that provide additional control over the execution of the context-aware application. Arcs are defined using the *Arc* metaclass, which is associated to the *Attribute* metaclass that allows defining arc inscriptions by extending the *Inscription* metaclass. Inscriptions define either primitive datatypes, tuples of primitive datatypes, object nets or tuples of object nets and are used to evaluate the tokens consumed or generated by transitions. It is important to note that inscriptions and expressions are defined as Reference Nets [20] operators, which are equivalent to Java binary, logical and assignment operators. The following section presents the development of a context-aware prototype. Explicit focus and details are provided on the validation of the dynamic behaviour using the PN capabilities of the proposed framework.

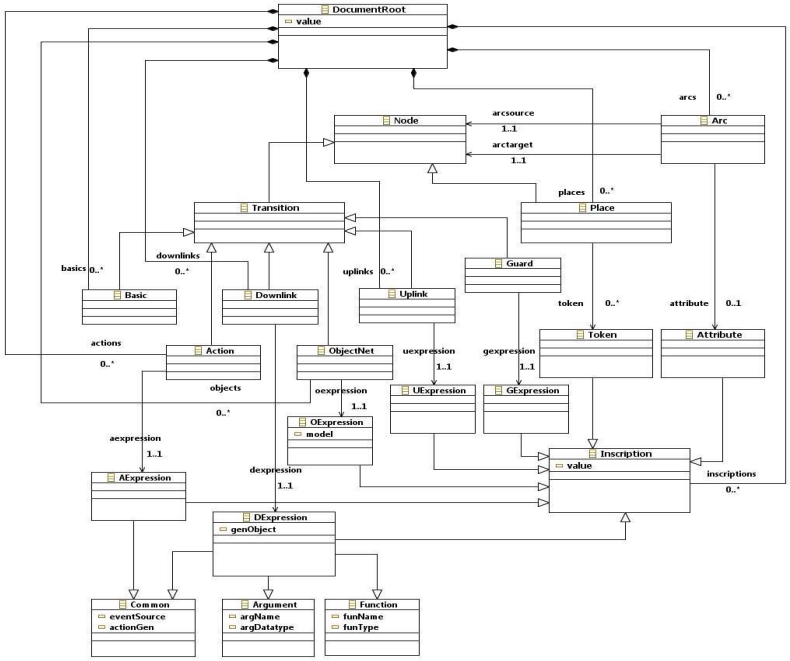

Figure 3. Petri Net based Process Modelling Language.

V. Context-Aware Prototype: A Museum Tourist Guide

A. Overview

The prototype presented in this work allows showcasing the applicability of the proposed framework for modelling, validating and implementing context-aware applications. This case study is selected since this application requires to be deployed on different platforms and be used by mobile users. The framework provides this capability since it ensures validation of the static and operational semantics of the models and allows generating different implementations from the models. The prototype refers to a museum tourist guide that aids and adapts the browsing experience of visitors. The application defines four zones that refer to the historic sites of the museum. Also, user-generated events are emulated in the code, so as to perform the required actions and adapt the behaviour of the application. For instance, the event generated by sensors when a user enters a historic site is emulated in the code. This event is captured by listeners, so as to dynamically present to the user related historical information (i.e. media, text-based).

B. Presentation and Context models

The prototype is initially defined in the form of models that represent the graphical user interfaces, the context information and the dynamic behaviour of the application. Figure 4 presents a small part of the presentation model (due to space limitations), which defines the GUIs of the prototype. The *Display* element is the main component (e.g. J2ME Display) that contains various container components (e.g. J2ME Form) that act respectively as carriers of additional graphical components (e.g. J2ME StringItem).

