MiDAS: A Model-Driven Approach for Adaptive Software

José Bocanegra, Jaime Pavlich-Mariscal and Angela Carillo-Ramos
Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana, Bogotá, D.C., Colombia

Keywords: Adaptation, Design, MDE, Requirements, Transformations.

Abstract: Some of the main problems in software engineering for adaptive software are: the lack of mechanisms to specify adaptive characteristics in software requirements; the difficulty to obtain a functional adaptive system based on the elicited requirements; and the need of maintaining synchronization and traceability between the requirements, design and implementation. To address the above problems, this paper proposes MiDAS, a framework that uses a model-driven approach to develop adaptive software. Specifically, MiDAS provides: (i) a new language for requirements engineering process that takes into account uncertainty in adaptive software; (ii) a method to derive concrete implementations in specific architectures supporting run-time adaptation; and, (iii) a mechanism to maintain traceability and synchronization between requirements specifications, design models and implementation architectures.

1 INTRODUCTION

An adaptive software system is a system able to "modify itself at run-time due to changes in the system, its requirements, or the environment in which it is deployed" (Andersson et al., 2009). Adaptive software plays an important role in several scenarios. One of them is when there are frequent changes in the context/goals/requirements during run-time. For example, in a slow connection, an adaptive software may modify its behavior at run-time and change the format in which information is presented (from graphical to textual) to give the user a better experience. Another scenario is when users with different abilities interact with software. For example, an Adaptive Learning Management System may adjust the contents presented to the student taking into account his/her skills, and so, improving the educational process.

However, developing adaptive software is a task far from trivial. Cheng et al., (2009a), identified the main problems in software engineering for adaptive software. The first issue is the lack of mechanisms to specify adaptive characteristics in software requirements. In traditional software development process, software engineers use standard phrases or statements, called *linguistic patterns*, which are habitual in requirement specifications (Durán et al., 1999). Some traditional linguistic patterns are *"The system shall do this <objective>", or "The system shall store information about <relevant concept>"*.

Linguistic patterns facilitate requirements writing because "requirements information is structured in a fixed form, so requirements engineers know what missing information must be searched, requirements can easily be treated by a software tool" (Toro et al., 1999). However, traditional linguistic patterns to elicit requirements are not suitable to deal with uncertainty, a key feature in adaptation (Esfahani and Malek, 2013). According to Cheng et al., (2009a), traditional linguistic patterns have a notion of mandatory behavior versus the optional behavior that could be used to specify adaptive systems. Adaptive software requires a new requirement vocabulary and new linguistic patterns, such as *The system might do this..., but it may do this... as long as it does this...*, or *The system ought to do this... but if it cannot, it shall eventually do this...* (Cheng et al., 2009a).

The second issue is the difficulty to design and implement a functional adaptive system based on the elicited requirements. There are two many reasons. First, in current approaches, the transition between a set of functional requirements to a design is not direct, repeatable, or constructive (Dromey, 2003). Second, the adaptation logic is "typically specified at the code level, tightly coupled with the main system functionality, making it hard to control and maintain" (Fleurey and Solberg, 2009).

It is important to provide systematic methods to automate the transition from requirements specifications to design and to concrete implementations that

Bocanegra J., Pavlich-Mariscal J. and Carrillo-Ramos A..
MiDAS: A Model-Driven Approach for Adaptive Software.
DOI: 10.5220/0005486202810286
In *Proceedings of the 11th International Conference on Web Information Systems and Technologies* (WEBIST-2015), pages 281-286
ISBN: 978-989-758-106-9

support run-time adaptation (Salehie and Tahvildari, 2009). These methods should help to reduce or eliminate the need to code an adaptive system manually.

The third issue is the need to maintain coherence between the requirements, design and implementation. Beatty and Cheng (2012), explain that a change request in a requirements specification may affect other parts of a system. A change request may affect elements such as code, business requirements, functional requirements, systems requirements, user requirements, external interface requirements, quality attributes, architecture, user interface, software design, and source code. For instance, if there is a change in a business requirement, this change may affect architecture, user interface, or software design.

A possible solution to these issues is traceability. Traceability is the ability to verify the history, location, or application of an item by means of documented recorded identification. In software engineering traceability provides a clear, bidirectional link between requirement and design models, between design and code, and between code and test cases (Cleland-Huang et al., 2014).

