CC BY

Towards a Domain-Specific Language to Design Adaptive Software: the DMLAS Approach[1]

Hacia un lenguaje específico de dominio para el diseño de *software* adaptativo: la aproximación DMLAS[2]

José Bocanegra García[3]
Jaime Pavlich-Mariscal[4]
Angela Carillo-Ramos[5]

doi:10.11144/Javeriana.iyu20-2.tdsl

How to cite this article:
J. Bocanegra García, J. Pavlich-Mariscal, A. Carrillo-Ramos, "Towards a domain-specific language to design adaptive software: the DMLAS approach," *Ing. Univ.*, vol. 20, no. 2, pp. 335-354, 2016. http://dx.doi.org/10.11144/Javeriana.iyu20-2.tdsl

---

[1] Submitted on: November 19th, 2015. Accepted on: March 8th, 2016. This article is derived from an investigation project named *MiDAS: A MDD Approach for Adaptive Systems*, code SIAP 00006686, developed by the ISTAR research group at the Pontificia Universidad Javeriana, Bogotá, Colombia.

[2] Fecha de recepción: 19 de marzo de 2015. Fecha de aceptación: 8 de marzo de 2016. Este artículo se deriva de un proyecto de investigación denominado *MiDAS: una aproximación dirigida por modelos para sistemas adaptativos*, código SIAP 00006686, desarrollado por el grupo de investigación ISTAR de la Pontificia Universidad Javeriana, Bogotá, Colombia.

[3] Ingeniero de sistemas, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia. DEA en Tecnología e Ingeniería de Software, Universidad de Sevilla, España. Estudiante del Doctorado en Ingeniería, Pontificia Universidad Javeriana, Bogotá, Colombia. E-mail: jose_bocanegra@javeriana.edu.co.

[4] Ingeniero Civil en Computación e Informática, Universidad Católica del Norte, Chile. Licenciado en Ciencias de la Ingeniería, Universidad Católica del Norte. PhD in Computer Science & Engineering, University of Connecticut, USA. Profesor asociado, Pontificia Universidad Javeriana, Bogotá, Colombia. E-mail: jpavlich@javeriana.edu.co.

[5] Ingeniera de sistemas y computación, Universidad de los Andes, Bogotá, Colombia. Maestría en Ingeniería de Sistemas y Computación, Universidad de los Andes. Doctorat en Informatique, Universite de Grenoble I, Francia. Profesora titular, Pontificia Universidad Javeriana, Bogotá, Colombia. E-mail: angela.carrillo@javeriana.edu.co

Abstract

An adaptive software has the ability to modify its own behavior at runtime due to changes in the users and their context in the system, requirements, or environment in which the system is deployed, and thus, give the users a better experience. However, the development of this kind of systems is not a simple task. There are two main issues: (1) there is a lack of languages to specify, unambiguously, the elements related to the design phase. As a consequence, these systems are often developed in an *ad-hoc* manner, without the required formalism, augmenting the complexity in the process of derivation of design models to the next phases in the development cycle. (2) Design decisions and the adaptation model tend to be directly implemented into the source code and not thoroughly specified at the design level. Since the adaptation models become tangled with the code, system evolution becomes more difficult. To address the above issues, this paper proposes *DMLAS*, a Domain-Specific Language (*DSL*) to design adaptive systems. As proof of concept, this paper also provides a functional prototype based on the Sirius plugin for Eclipse. This prototype is a tool to model, in several layers of abstraction, the main components of an adaptive system. The notation used both in the models and the tool was validated against the nine principles for designing cognitively effective visual notations presented by Moody.

Keywords

adaptation; adaptive software; context; design; domain-specific language; model-driven engineering; notation

Resumen

Un *software* adaptativo es capaz de modificar su comportamiento en tiempo de ejecución debido a cambios en el sistema, en los requisitos o en el entorno en el que se despliega. La importancia del software adaptativo radica en el hecho de que puede ajustar su propio comportamiento a diferentes entornos y contextos, y por lo tanto, dar a los usuarios una mejor experiencia. Sin embargo, el desarrollo de sistemas adaptativos no es una tarea sencilla, por dos inconvenientes: 1) faltan lenguajes para especificar los elementos relacionados con la fase de diseño. Como consecuencia, estos sistemas se desarrollan a menudo en una manera ad-hoc, sin el formalismo requerido, dificultando el proceso de derivación de modelos de diseño para las siguientes fases del ciclo de desarrollo.2) las decisiones de diseño y el modelo de adaptación tienden a ser implementados directamente en el código y no se especifican a nivel de diseño. Cuando los modelos de adaptación se acoplan directamente con el código, la evolución del sistema se vuelve más difícil. Para hacer frente a los problemas mencionados, este artículo propone *DMLAS*, un lenguaje específico de dominio para el diseño de sistemas adaptativos. Como prueba de concepto, este artículo proporciona un prototipo funcional basado en el *plugin* Sirius para Eclipse. El propototipo desarrollado es una herramienta que permite modelar, en varios niveles de abstracción, los principales componentes de un sistema adaptativo. La notación usada tanto por los modelos como por la herramienta ha sido validada de acuerdo con los nueve principios formulados por Moody.