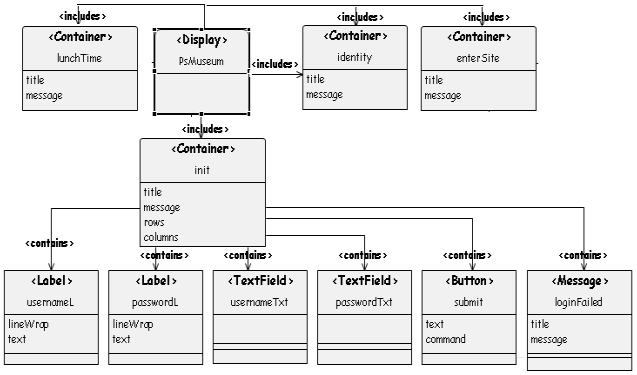

Figure 4. Part of the presentation model.

Figure 4 shows the *init Container* component with all its associated child components. This component defines a login form that enables the user to enter the required authentication information, in order to use the context-aware application. The component acts as the container of secondary graphical components (e.g. *submit Button*) and includes also properties defined using the modelling editor's properties view. For instance, the *title* property defines the title that appears on this container.

The context model describes the context-awareness characteristic in the form of an advanced information model.

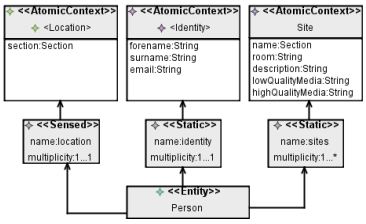

Figure 5. Small part of the context model.

It defines the entities, context sources, context information and contextual situations relevant to the application. Figure 5 illustrates a fraction of the context model, which presents the *Person* entity that resembles the user of the service. This part of the model captures the user of the application, which is associated using the static *identity* source to the *Identity* context. The user's profile includes information such as the forename and surname; defined in the form of *String* primitive datatypes. This source is static since information can be retrieved from a database. An additional static context source (i.e. *Site*) is defined in the model, which associates each user with the different historic sites of the museum.

The modelling components allow designing the models and validating their structure by enforcing static OCL constraints. This ensures the consistency of the structure of the models that define the context-aware application. The following section presents details on the definition of the PN process model and its validation by means of dynamic model execution.

C. Petri Net process model

The functionality of the application is defined in the form of a PN model. In this model entities of the presentation and context models are defined in the form of tokens circulating through the model during its execution. This approach allows composing a unified application specification through the PN model. Moreover, the operational semantics of the application can be validated using external PN simulation tools by transforming the model to the PNML format. This enables the validation of the dynamic behaviour, complements the validation of the static structure of the models and ensures the correctness of the generated code.

Figure 6 illustrates the functionality of the application, which kicks off its execution from states p_1, p_3 and p_5 that are populated with initialisation tokens. Therefore, transitions t_1, and t_2 fire and create instances of the *Person* and *Device* entities defined in the context model. In addition, transition t_3 is enabled and creates an instance of the *PsMuseum* entity defined in the presentation model. This hierarchical structure reflects the concept of object-oriented programming (OOP), where the main class is aware of other classes and can utilise their parameters and functions. Following, the downlink transition t_4 is enabled, defining a communication channel with the uplink transition t_{19}. In terms of OOP t_4 refers to a method invocation, while t_{19} refers to the method that implements the needed functionality. In this case the method returns the login form that allows the user to enter his authentication details.

At this stage the user enters his credentials and clicks the submit button. Transition t_5 defines this user-generated action and enables upon firing the downlink transition t_6 and its associated uplink transition t_{20}. The communication channel represents the functionality that authenticates the credentials of the user. In case authentication is unsuccessful transition t_{29} is enabled and the user is asked to re-enter his credentials. Otherwise transition t_{30} is enabled and the most important state is reached; i.e. p_9. The non-deterministic behaviour of the context-aware application is exhibited at this state, since different execution paths can be undertaken. This enables designers to define the complex behaviour of the application without any restrictions in terms of concurrency and non-deterministic requirements.

