Towards a Model-Driven Approach for Context-Aware Software

Jose Bocanegra
Systems Engineering Department
Pontificia Universidad Javeriana
Bogotá, Colombia
jose_bocanegra@javeriana.edu.co

***Abstract*—Context-aware software (C-AS) uses context to provide users with relevant information or services. Although C-AS has an increasing importance, this area is still immature in terms of tools, languages, and methods. Therefore, C-AS presents several opportunities and challenges for software engineers, and new models and theories are needed to address these challenges. Some of the main problems in software engineering for C-AS are the lack of mechanisms for representing some relevant concepts in the requirements and design specifications directly, and the complexity to translate requirements and design specifications to a concrete implementation. As an alternative of solution, the research project proposes MiDAS, a framework that uses a model-driven approach to develop C-AS. Specifically, MiDAS comprises two domain-specific languages to specify requirements and design in C-AS; a model-to-model transformation to derive design models from requirements models; a model-to-text transformation to derive a platform independent model (PIM) from design models; and a model-to-text transformation to derive context-aware software for specific platforms, taking as input the PIM.**

I. Problem

An adaptive software is able to adapt its own behavior when a change occurs in its surrounding context [1]. A context-aware software is a particular kind of adaptive software that uses context to provide the users with relevant information or services [2].

For example, a context-aware application may provide specially-tailored contents to students considering their cognitive skills or prior knowledge [3].

Some authors [1], [4], [5], [6] have identified several challenges in the development process of context-aware software. These challenges can be grouped in two areas: (i) requirements and design, and (ii) implementation.

In the field of requirements and design, the main issue is the lack of mechanisms for representing the *relevant concepts in C-AS* in the requirements and design specifications.

Some of these relevant concepts are the user (the person or group of people that performs an operation with the system or the person or group of people affected by an operation); the operations performed by the system; the objects (information required or affected by an operation); the contextual entities (entities that should be considered for context manipulation purposes); and contextual elements (any piece of data or information that can be used to characterize an entity in an application domain).

Generally, these concepts are usually placed directly in the source code [7], [8]. However, employing these kind of ad hoc solutions usually increase the development effort and complexity, and this reduces the project success rate [9].

One possible strategy to address this problem is to use general purpose languages [10] to model requirements and design for C-AS. However, these languages are not sufficiently expressive to represent these relevant concepts.

The use of Domain-Specific Languages

(DSL) would be more accurate, brief, and with fewer errors than using modeling languages of general purpose [11], [12].

In the field of implementation, the problem is the difficulty of developing context-aware software based on the specified requirements. Assuming that software engineers have an adequate language to formally specify the requirements and design of C-AS, one problem persists: to translate those specifications to an implementation. The traditional way to solve this problem is to manually code a system that implements the requirements and design specified by software engineers [13]. In the domain of context-aware software, this scheme has several opportunities for improvement.

One of them is that context-aware software tends to share similar features. For example, some context-aware applications take into account the location and characteristics of the user to display information about places of interest. An application can display the closest restaurants to a user and filter those that suit the user tastes. An application related to the medical field can display a list of nearby hospitals that provide specialists for a particular pathology of a user. In both cases, the services provided by both systems are very similar and can be abstracted into reusable modules. Although this is a common practice in developing traditional systems, to the best of our knowledge, there are no libraries that provide specially-tailored functionality for context-aware software.

Another opportunity for improvement is related to the manual coding of certain parts of context-aware software. Assuming that one has a set of components that provide part of the functionality of context-aware software, there is still the need to manually integrate them into a full application and to add all of the functionality that is not commonly implemented by those components. Since manual coding is slow, wasteful, and error-prone, there is an improvement opportunity by automatizing the implementation through code generation. A basic premise of this proposal is that this automation is possible and would significantly accelerate the development process, similarly to what happens with traditional systems.

Taking into account the above assumptions, this research project addresses two fundamental issues, which are embodied in the following research questions: *(i) how to develop a domain-specific language to specify the requirements and design of* context-aware software*?;* and *(ii) how to automate code generation of* context-aware software *from the requirements and design specifications?*

II. Related Work

This section analyzes the works that have considered the concepts of requirements and design specifications of context-aware software, and the implementation of context-aware software from these specifications.