Palabras clave

adaptación; contexto; ingeniería dirigida por modelos; lenguages específicos de dominio; notación; software adaptativo

Introduction

Adaptive software has the ability to modify its behavior at runtime due to changes in the system, requirements, or the environment in which it is deployed [1]. Adaptive software plays an important role in several scenarios. One of them is when there are ever-changing environments with fluctuating network resources and availability [2]. For example, if a device has a slow connection to the network, an adaptive software could modify its behavior at runtime and change the format of presentation of information (*e.g.,* from graphical to textual) to give the user a better experience. Another scenario is when users with different skills, knowledge, and preferences interact with the software. For example, an adaptive Learning Management System may adjust the contents presented to the students considering their cognitive skills or prior knowledge, and thus, improve the educational process.

Adaptive software tends to share the same essential elements of traditional software. The software receives a request from the user, performs a process, and generates a response. However, adaptive systems need to take into account the context. According to Bauer and Dey, a widely accepted definition to context is "any information that can be used to characterize the situation of an entity" [3]. In other words, context involves places or objects that are relevant to the interaction between users and applications. Context, which also includes users and applications, can be denoted through profiles. Profiles, often depicted as class diagrams, are a description of the information of users, applications, and other context elements.

The main difference between adaptive software and traditional software is that adaptive software must take into account the context to provide users with specially-tailored information. In addition, adaptive software should also be able to alter the way it processes the information to convey to the users. All of these elements: context, content, presentation format, and process may be defined as adaptive characteristics.

Due to the importance of adaptive software, it is necessary to improve the software development process of these kinds of systems. However, the development of adaptive software is a task far from trivial. De Lemos *et al*. [4] have identified the main problems of software engineering for the development of adaptive systems. One of them is related to the design process. In this field, the main problem is the difficulty for designing an adaptive system based on the specified requirements. Existing modeling languages do not adequately represent adaptation concepts, such as profiles, presentation, navigation, and content models. Although general-purpose languages, such as Unified Modeling Language (UML) can be utilized to specify adaptive systems, they lack adaptation-specific concepts. This is an issue that hinders the design process of adaptive systems. As stated by Kosar, Bohra, and Mernik [5], a modeling language is more effective if it is able to seamlessly specify concepts from the domain of the problem. Therefore, the design process of adaptive systems could be significantly improved if designers could utilize languages that could model adaptation-specific concepts.

To address the above issue, this paper proposes *Design Modeling Language for Adaptive Systems* (*DMLAS*), a domain-specific language to design adaptive systems. This paper is an extension of a previously published paper [6]. In that previous paper, the authors propose a preliminary version of *DMLAS* and a prototype developed in the Generic Modeling Environment [7]. However, that prototype was a simple tool utilized to perform an initial proof-of-concepts.

In this extended paper, the authors propose the following new contributions: (i) a modification to the metamodels that support *DMLAS*, taking as reference the concepts provided by Dataflow Programming [8]; and (ii) a new prototype based on the Sirius framework [9].

The remainder of this paper is structured as follows. Section 1 provides the related work. Section 2 describes *DMLAS*. Section 3 illustrates the developed prototype. Section 4 describes the first steps to validate the metamodel and the notation of *DMLAS*. In Section 5, the authors provide a discussion about the advantages of *DMLAS* compared to other approaches. This paper ends with the conclusions and the explanation of the future work.

1. Related Work

Several authors discuss about the importance and the key elements in the design of adaptive systems. According to Brun *et al.* [10], an adaptive system should be developed according to a conceptual model of adaptation, without considering technologies and tools for its implementation. However, often the

models for adaptive systems are represented implicitly in the form of domain knowledge or the engineer's expertise. This makes the development process of these systems harder.

According to De Lemos *et al.* [4], one of the challenges in the development of adaptive systems is the need to define models at the design space that can represent a wide range of system properties. The authors explain that there is a lack of systematic studies of the overall design space for adaptive systems.

Baude, Henrio, and Ruz [11] consider that the process to develop adaptive systems can be accomplished in two ways: extending existing programming languages/systems or defining new adaptation languages. Although numerous research efforts have investigated both solutions and their combination, there is still a lack of powerful languages, tools, and frameworks that could help realize adaptation processes in a systematic manner.

In [12], Gamez, Fuentes, and Troya provide the main elements to take into account in the design of adaptive systems. These elements cover items such as the context definition, the change in the context, the device heterogeneity, and the languages issues. However, authors do not detail the particularity of design specification that covers the mentioned elements.

The literature provides several options to model the design of adaptive systems. Table 1 presents a comparison of each proposal with the following criteria: (i) explicit separation of concerns (in several types of adaptation), (ii) design approaches, (iii) use of a domain-specific language (DSL) and (iv) support for reusable models.

Table 1. Comparison of proposals

Criteria of comparison	Work				
	[13]	[14]	[15]	[16]	DMLAS
Separation of concerns			±	-	+
Design approach	?	MAS	Object oriented	Data flow oriented	Data flow oriented
Use a DSL		+	+	-	+
Models reuse	+	+	+	-	+

+: denotes that proposal covers the related item; —: denotes that proposal does not cover the related item. ?: it is not clear the way the approach addresses a certain criterion; ±: partial support for the criterion.
DSL: domain-specific language
Source: authors' own elaboration

Authors such as Berkane, Seinturier, and Boufaida [13] have proposed a set of adaptation-oriented design patterns that support the development of adaptive systems. These design patterns provide reusable models that can be instantiated across different domains, thus facilitating the reuse of adaptation expertise. However, this proposal does provide an explicit classification of adaptation types, so they cannot adequately separate concerns in an adaptive application.