To address all the problems mentioned above, Model-Driven Engineering is a possible solution. Model-Driven Engineering (MDE) (Kent, 2002) is an approach to develop software. MDE uses models to specify different aspects of a system, from requirements and domain-specific constructs, to the various aspects of design descriptions. A key element in MDE is the use of transformation tools to automatically create the code that realizes those models. These transformation tools take as input a set of models representing domain-specific or design concepts, and output code in a certain programming language and platform that realizes those models.

Taking as reference the advantage of MDE, this paper proposes MiDAS, a framework that uses a model-driven approach to develop adaptive software.

The expected contributions of MiDAS will be the following.

First, MiDAS will provide new languages to specify requirements and design in adaptive software. These languages are depicted in Section 3.1 and 3.2.

Second, MiDAS will provide a mechanism to both (i) derive a system implementation from requirements and design specifications, and (ii) maintain traceability and synchronization between requirements, design, and implementation models. These mechanisms will be supported by a language that is detailed in Section 3.3.

Third, MiDAS will provide a Case Tool that integrates all of the above elements to support the entire development process for adaptive software. This tool is presented in Section 3.4.

Fourth, this paper presents the expected advantages of use MiDAS. These advantages are presented in Section 4.

This paper concludes with a set of conclusions and future venues of research.

2 BASELINE AND RELATED WORK

The requirements specification process in adaptive systems has not been frequently addressed (Whittle et al., 2010). Some authors indicate that "it is a challenging task due to the inherent uncertainty associated with an unknown environment" (Cheng et al., 2009b). Due to this uncertainty, variability in adaptive systems is hard to identify and model (Greenwood et al., 2011).

Although models for adaptive systems are often constructed in an *ad-hoc* manner, several authors ((Espada et al., 2011), (Gnaho et al., 2013)) have analyzed the use of well-known languages. Ahmad et al., (2013) have studied the differences and potential combinations of 4 different requirements modeling languages in adaptive systems: KAOS(Lamsweerde, 2009), SysML (OMG, 2015), SysML/KAOS (Laleau et al., 2010), and RELAX (Whittle et al., 2010). Although all the above approaches address important issues of adaptive systems, they do not address traceability. Moreover, these works only take into account static aspects of the system, and shelve the dynamic aspects. In the specific case of RELAX, this is a textual language for dealing with uncertainty in adaptive systems. However, the language "was not integrated with modeling approaches used in the requirements engineering community" (Whittle et al., 2010).

Model-Driven Engineering and adaptive software have been studied, among others, by Vogel and Giese (2014) . Those authors propose EUREMA, a model-driven approach to develop adaptive software. However, EUREMA is only oriented to the specification and execution of adaptation engines, but it does not cover elements of requirements specification.

The above proposals combine adaptation, requirements engineering, and MDE. However none of them provide an integrated approach to address all the challenges explained in the introduction of this work. Particularly, there are no approaches that propose a complete process that covers requirements, design, and implementation of adaptive software. Table 1 provides the comparison of related work.

Table 1: Comparison of related work.

Criteria		Approaches					
		KAOS	SysML	SysML /KAOS	RELAX	EUREMA	MiDAS
Traceability	Reqs/Design						+
	Design/Implem						+
Static aspects	Goals	+	+	+	+		+
	User profiles	+	+	+			+
	Context profiles	+	+	+			+
Dynamic aspects	Processes					+	+

3 MiDAS

The analysis of the previous sections motivates the proposal of a new approach to develop adaptive systems, called MiDAS. MiDAS is an acronym for *Model-driven approach for adaptive systems* and is expected to: (i) provide a new language for adaptive systems; (ii) derive a functional system based on the specified requirements; and, (iii) address traceability and synchronization between requirements, design, and implementation.

MiDAS comprises three key elements: (i) a Requirements Specification Language for Adaptive Systems (RSLAS); (ii) a Design Modeling Language For Adaptive Systems (DMLAS); and (iii) a Model Transformation Language for Adaptive Systems (MTLAS).

The remainder of this section explains the features of each language.

3.1 Requirements Specification Language for Adaptive Systems (RSLAS)

One of the fundamental objectives in a requirements specification is to clearly define the problem to be solved.