It is important to clarify that most of related works are oriented to adaptive and self-adaptive systems. Although a context-aware system is a subset of adaptive systems, the former has a much more specialized field of action than the latter. In this way, if the process for specifying requirements and design in adaptive and self-adaptive software has had limited attention [14] and it is a complex task [1], [15], this limitation is more relevant in context-aware systems [16]. Therefore, most of the works analyzed in this section are particularly oriented to adaptive and self-adaptive systems.

One of the most relevant works in the field of requirements for adaptive software is RELAX [14]. RELAX is a DSL designed to support the rigorous specification of adaptive systems requirements, that explicitly addresses uncertainty inherent in these systems. The formal semantics for RELAX is expressed in terms of fuzzy logic. That language also “enables developers to identify uncertainty in the requirements, thereby facilitating the design of systems that are, by definition, more flexible and amenable to adaptation in a systematic fashion”. Although the main focus of RELAX is the formalization of several operators (modal, temporal, and ordinal operators and uncertainty factors), some important elements required in a context-aware system, such as the

subjects and the objects are not considered. In addition, this work does not propose transformations to the design and subsequents phases in the software development process.

Other work related to requirements is Adaptive RML [17]. This is a modeling language focused on the representation of early requirements for self-adaptive systems. The language has graphical primitives in line with classical goal modeling languages and it is formalized via a mapping to Techne [18]. As Adaptive RML takes as reference classical goal-oriented languages such as i^*, it contains several flaws in its visual notation [19].

The work of Hog *et al.* [20] proposes AWS-UML, an UML profile for adaptive web services which increases the expressiveness of UML. AWS-UML provides (i) a special use case model with three kind of actors (application consumer, human consumer, and provider) and a special icon for each one of them; (ii) three predefined use cases with a specific notation, (iii) a set of OCL constraints, (iv) a class diagram enriched with user's profiles; and (v) a sequence diagram with predefined objects and predefined messages. Similarly, [21] propose an extension to UML to represent contextual information using profiles that contain stereotypes, tagged values and constraints.

In the area of design, Vogel and Giese [22] propose EUREMA, a model-driven approach for engineering adaptation engines for self-adaptive software. EUREMA provides a domain-specific modeling language to specify and an interpreter to execute feedback loops. However, EUREMA only aims at the specifications and implementing adaptation engines, which are closer to the solution domain rather than the problem domain.

In [23], the authors present PerCAS, a model-driven approach for developing dynamic and personalized context-aware services. A natural language-like rule language is proposed for specifying context-awareness logic and personalized rules can be dynamically switched at runtime. However, the proposed language does not provides a way to model subjects and objects.

Hoyos *et al.* [12] propose MLContext, a textual Domain-Specific Language (DSL) which is specially tailored for modeling context information. The work also provides the generation of software artifacts from abstract models which do not include implementation details. In MLContext, models do not include information related to specific platforms or implementations, thus, the models can be reused in different context-aware applications which are based on the same context. The language also supports a taxonomy of types of context.

In conclusion, (i) none of the proposals takes into account all the relevant concepts defined for a C-AS; (ii) they do not propose both transformations (from requirements to design and from design to implementation); (iii) they only use a kind of notation; and (iv) the works do not support some of the language design principles.

III. Proposed Solution

To solve the research problem stated in Section I, this research project proposes MiDAS, a model-driven approach to develop context-aware software. MiDAS is composed by:

1) c-RSL: a domain-specific language to specify requirements in C-AS. c-RSL addresses the representation of users, objects, contextual entities, and contextual elements.
2) c-DSL: a domain-specific language to specify the design in C-AS. c-DSL addresses the representation of the operations (i.e., the way in which the system adapt a content to the user, or the the way in which the system executes a service).
3) A set of transformations. The first transformation is aimed to derive design specifications from requirements specifications. The second transformation addresses the derivation of a platform independent model (PIM) from a design specification model. The third transformation generates context-aware software in specific platforms (*e.g.*, J2EE, PHP, Android) taking as input the PIM.

Both languages use two types of notation (textual and visual). Textual representations have the advantage of detailing the specific elements of a system whereas graphical representation provide a better overview and ease the understanding of models.