Mao *et al.* [14] model adaptive systems as a multi-agent organization, based on agent technology and an organizational metaphor. To these authors, an agent is considered an autonomous entity situated within the environment to satisfy the design objectives. An adaptive system is modeled as a Multi-Agent System (MAS) organization, consisting of various roles and agents in the organization context, that defines their environments. Roles are the abstract classification of the behavior and environment of the agents in the organization. An agent can play multiple roles and a role can be played by multiple agents. However, this approach is tightly related to agent-based design, which makes it difficult to specify a design utilizing other approaches. In addition, authors do not consider the concepts of content adaptation, navigation, or presentation, which makes it harder to adequately separate concerns in an adaptive application.

Authors such as Vogel and Giese [15] have studied the use of a Model-Driven Engineering approach in adaptive software. These authors propose *EUREMA*, a model-driven approach to develop adaptive software. *EUREMA* provides models to two levels of abstraction: an architectural view and a behavioral view. However, *EUREMA* is only oriented to the specification and execution of adaptation engines, and it does not cover elements of design of adaptive systems.

Gelogo and Kim [16] propose a development process for adaptive ubiquitous learning systems. Authors suggest three activities for the design stage: the design of user/admin user interface, the design of the story board, and the design of a database. However, authors do not provide details about separation of concerns, reuse of models, or the use of a *DSL*.

2. The Design Modeling Language for Adaptive Systems

This section describes the essential elements of the Design Modeling Language for Adaptive Systems (*DMLAS*). To illustrate *DMLAS*, this paper uses an example of an adaptive system for prenatal care, called Prenat. The goal of Prenat is to provide pregnant women and their families with specially-tailored information about the gestation process and to assist them during the prenatal stage.

Prenat offers to users several services. Some of them are the following: provide a list of hospitals nearest to the current location of the user; provide a list of maternity clothing stores filtered by user brand preferences; and provide a list of general and obstetrician practitioners. Prenat uses the information about profiles and context to adapt the services.

To better explain the rationale behind *DMLAS*, it is important to understand the way adaptive software is traditionally modeled. In the context of adaptive software development, software engineers often divide the adaptation model into two groups: presentation and navigation. The former adapts the information and the format in which it will be presented to the user. The latter adapts the sequence of the interaction between the user and the system, which is usually reflected in a system's menu [17].

In contrast, *DMLAS* separates presentation adaptation into two individual concepts. The first is *content adaptation*, to adapt the information to convey to users, regardless of the format in which the information will be presented to them. The second is called *presentation adaptation*, but unlike the type of adaptation with the same name that is used by other researchers, in *DMLAS* it refers specifically to the format in which the information will be presented to users. *DMLAS* also addresses *navigation adaptation*, but separates it into information-related and presentation-related navigation. The former addresses the options available to each user at every specific time in the interaction with the system. The latter addresses the format in which those options will be presented to users.

The above concepts reflect an essential premise behind *DMLAS*: Adaptation should separate information from presentation as different concerns. We believe this premise is important, since in practice these two concerns can often be addressed separately. Separation of concerns in this context has two main benefits: design simplicity, because software engineers may avoid combining heterogeneous concepts in a single model; and, maintainability, because potential changes are located in specific parts of the models. The following are some concrete examples of the pertinence of separation of concerns.

- Users can utilize access devices with heterogeneous features to access an adaptive system. For example, a pregnant woman may access to Prenat from a mobile device with a 3G connection, while another user may access Prenat through a desktop device with a broadband connection. The former user has a device with limited screen size and low connection speeds, and may prefer to receive the information in textual format. The latter user may prefer a graphical format complemented with other options (*i.e.*, video, audio, etc.).

Therefore, it is important to adjust the format presented to each device, regardless of the specific information that is sent to the user.

- Users, according to their likes and preferences, may require that certain pieces of information have more priority than others. For example, a pregnant woman in her first pregnancy may require more detailed information about pregnancy advices and delivery preparation than a woman who has had more than one pregnancy. In another case, if a woman will give birth by C-section, she may require more detailed information about this surgical procedure than another woman who will give birth by natural means. The priority given to pieces of information is largely independent of the format in which the information is presented.

In addition, the user interface layer is considered one of the key components of software applications since it connects their end-users to the functionality [18]. For instance, a user with visual disabilities may prefer textual information in a font with a greater size than a user with normal visual acuity. Since user interface is a complex and important concern, it might be desirable to separate it from the information concern.

Based on the above rationale, a *DMLAS* model comprises the following two key elements: (i) the profiles model, and (ii) the process model. The remainder of this section utilizes the Prenat example to explain the main components of *DMLAS*.