Adaptive systems differ from traditional systems, since they dynamically change their behavior, based on external inputs. These inputs, called adaptation parameters, typically include logical conditions based on user characteristics and context (Brusilovsky, 2001). Since adaptive systems are so dependent on these inputs, requirements tend to be specified differently than traditional systems.

The first challenge in adaptive software is the description of the adaptation parameters in the requirements specification. However, when software engineers develop adaptive software, they tend to specify adaptation parameters in the source code and not as part of the requirements specification (Cheng et al., 2009a). This situation has several drawbacks:

1. The lack of formalism in modeling adaptive systems. This originates that the development of adaptive systems are made, generally, on an *ad-hoc* manner.
2. The difficulty of porting the application across different architectures and platforms. This situation may increase the investment in time and resources in the software development process.
3. The complexity to validate the system. This due to that, for a user, is more complex understand a source code that a more abstract specification.
4. Adaptive parameters are closer to the problem domain that the solution domain. The source code is part of the solution domain, thus including adaptation parameters into the source code would unnecessarily tangle them with concerns of the solution domain.

Therefore, it is necessary that the requirements specification take into account adaptive parameters.

RSLAS will provide a new way to include adaptive parameters in requirements specifications. RSLAS will use a special language pattern in which traditional sentences as *"The system shall do this..."*, are replaced by conditional sentences, such as *"The system might do this..."*, "*But it may do this... as long as it does this...*", or "*The system ought to do this... but if it cannot, it shall eventually do this...*" (Cheng et al., 2009a).

RSLAS will depict the main elements in a requirements specification: goals, functional requirements, non-functional requirements, and storage information requirements. RSLAS also will provide a set of reusable patterns, taking into account that in software projects there are requirements that appear with higher frequency than others (Kopczynska and Nawrocki, 2014). These patterns may be depicted by reusable templates that represent repetitive tasks, such as CRUD operations or login processes.

Context plays an important role in adaptive software, since it provides useful information to cus-

tomize and personalize services. To enable context-aware adaptation, context profiles (e.g., connection, mobile device, location, etc.) and user profiles (e.g., personal data, interests, tastes, habits) could be specified with RSLAS.

Requirements elicitation in RSLAS relies on a structured requirements model and not on an informal document written in natural language. This is important because RSLAS is expected to facilitate translation, modeling, and relating stakeholder's expectations to adaptation requirements. For example, if a non-technical stakeholder needs to review and validate the requirements, RSLAS may use transformations to extract information from the structured requirement model and create a document in a language that the stakeholder understands. Additionally, with a structured model is easier to derive automatically design models.

To develop this language, RSLAS may be inspired by current languages that manage uncertainty, such as RELAX, complemented and integrated to approaches that provide support to model goals such as Tropos (Bresciani et al., 2004) or Kaos (Dardenne et al., 1993).

3.2 Design Modeling Language for Adaptive Systems (DMLAS)

DMLAS provides support to store information about static and dynamic components of adaptive software. Regarding to static elements, DMLAS store information about user, context, and additional profiles, which will be tightly related to the adaptation parameters defined by RSLAS.

Regarding behavior, DMLAS may use several approaches, such as Event-Condition-Action or BPMN, among others.

DMLAS should provide several design elements to realize the requirements specified by RSLAS. These design elements should abstract different approaches to provide adaptation, such as if-else statements, rule engines, expert systems, logic programming, constraint programming, etc. With DMLAS the designer should have the possibility to select and combine the most convenient elements to create the solution. DMLAS should also provide a design spanning several levels of abstraction (from those closer to the problem domain to those closer to implementation domain).

3.3 Model Transformation Language for Adaptive Systems (MTLAS)

Another challenge in adaptive software is to maintain traceability and synchronization between requirements, design and implementation, because traceability ensures that artifacts of subsequent phases of the development cycle are consistent.

Although current tools for transformation such as ATL or ETL provide mechanisms for traceability, these tools only provide unidirectional traceability(Macedo and Cunha, 2014).

An alternative of solution is to use bidirectional transformations (Czarnecki et al., 2009). Bidirectional transformations ensure that if some elements in a model (e.g., in design models) are modified, other models (e.g., requirements models) are automatically and consistently synchronized.

The main problem with current languages for bidirectional transformations (such as the Beanbag language (Xiong et al., 2009)) it that some information about the design may be lost, mainly because of the mappings from models to code and vice versa are not 1:1.