In particular, the user is able to quit the application (i.e. action transition t_{31}) reaching the final state p_{20} in the model execution. The user may also generate an action (i.e. transition t_7) and choose a historic site (i.e. transition t_8 and t_9), so as to be presented with relevent historical information. Synchronised transitions t_{10} and t_{22} compose the method that allows retrieving the information and presenting them to the user. Apart from user generated events, different contextual situations may occur from state p_9. For instance, the site contextual situation composed by the synchronised transitions t_{17} and t_{26} indicates that the user has entered a historic site. Hence, the behaviour of the application defines that related media and text-based context information should be presented to the user. By following the same reasoning the execution paths of all contextual situations are defined in the model.

Listing 1. Part of the PNML transformation template using pseudocode.

```
1. FOREACH downlinks AS downlink
2. Generate downlink id
3. IF (downlink.expression != null)
4.     Generate expression graphics
5.     Generate expression operator
6. ENDIF
7. Generate transition graphics
8. ENDFOREACH
```

The definition of the dynamic behaviour concludes the design of the context-aware application. Following the static structure of the models is validated and the transformation of the PN model to the PNML format is performed. This enables the validation of the operational semantics of the application by means of model execution. Listing 1 shows part of the defined transformation template in the form of pseudocode. In particular, it illustrates the transformation of a downlink transition to the corresponding PNML format. It initially defines an iteration that allows parsing downlink transitions defined in the PN model. In case the

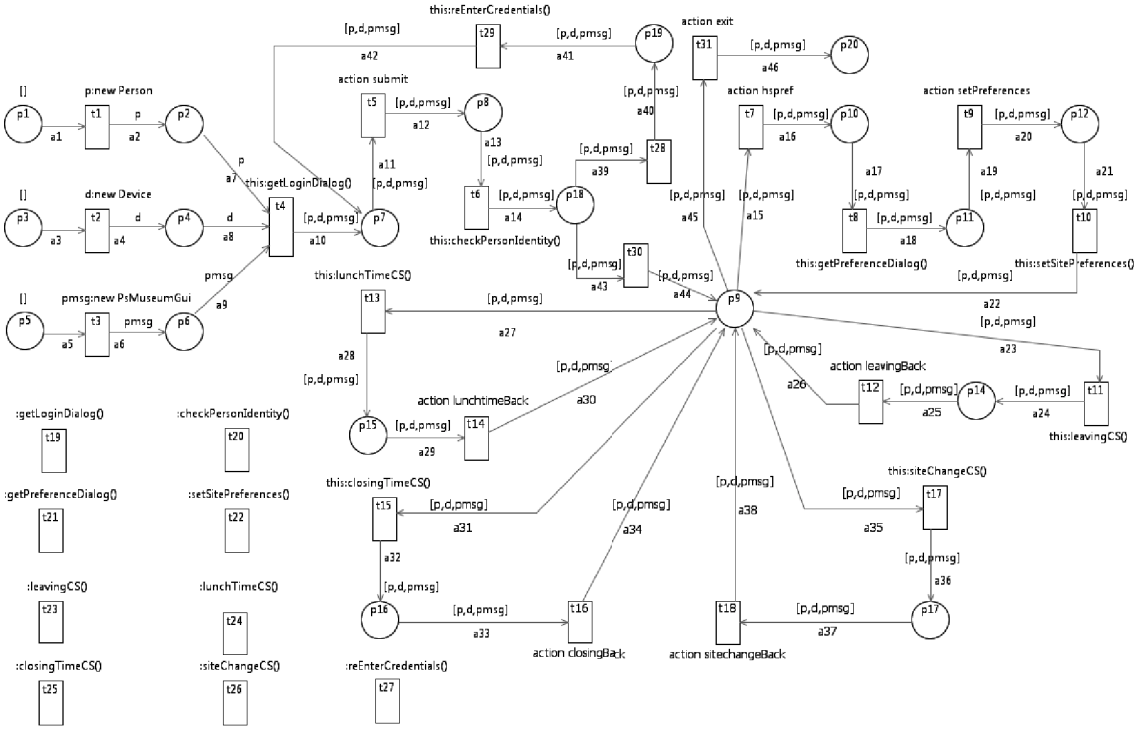

Figure 6. The Petri Net process model.

downlink transition defines an expression, the graphical representation and value of the expression are generated in PNML format. Then, the graphical representation of the downlink transition is also generated. This enables to import elements, properties and relationships in existing PNML-enabled simulators that support PN model execution. Listing 2 shows the resulting PNML format of the downlink transition t_{10} (i.e. Figure 6) generated after execution of the aforementioned transformation template.