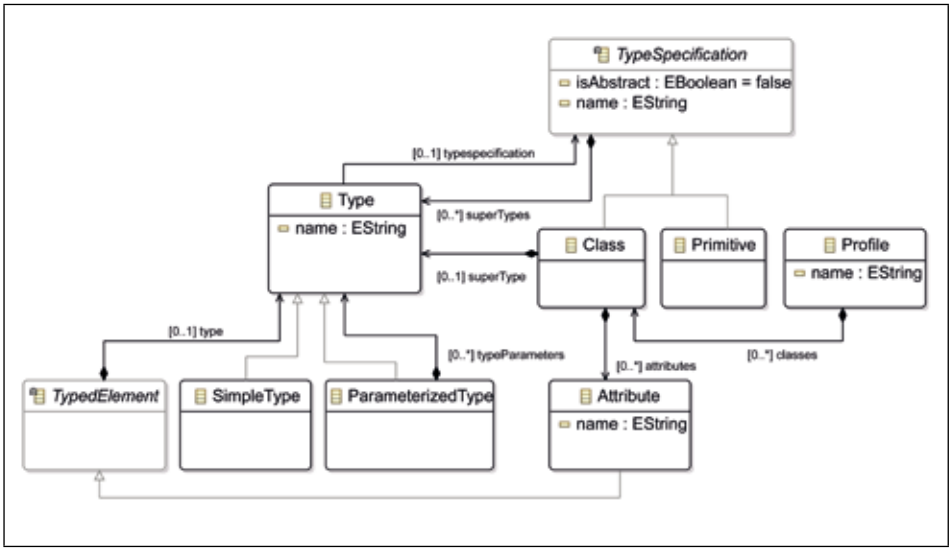
2.1. Profiles Model

The profiles model represents the profiles utilized to perform the adaptation. Figure 1 depicts the elements in the profiles metamodel. The main element is the *Profile* class which represents the elements surrounding the system (context). In *DMLAS* the information related to profiles is modeled using a set of classes. Each *Class* has a set of attributes. Each attribute has a name and a type. To model types, *DMLAS* uses a class named *TypeSpecification*. That class covers concepts such as *Class* and *Primitive*. The *Primitive* class represents the basic types of the language (*i.e.*, integer, string, boolean, and float). *DMLAS* also utilizes a *ParameterizedType* to perform an instantiation of a generic type with type arguments.

Figure 2 depicts an example of a profiles model in the context of Prenat. The figure is a top-level diagram in *DMLAS* that shows three profiles that represents the *User*, the *Device* by which the user accesses to the system, and the *Context*.

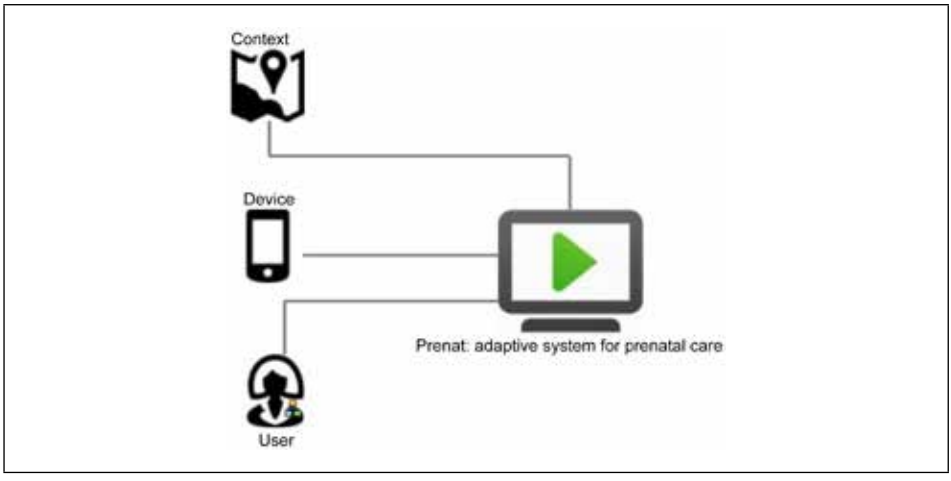
These profiles provide information to the adaptive system. Figure 3 depicts a low-level diagram of the specific information about the user. In that level, *DMLAS* represents profile information as a set of classes with attributes and types. Table 2 describes the components utilized in the profiles model.

Figure 1. Profiles metamodel



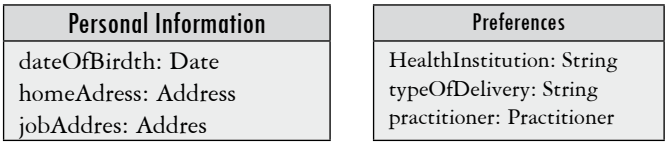
Source: authors' own elaboration

Figure 2. Profiles model: general view



Source: authors' own elaboration

Figure 3. Profiles model: detailed view



Source: authors' own elaboration

Table 2. Profiles model components

Icon	Class	Description
	AdaptiveSystem	Represents the adaptive system in a high level of abstraction
	Profile	Represents the information that may be utilized to characterize the surroundings of an adaptive system
	Class	Represents specific information about profiles