To fix that issue, MTLAS may provide bidirectional transformations to maintain traceability and synchronization between requirements, design, and implementation models.

3.4 Tool Support

Traditionally, CASE tools do not support the development process for adaptive software in a whole perspective: requirements specification, design, and implementation.

MiDAS will provide a CASE tool that supports the proposed process to develop adaptive software: requirements specification, design, and implementation. The CASE tool will support the requirements specification process and the design of adaptive applications. In addition, the tool will perform semi-automatic generation of adaptive applications from the models, using a code generator.

This tool depends on several technologies. For example, a visual workspace is necessary to create the requirements models. It is important to explore alternatives such as the Eclipse Graphical Modeling Project, Eclipse Graphiti, or EuGENia. From these alternatives, MiDAS may be use one or more, depending on the application context. Indeed, the definition of criteria of use is part of the future work of this project.

The tool should provide a workspace to customize and execute semi-automatic transformations to create

design models from the requirements models. These transformations will be developed using MTLAS.

The tool also should provide support to transform the models to an executable platform that supports adaptation at run-time. For these transformations, the tool should also use MTLAS.

Finally, the generated code by the case tool may be complemented with set of predefined libraries. Those libraries will be represented abstractly in the design model.

4 ANALYSIS OF ADAPTIVE SOFTWARE DEVELOPMENT WITH AND WITHOUT MiDAS

In previous work, the authors of this project have proposed an adaptive Virtual Learning Environment named ASHYI-EDU, which assists teachers to create especially-tailored plans for every student, taking into account their specific personalities, learning styles, competences, and abilities.

The development process in ASHYI-EDU was made in a traditional way. For example, the models were used just as a tool to capture ideas in brainstorming sessions. When the source code was modified, models were not updated automatically, i.e., code and models were not synchronized. Another issue is that many adaptive design decisions were placed directly into the code.

ASHYI-EDU was built using a Learning Management System (LMS), called Sakai. Because of that, the initial configuration of the development environment and the integration of adaptation concerns was a complex task, requiring to invest a considerable amount of time.

Last, but no least, migrating ASHYI-EDU to different LMS platforms, such as Moodle, would require a high amount of time and resources, because it would be necessary to develop the software using a different programming language and architecture.

Using MiDAS in the development of ASHYY-EDU models will fulfill an important role in the development process. The adaptive parameters (e.g., personalities, learning styles, competences, and abilities) will be specified directly in the models and not in the code; and thanks to the semi-automatic code transformations development process may be faster. Switching between platforms can be done in less time thanks to the abstraction provided by models. Additionally, the design decisions remain in models and may be modified not in code but models.

5 CONCLUSIONS AND FUTURE WORK

There are some approaches to bridge the gap between the requirements specification, design and implementation. However, none of them provide an effective process to generate the design from requirements, not only for traditional systems but adaptive systems. This position paper proposes a framework that supports the entire development process for adaptive software, spanning from requirements specification to design to implementation.

Ongoing work is the construction of this framework, which is divided in several activities. The first one is to define the languages for requirements specification and design. The second activity is to create parameterizable transformations between requirements, design, and implementation. The goal is to analyze and define the most recurrent scenarios and parameters in adaptive systems. The third activity is the construction of the CASE tool. Finally, it is necessary validate the framework through several case studies to evaluate the approach and get feedback.

ACKNOWLEDGEMENTS

This paper is part of the project "*ASHYI: Plataforma basada en agentes para la planificación dinámica, inteligente y adaptativa de actividades aplicada a la educación personalizada*", executed by the ISTAR research group of the Pontificia Universidad Javeriana, cofinanced by Colciencias, project 1203-569-33545.

REFERENCES

Ahmad, M., Araújo, J., Belloir, N., Bruel, J.-M., Gnaho, C., Laleau, R., and Semmak, F. (2013). Self-adaptive systems requirements modelling: Four related approaches comparison. In *Comparing Requirements Modeling Approaches Workshop*, pages 37–42. IEEE.

Andersson, J., De Lemos, R., Malek, S., and Weyns, D. (2009). Modeling dimensions of self-adaptive software systems. In *Software engineering for self-adaptive systems*, pages 27–47. Springer.