Listing 2. The PNML representation of downlink transition t_{10}.

```
1. <transition id="t10">
2.    <downlink>
3.       <graphics>
4.          <offset x="0" y="-30" />
5.       </graphics>
6.       <text>this:setSitePreferences()</text>
7.    </downlink>
8.    <graphics>
9.      <position x="696" y="165" />
10.     <dimension x="25" y="45" />
11.     <fill color="rgb(112,219,147)" />
12.     <line color="rgb(0,0,0)" />
13.   </graphics>
14. </transition>
```

The transformation of the model allows to import, redraw and validate the dynamic behaviour of the context-aware application. In this work the Renew PN simulator [20] is used to carry out the model simulation. Figure 7 illustrates a snapshot of a part of the model during the executed simulation. The model execution allows validating the correctness of the operational semantics of the application; e.g. detect and rectify deadlocks. Hence, the successful model execution ensures the integrity and correctness of the operational semantics of the code to be generated. Following, the code generation phase is executed, where a significant part of the implementation is generated from the presentation, context and PN process model. The meta-models, the complete models of the prototype and a demo video showing the PN model validation can be found at http://www.cs.ucy.ac.cy/∼aachila/research.html.

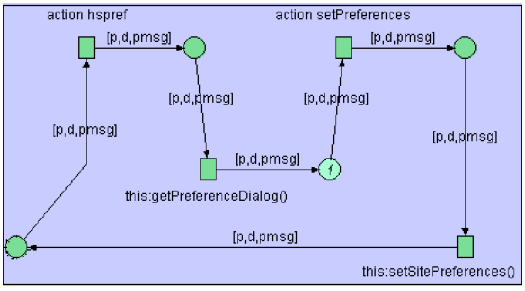

Figure 7. Validation of the model using a Petri Net simulator tool.

VI. Conclusions and Future Work

In this paper a Petri Net based validation framework is proposed, which combines MDA with Petri Nets to

support the validation of the static and dynamic structure of context-aware applications. This complements existing context modelling approaches that undertake validation by enforcing static OCL constraints defined in class diagrams. The framework defines a systematic process and a model-driven engineering environment that enable designers to define context-aware applications in the form of models. The adoption of the widely-used EMF provides a common understanding of the modelling languages and improves communication between designers and developers. Moreover, the OCL language and the Petri Nets formalism allow validating both the static and dynamic structure of context-aware applications. This ensures the integrity of the specification prior to generating the implementation. An added benefit of the approach is the capability to rapidly adapt the code generation phase to address new platforms (e.g. Android, iOS) by defining the required code generators.

The limitations of the approach are the initial effort required to define the modelling languages and the necessary code generators, which is compensated to a degree by the dynamic validation and automatic code generation capabilities. Future work will examine the development of an Eclipse plug-in for the development environment, which will avoid generating an intermediary PNML format and using external PN tools (e.g. Renew) for the validation of the dynamic behaviour of the context-aware application.

REFERENCES

[1] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, D. Riboni, "A survey of context modelling and reasoning techniques", *Elsevier Pervasive and Mobile Computing Journal*, Volume 6, Issue 2, Pages 161-180, ISSN 1574-1192, 10.1016/j.pmcj.2009.06.002, April 2010.

[2] V. Dhingra and A. Arora, "Pervasive Computing: Paradigm for New Era Computing", *Proc. of the IEEE International Conference on Emerging Trends in Engineering and Technology*, Nagpur, India, July 2008, pp. 349-354.