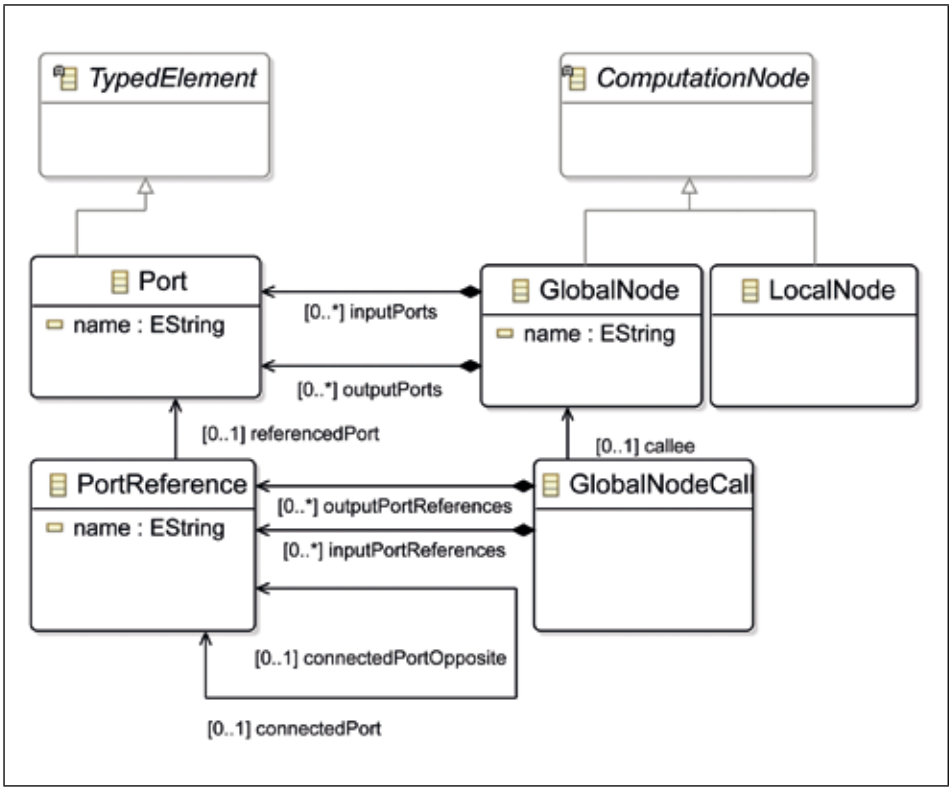
Source: authors' own elaboration

2.2. Process Model

The process model shares the same infrastructure to address the design of content, presentation, and navigation adaptation. The difference between components is the set of operations for each one. In *DMLAS*, these operations are represented as a set of nodes and ports. These concepts are taken from visual programming languages such as Dataflow Programming [8]. This approach was selected because adaptive software may be modeled as a set of nodes that receives a set of inputs through of input ports, performs a process of adaptation, and delivers a result through of output ports.

The process metamodel is divided in two parts. The first part depicts the concepts related to nodes and ports, and the second part depicts the concepts related to expressions. Figure 4 depicts the first part or the process metamodel. The main concept in this metamodel is the *ComputationNode*, which is analogous to a function in traditional programming languages. There are two types of computation nodes: *LocalNode* and *GlobalNode*. A local node is one whose scope is restricted locally to that section in which is declared. This means that this node will only be able to be manipulated in that section and may not be referenced in other contexts. In contrast, a global node is accessible from any section of the model and its purpose is to define the primitive adaptation operations that need to be utilized anywhere in the model.

Figure 4. Process Metamodel: Data Flow



Source: authors' own elaboration

In the same way of several programming languages, in *DMLAS* there is a distinction between the definition of a function (or a computational node) and its invocation. *GlobalNode* is the definition of a global node, and *GlobalNodeCall* is the invocation to that node.

A *ComputationNode* has a set of ports. These ports are utilized as input or output of data (similarly as input parameters or return values in a function). The class *Port* provides the definition of ports (or formal parameters), while the *PortReference* class provides the reference to these ports. The latter is utilized at the computation node calls to reference the ports of the computation node being called.

Although content, presentation, and navigation adaptation share the same metamodel, they differ in the particular computation node instances they utilize as their primitive operations. These operations are the following:

Content adaptation operations. (i) *TopN*: select the n-Th term in a list; (ii) *Sort*: sort a list; (iii) Filter: filter a list; (iv) *Union*: performs a union between two or more data sources; (v) *Intersection*: performs an intersection between two or more data sources; (vi) Split: split a data source into two or more lists; (vii) *DataSource*: represents a source of information to be adapted; and (viii) *DataResult*: represents the adapted information as result of a process of adaptation.

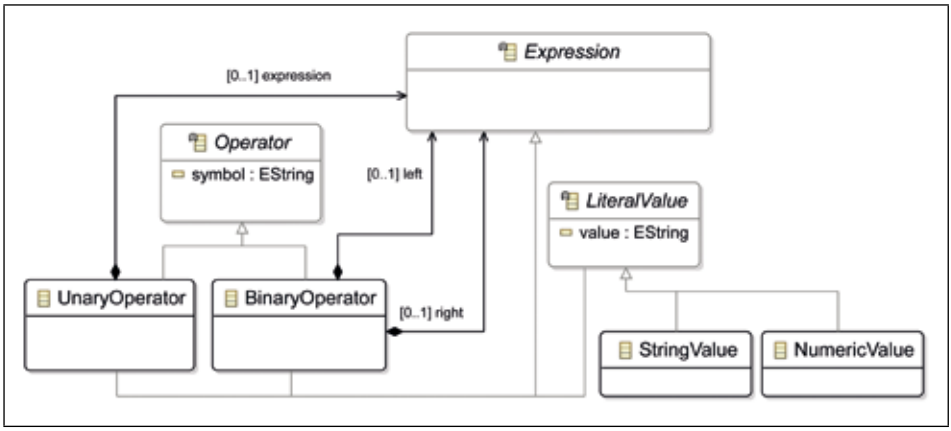
Presentation and navigation operations. (i) *Audio*: turns an information into audio; (ii) *Video*: turns an information into video; (iii) *Text*: turns an information into text; (iv) *Graph*: Turns an information into a graphic; (v) *Map*: turns an information into a map; (vi) *Lab*: Turns an information into a virtual lab; (vii) *Highlight*: highlights a set of elements: (viii) *Dim*: dims a set of elements; (ix) *PresentationAdapter*: takes the source information to determine the priority of each element in the web site; (x) *DecisionTree*: evaluates several alternatives to decide what kind of adaptation perform over a source of information (e.g., choose a textual information over a graphical information taking into account the information provides by profiles); (xi) W*idget*: represents the information to be presented in a user interface; and (xii) *Menu*: the set of options adapted according to the user and context preferences.