The languages also cover three principles suggested by Moody in terms of symbol redundancy, symbol overload, and perceptual discriminability.

The modeling process and the transformations will be supported by means of a case tool based-on frameworks such as Eclipse EMF, XText, Sirius, and Accelo.

IV. Plan for Evaluation and Validation

The main aspects to be evaluated in the framework are (i) the usability and (ii) expressiveness of the proposed languages, (iii) the consistency of the models generated by the transformation, and (iv) the productivity improvement.

Usability evaluation will be based on some metrics proposed by Hoyos et al. [12]. These metrics measure, for example, the required effort to understand grammar or the facility to learn the language.

Expressiveness will be measured by requesting a group of C-AS expert developers that use the framework and determine if it is possible to represent the most important concepts required to develop a functional system from the specifications.

To evaluate the consistency of the automatically generated models (particularly the design models), we will develop a set of algorithms to ensure that there are no redundant, lost, missing, and mismatched data [24].

In regards to productivity, we will measure the ratio between models and generated code.

V. Expected Contributions

This research project expects to have following outcomes: (i) the development of two domain-specific languages; (ii) the definition and implementation of three transformations; and (iii) the validation and evaluation of the developed framework. It is expected that with these contributions, the software development community may use the framework to develop context-aware software. The use of the proposed framework may reduce the development times, improve the quality of developed systems, and increase the productivity and competitiveness of software organizations.

VI. Current Status

The following is the list of the preliminary work developed in the context of this research project.

First, an outline of the main problems in software engineering for adaptive software and the definition of the big picture of MiDAS were presented in [25]. Second, in [26] was presented the correlation between the main problems to develop adaptive software and Model-Driven Engineering. Third, the author of this work has developed a preliminary version of c-DSL[1] [27]. That work also provides a functional prototype based on the Generic Modeling Environment (GME) and suggests the first steps to validate the graphical notation. Fourth, an improved version of c-DSL which uses a more complex metamodel, a refined notation, and a new functional prototype based on the Sirius plugin for Eclipse was presented in [28].

The next proposed activities are the following: (i) the definition of the transformations from requirements to design and from design to implementation; (ii) the creation of the CASE tool as support for the entire software development process; (iii) the definition of the experiments to evaluate and validate the framework; and (iv) the application of these experiments.

References

[1] B. H. Cheng, R. De Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, *et al.*, "Software engineering for self-adaptive systems: A research roadmap," in *Software engineering for self-adaptive systems*, pp. 1–26, Springer, 2009.

[2] A. K. Dey, "Understanding and using context," *Personal and ubiquitous computing*, vol. 5, no. 1, pp. 4–7, 2001.

[1]This language was previously named DMLAS

[3] G. G. Castro, E. L. Dominguez, Y. H. Velazquez, M. Y. R. Matla, C. B. E. Toledo, and S. E. P. Hernandez, "Mobilearn: Context-aware mobile learning system," *IEEE Latin America Transactions*, vol. 14, no. 2, pp. 958–964, 2016.

[4] M. Salehie and L. Tahvildari, "Self-adaptive software: landscape and research challenges," *ACM TAAS*, vol. 4, no. 2, p. 14, 2009.

[5] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*, pp. 1–32, Springer, 2013.

[6] F. D. Macías-Escrivá, R. Haber, R. del Toro, and V. Hernandez, "Self-adaptive systems: A survey of current approaches, research challenges and applications," *Expert Systems with Applications*, vol. 40, no. 18, pp. 7267–7279, 2013.

[7] X. Mao, M. Dong, L. Liu, and H. Wang, "An integrated approach to developing self-adaptive software," *Journal of Information Science and Engineering*, vol. 30, no. 4, pp. 1071–1085, 2014.

[8] B. H. Cheng, P. Sawyer, N. Bencomo, and J. Whittle, "A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty," in *Model Driven Engineering Languages and Systems*, pp. 468–483, Springer, 2009.

[9] T. Ruiz-López, C. Rodríguez-Domínguez, M. J. Rodríguez-Fórtiz, S. F. Ochoa, and J. L. Garrido, "Context-aware self-adaptations: From requirements specification to code generation," in *UCAmI*, pp. 46–53, Springer, 2013.