[3] Object Management Group (OMG), "Object Constraint Language Specification (OCL) 2.2", Available online: http://www.omg.org/spec/OCL/2.2/, 2010.

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

[5] C. Girault and R. Valk, *Petri Nets for System Engineering: A Guide to Modelling, Verification and Applications*, Springer, 2003.

[6] A. Achilleos, K. Yang and N. Georgalas, "Context modelling and a context-aware framework for pervasive service creation: A model-driven approach", Elsevier Pervasive and Mobile Computing, July 2009.

[7] A. Achilleos, N. Paspallis and G. A. Papadopoulos, "Automating the Development of Device-Aware Web Services: A Model- Driven Approach", in Proceedings of the IEEE Signature Conference on Computer Software and Applications (COMPSAC), 2011.

[8] F. Ay, "Context Modelling and Reasoning Using Ontologies", Technical Report, University of Technology, Berlin, July 2007, pp. 1-9.

[9] H. Chen, F. Perich, T.W. Finin and A. Joshi, "SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications", 1st Annual International Conference on Mobile and Ubiquitous Systems, MobiQuitous 2004, IEEE Computer Society (2004).

[10] H. Chen, T. Finin and A. Joshi, "Semantic Web in the Context Broker Architecture", Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications, PerCom 2004, IEEE Computer Society, (2004).

[11] E. Serral, P. Valderas and V. Pelechano. "Towards the Model Driven Development of context-aware pervasive systems". *Pervasive and Mobile Computing Journal*. vol. 6, no. 2, pp. 254-280, Apr. 2010.

[12] K. Henricksen and J. Indulska, "Developing Context-Aware Pervasive Computing Applications: Models and Approach", *Pervasive and Mobile Computing Journal*, vol. 2, no. 1, pp. 37-64, Feb. 2006.

[13] J. Indulska, J. Fong and R. Robinson, "Tool Support for Designing CML Based Context Models in Pervasive Computing", *Proc. of the 4th Workshop on Software Engineering Challenges for Ubiquitous and Pervasive Computing (UPC'2010) held in 7th International Conference on Pervasive Services*, Jul. 2010, pp. 232-238.

[14] C. Simons and G. Wirtz, "Modelling Context in Mobile Distributed Systems with the UML", *Journal of Visual Languages and Computing*, vol. 18, no. 4, pp. 420-439, Aug. 2007.

[15] C. R. G de Farias, M. M. Leite, C. Z. Calvi, R. M. Pessoa and J. G. P. Filho, "A MOF metamodel for the Development of Context-Aware Mobile Applications", *Proc. of the ACM symposium on Applied computing*, Seoul, Korea, Mar. 2007, pp. 947-952.

[16] C. Dumez, J. Gaber and M. Wack, "Model-driven Engineering of Composite Web Services using UML-S", *Proc. of the International Conference on Information Integration and Web-based Applications and Services*, 2008, pp. 395-398.

[17] C. Dumez, A. Nait-Sidi-Moh, J. Gaber and M. Wack, "Modelling and Specification of Web Services Composition using UML-S", *Proc. of the International Conference on Next Generation Web Services Practices*, Oct. 2008, pp. 15-20.

[18] H. Dun, H. Xu, L. Wang, "Transformation of BPEL Processes to Petri Nets", *Proc. of the IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering*, China, June 2008, pp.166-173.

[19] E. Kindler, "Software and Systems Engineering - High-level Petri Nets: Part2 Transfer Format - Proposed Draft Addendum to International Standard", ISO/IEC 15909 Part 2 - Version 0.6.3, June 2005.

[20] O. Kummer, F. Wienberg, M. Duvigneau and L. Cabac, "Reference Net User Guide", Available online: http://www.informatik.uni-hamburg.de/TGI/renew/renew.pdf, Theoretical Foundations Group, Department of Informatics, University of Hamburg, 2009.