The second part or the adaptation metamodel is related to expressions, which is depicted in Figure 5. In *DMLAS*, an expression is a combination of literal values (*StringValue* and *NumericValue*), operators (*UnaryOperator* and *BinaryOperator*), and calls to computation nodes. Figures 6, 7, and 8 depict some examples of models for content, presentation, and navigation adaptation.

Figure 6 depicts the process to adapt a list of practitioners. The content model utilizes five operations: *DataSource*, *Filter*, *TopN*, *Sort*, and *DataResult*. The operation *DataSource* takes as input the required source of information to retrieve the data to be adapted. In this example, the *DataSource* retrieves the information of a table, called Practitioners, which is stored in a data base. That criterion is provided to the respective port by an expression. The criterion expression is specified in the properties view of the developed tool (as depicted at the bottom part of the Figure 9). The output of that operation is a list of practitioners.

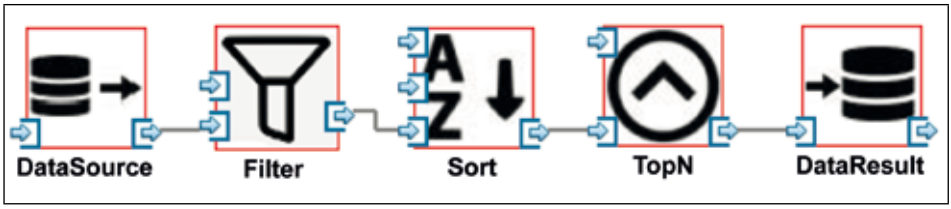
The operation *Filter* takes as input the list of practitioners, and the criteria to perform the filter, in this case, practitioners with female gender. The filtered list is taken as input in the operation *Sort*. That operation has two additional input parameters: the order criteria (ascendant or descendent), and the field to perform the order (for example, the name of the practitioner). The output of that operation is a sorted list.

Figure 5. Process metamodel: expressions



Source: authors' own elaboration

Figure 6. Process model (for content adaptation)



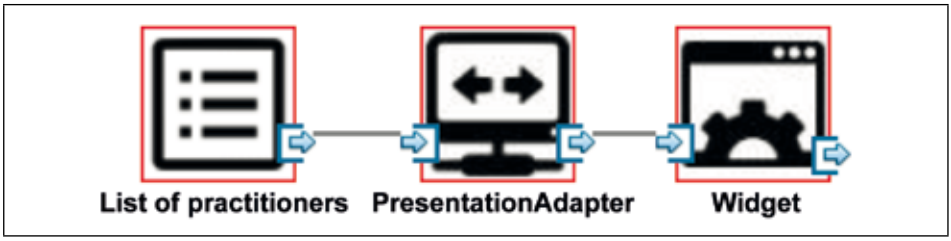
Source: authors' own elaboration

The *TopN* operation takes as input the sorted list, and the *N* value to limit the list. The output of the *Sort* operation is a list of *N* practitioners. Figure 7 is a presentation model for Prenat. A presentation model has several levels of abstraction. For space considerations, this example only provides one level of abstraction. The figure shows an element called *PresentationAdapter*. This element takes as input the adapted list of practitioners (which are the output of information model presented in Figure 6). The output of this model is a *Widget* that represents the information to be presented in a user interface.

In a *PresentationAdapter*, the software engineers may use operations such as a *DecisionTree* to determine the priority of the formats to be used to represent the information. For example, if the user accesses to Prenat through a mobile device the system will present the information in a textual format. Otherwise, if the user accesses to Prenat through a PC, the system will present the information in a graphical format. The *DecisionTree* also may be utilized to determine the elements

that may be "highlighted" (shown more prominently) or "dimmed" (shown less prominently) in the user interface.

Figure 7. Process model (for presentation adaptation)



Source: authors' own elaboration

Figure 8 depicts a navigation adaptation for Prenat. The model represents a reference to the three services provided by Prenat. The node *Union* combines the three lists of data (practitioners, hospitals, and stores). The result of this type of adaptation is an adapted menu. It is important to note that this menu is "abstract" in the sense that, at this point, the system has not yet decided the way to represent that menu to the user (presentation). The symbols utilized for content, presentation, and navigation adaptation are depicted in Table 3.

3. Prototype

To validate the language, we developed a prototype based on the Sirius [9] plugin for Eclipse *IDE*. Figure 9 depicts a screenshot of the prototype made in Sirius. On the left side of the screen is the Model Explorer that provides a tree view of the models. The center contains the canvas that depicts the models. The right part of the screen contains the tools palette utilized to create the elements of the model. At the bottom of the screen is the properties view, utilized to graphically edit the properties of the elements in the model.