Beatty, J. and Chen, A. (2012). *Visual Models for Software Requirements*. O'Reilly Media, Inc.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., and Perini, A. (2004). Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, 8:203–236.

Brusilovsky, P. (2001). Adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 11(1-2):87–110.

Cheng, B. H., De Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., et al. (2009a). Software engineering for self-adaptive systems: A research roadmap. In *Software engineering for self-adaptive systems*, pages 1–26. Springer.

Cheng, B. H., Sawyer, P., Bencomo, N., and Whittle, J. (2009b). A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In *Model Driven Engineering Languages and Systems*, pages 468–483. Springer.

Cleland-Huang, J., Gotel, O. C. Z., Huffman Hayes, J., Mäder, P., and Zisman, A. (2014). Software traceability: Trends and future directions. In *Proceedings of the FOSE*, pages 55–69. ACM.

Czarnecki, K., Foster, J. N., Hu, Z., Lämmel, R., Schürr, A., and Terwilliger, J. F. (2009). Bidirectional transformations: A cross-discipline perspective. In *Theory and Practice of Model Transformations*, pages 260–283. Springer.

Dardenne, A., Van Lamsweerde, A., and Fickas, S. (1993). Goal-directed requirements acquisition. *Science of computer programming*, 20(1):3–50.

Dromey, R. G. (2003). From requirements to design: Formalizing the key steps. In *Conference on Software Engineering and Formal Methods*, pages 2–11. IEEE.

Durán, A., Bernárdez, B., Toro, M., Corchuelo, R., Ruiz, A., and Pérez, J. (1999). Expressing customer requirements using natural language, requirements templates and patterns. In *Proceedings of IMACS/IEEE CSCC*.

Esfahani, N. and Malek, S. (2013). Uncertainty in self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II*, volume 7475, pages 214–238. Springer.

Espada, P., Goulão, M., and Araújo, J. (2011). Measuring complexity and completeness of kaos goal models. In *Empirical Requirements Engineering (EmpiRE), 2011 First International Workshop on*, pages 29–32. IEEE.

Fleurey, F. and Solberg, A. (2009). A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems. In *Model Driven Engineering Languages and Systems*, pages 606–621. Springer.

Gnaho, C., Semmak, F., and Laleau, R. (2013). An overview of a sysml extension for goal-oriented nfr modelling: Poster paper. In *Conference on Research Challenges in Information Science*, pages 1–2. IEEE.

Greenwood, P., Chitchyan, R., Rashid, A., Noppen, J., Fleurey, F., and Solberg, A. (2011). Modelling adaptability and variability in requirements. In *Requirements Engineering Conference*, pages 343–344. IEEE.

Kent, S. (2002). Model-driven engineering. *Lecture Notes in Computer Science*, 2335:286–298.

Kopczynska, S. and Nawrocki, J. (2014). Using non-functional requirements templates for elicitation: A case study. In *Requirements Patterns (RePa), 2014 IEEE 4th International Workshop on*, pages 47–54.

Laleau, R., Semmak, F., Matoussi, A., Petit, D., Hammad, A., and Tatibouet, B. (2010). A first attempt to combine sysml requirements diagrams and b. *Innovations in Systems and Software Engineering*, 6(1-2):47–54.

Lamsweerde, A. V. (2009). *Requirements engineering: from system goals to UML models to software specifications*. Wiley.

Macedo, N. and Cunha, A. (2014). Least-change bidirectional model transformation with qvt-r and atl. *Software & Systems Modeling*, pages 1–28.

OMG (2015). Sysml specification. http://www.sysml.org/docs/specs/OMGSysML-v1.3-12-06-02.pdf. Last accessed: January/2015.

Salehie, M. and Tahvildari, L. (2009). Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):14.

Toro, A. D., Jiménez, B. B., Cortés, A. R., and Bonilla, M. T. (1999). A requirements elicitation approach based in templates and patterns. In *WER*, pages 17–29.

Vogel, T. and Giese, H. (2014). Model-driven engineering of self-adaptive software with eurema. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 8(4):18.

Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H., and Bruel, J.-M. (2010). Relax: a language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering*, 15(2):177–196.

Xiong, Y., Hu, Z., Zhao, H., Song, H., Takeichi, M., and Mei, H. (2009). Supporting automatic model inconsistency fixing. In *Symposium on The foundations of software engineering*, pages 315–324. ACM.