[10] G. Booch, J. Rumbaugh, and I. Jacobson, *The unified modeling language user guide*. Addison-Wesley, 1999.

[11] S. Kelly and J.-P. Tolvanen, *Domain-specific modeling: enabling full code generation*. John Wiley & Sons, 2008.

[12] J. R. Hoyos, J. García-Molina, and J. A. Botía, "A domain-specific language for context modeling in context-aware systems," *Journal of Systems and Software*, vol. 86, no. 11, pp. 2890–2905, 2013.

[13] F. Fleurey and A. Solberg, "A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems," in *Model Driven Engineering Languages and Systems*, pp. 606–621, Springer, 2009.

[14] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel, "Relax: a language to address uncertainty in self-adaptive systems requirement," *Requirements Engineering*, vol. 15, no. 2, pp. 177–196, 2010.

[15] N. Esfahani and S. Malek, "Uncertainty in self-adaptive software systems," in *Software Engineering for Self-Adaptive Systems II*, vol. 7475, pp. 214–238, Springer, 2013.

[16] U. Alegre, J. C. Augusto, and T. Clark, "Engineering context-aware systems and applications: a survey," *Journal of Systems and Software*, vol. 117, pp. 55–83, 2016.

[17] N. A. Qureshi, I. J. Jureta, and A. Perini, *Towards a requirements modeling language for self-adaptive systems*, pp. 263–279. Springer Berlin Heidelberg, 2012.

[18] I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, "Techne: towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling," in *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pp. 115–124, IEEE, 2010.

[19] D. L. Moody, P. Heymans, and R. Matulevicius, "Improving the effectiveness of visual representations in requirements engineering: an evaluation of i* visual syntax," in *2009 17th IEEE International Requirements Engineering Conference*, pp. 171–180, IEEE, 2009.

[20] C. Hog, R. B. Djemaa, and I. Amous, "Towards an UML based modeling language to design adaptive web services," in *Proceedings of the International Conference on Semantic Web and Web Services*, pp. 38–44, 2011.

[21] M.-S. Benselim and H. Seridi-Bouchelaghem, "Towards a uml profile for context-awareness domain.," *International Arab Journal of Information Technology (IAJIT)*, vol. 14, no. 2, 2017.

[22] T. Vogel and H. Giese, *Model-driven engineering of adaptation engines for self-adaptive software: Executable runtime megamodels*. Universitätsverlag Potsdam, 2013.

[23] J. Yu, J. Han, Q. Z. Sheng, and S. O. Gunarso, "Percas: an approach to enabling dynamic and personalized adaptation for context-aware services," in *International Conference on Service-Oriented Computing*, pp. 173–190, Springer, 2012.

[24] S. Sadiq, M. Orlowska, W. Sadiq, and C. Foulger, "Data flow and validation in workflow modelling," in *Proceedings of the 15th Australasian database conference-Volume 27*, pp. 207–214, Australian Computer Society, Inc., 2004.

[25] J. Bocanegra, J. Pavlich-Mariscal, and A. Carrillo-Ramos, "MiDAS: a model-driven approach for adaptive software," in *Proceedings of the 11th International Conference on Web Information Systems and Technologies - Volume 1: WEBIST,*, pp. 281–286, INSTICC, SciTePress, 2015.

[26] J. Bocanegra, J. Pavlich-Mariscal, and A. Carrillo-Ramos, "On the role of model-driven engineering in adaptive systems," in *Computing Conference (CCC), 2016 IEEE 11th Colombian*, pp. 1–8, IEEE, 2016.

[27] J. Bocanegra, J. Pavlich-Mariscal, and A. Carrillo-Ramos, "DMLAS: a domain-specific language for designing adaptive systems," in *2015 10th Computing Colombian Conference (10CCC)*, pp. 47–54, Sept 2015.

[28] J. Bocanegra, J. Pavlich-Mariscal, and A. Carrillo-Ramos, "Towards a domain-specific language to design adaptive software: the DMLAS approach," *Ingeniería y Universidad*, vol. 20, no. 2, pp. 277–296, 2016.