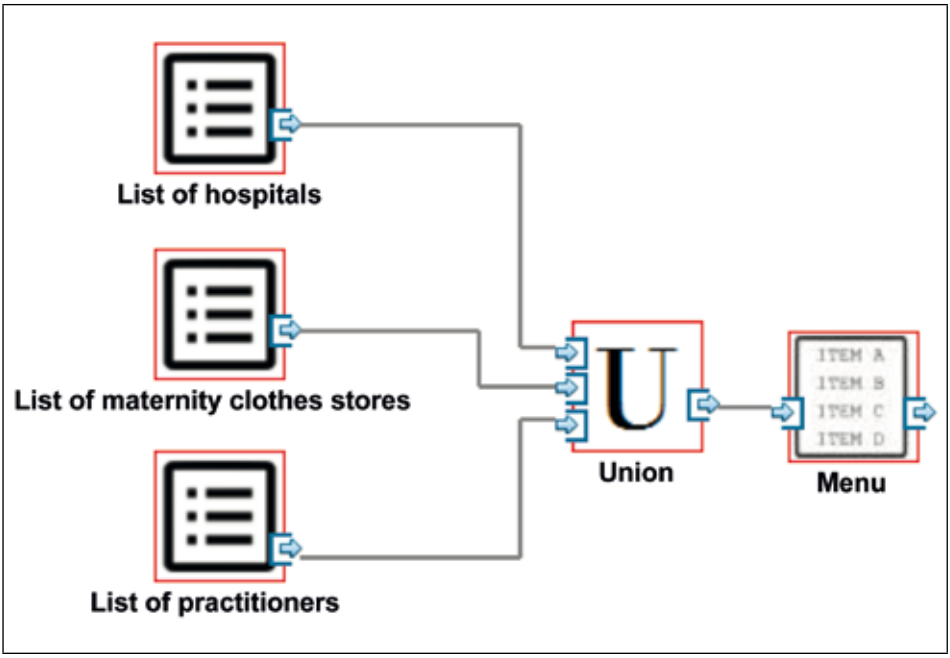
4. Validation of the Language

To validate the notation utilized in *DMLAS* this paper takes into account the 9 principles for designing cognitively effective visual notations presented by Moody in [19]. Principles 1 to 5 are fully addressed, while principles 6 to 9 are covered partially. Future work derived from this proposal is to assure the entire completion of those principles.

Complexity management: the ability of a visual notation to represent information without overloading the human mind. *DMLAS* uses models distributed

across hierarchical levels of abstraction, which avoids the overload of elements in a single model.

Figure 8. Process model (for navigation adaptation)



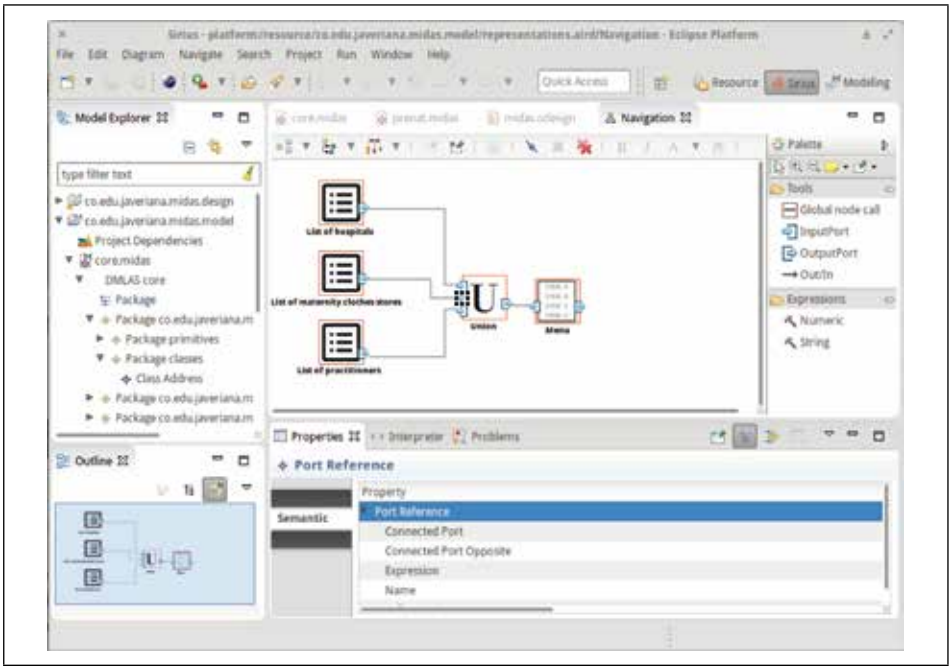
Source: authors' own elaboration.

Table 3. Process Model Components

Icon	Class	Description
	GlobalNodeCall	Represents the call to a global node
	PortReference (inputValues)	Represents the reference to an input port
	PortReference (outputValues)	Represents the reference to an output port

Source: authors' own elaboration

Figure 9. Case tool: prototype in Eclipse Sirius



Source: authors' own elaboration

Cognitive integration: applies when multiple diagrams are used to represent a system. *DMLAS* uses mechanisms to simplify navigation and transitions between diagrams and help the reader to assemble information from separate diagrams into a coherent mental representation of the system.

Graphic economy: the number of graphical symbols in a notation. *DMLAS* uses a maximum of 12 graphical symbols for each diagram. This is a small number compared to *UML* Class Diagrams that have a graphic complexity of over 40.

Dual coding: using text and graphics together to convey information is more effective than using either on their own. Each element in *DMLAS* has its corresponding legend in text below the graphical representation.

Semiotic clarity: there must be a one-to-one correspondence between symbols and their referent concepts. As shown in Table 1 and Table 2, each symbol has a one-to-one correspondence with a metamodel concept; none of the concepts is represented by multiple graphical symbols; there are no graphical symbols that do not correspond to any semantic construct; and there are no semantic constructs that are not represented by any graphical symbol. Although there is some overload in some symbols (for example, some elements may be depicted as nodes

in a diagram or as ports of other nodes), this overload is required to depict the model at several levels of abstraction.

Perceptual discriminability: the ease and accuracy with which graphical symbols can be differentiated from each other. In *DMLAS* each symbol uses its own graphical representation and they are distinguishable from others. However the degree of clarity and simplicity it is not fully demonstrated, because it requires an experimental validation. This validation is also part of the future work.

Semantic transparency: the extent to which the meaning of a symbol can be inferred from its appearance. In DMLAS the visual metaphors applied to each of the elements are representative, and its semantic is the closest possible to its intended meaning. However, validating this principle also requires an experimental study.

Visual expressiveness: the number of visual variables used in a notation. Some diagrams in *DLMAS* contain more than 12 visual elements. This means that the notation may be classified as visually saturated. In addition, *DMLAS* uses only one visual variable (the shape) to distinguish the elements in the diagram. Future work includes supporting more visual variables (*e.g.*, color, size, brightness, orientation, texture, and position).

Cognitive fit: use different visual dialects for different tasks and audiences. In the current specification of *DMLAS*, the designer may change the icons that represent each element. This can be utilized to address different kinds of users. The use of dual coding (text and graphics) also supports the representation of information for different tasks and audiences. Nevertheless, this principle also requires further experimental validation.

5. Discussion

The main contributions of this paper, compared to other related works are:

- The use of a new refined metamodel. This metamodel covers the main elements in adaptive systems: content, presentation, and navigation. The main characteristics of this metamodel compared with another approaches are the following: (i) the use of a paradigm inspired in dataflow programming. In this paradigm, the components of an adaptive system may be modeled as a set of nodes, connections, and data flows; (ii) an extensive representation of profiles model with complementary elements such as type specifications and parameterized types, and (iii) a set of elements to model expressions (*e.g.*, logical, arithmetical). Those features provide to *DMLAS* with a greater degree of expressiveness and easeness of use due to its graphical representation.

- The development of an updated prototype using the Sirius plugin for the Eclipse *IDE*. The main advantage of this prototype is that the models designed with the tool may be used in the next activities in the development process. For example, models may be transformed into other models using transformation languages, or may be transformed to code using code generators.
- The partial validation of the language against the Moody's principles for graphical notations. As visual notations are an integral part of the languages in software engineering, the validation of notation ensures that the graphical representation of the models facilitates the communication of ideas between practitioners and helps convey information more effectively to nontechnical people.

An area in which is necessary to complement the developed work is the one related to validation. It is important to develop a more extensive bank of tests in several scenarios to probe the expressiveness of the proposed language. In addition, it is necessary to cover completely the principles provided by Moody to assure the effectiveness of the visual notation used in *DMLAS*.

Conclusions and Future Work

This paper presented *DMLAS*, a *DSL* to visually specify the design of adaptive systems. *DMLAS* focuses on separating different adaptation concerns: information, navigation, and presentation, and it provides a hierarchical representation of interrelated models, which should simplify the understanding and evolution of such design.

Future work is related to two main activities: (i) the creation of a textual language for *DMLAS*, and (ii) the development of transformations between models, using a transformation language that automates the transition from design models to code. That future work has the following advantages. First, a textual language for *DMLAS* addresses the specification, in a simple way, of components such as expressions, which are more complex to specify in a visual language. Second, a proper automation can accelerate the implementation of most software systems, ensuring that their code follows good development practices, and reduces the chance of programming errors.

References

[1] K. Geihs and C. Evers, "User intervention in self-adaptive context-aware applications," in *Proc. Australasian Computer Science Week Multiconference*, 2016.

[2] C. Krupitzer, G. Schiele, S. VanSyckel, F. M. Roth. and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive Mob. Comput.,* vol. 17, pp. 184-206, 2015.

[3] C. Bauer and A. K. Dey, "Considering context in the design of intelligent systems: Current practices and suggestions for improvement," *J. Syst Soft.,* vol. 112, pp. 26-47, 2016.

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

[5] T. Kosar, S. Bohram and M. Mernik, "Domain-Specific Languages: A systematic mapping study," *Inform Software Tech.,* vol. 71, pp. 77-91, 2016.

[6] J. Bocanegra, J. Pavlich-Mariscal, and A. Carrillo-Ramos, "DMLAS: A domain-specific language to specify the design of adaptive systems," in *10CCC*, 2015.

[7] Institute for Software Integrated Systems (ISIS), "Generic Modeling Environment – GME," [Online]. Available: http://www.isis.vanderbilt.edu/projects/gme/. [Accessed March 2016].

[8] H. Wei, S. Zuckerman, X. Li and G. Gao, "A dataflow programming language and its compiler for streaming systems," in *Inte. Conf. Computational Sci.*, vol. 29, pp. 1289-1298, 2014.

[9] Eclipse Foundation, "Sirius," [Online]. Available: https://eclipse.org/sirius/. [Accessed March 2016].

[10] Y. Brun, R. Desmarais, K. Geihs, M. Litoiu, A. Lopes, M. Shaw, and M. Smit, "A design space for self-adaptive systems," in *Software Engineering for Self-Adaptive Systems*. Berlin: Springer, 2013, pp. 33-50.

[11] F. Baude, L. Henrio, and C. Ruz, "A component-based programming model for autonomic applications," *Programming Distributed and Adaptable Autonomous Components—the GCM/ ProActive Framework,* vol. 45, no. 9, pp. 1189-1227, 2015.

[12] N. Gamez, L. Fuentes, and J. M. Troya, "Creating self-adapting mobile systems with dynamic software product lines," *IEEE Software,* vol. 32, no. 2, pp. 105-112, 2015.

[13] M. Berkane, L. Seinturier and M. Boufaida, "Using variability modelling and design patterns for self-adaptive system engineering: Application to smart-home," *Int. J. Web Eng. Technol.,* vol. 10, no. 1, pp. 65-93, 2015.

[14] X. Mao, M. Dong, L. Liu, and H. Wang, "An integrated approach to developing self-adaptive software," *J. Inform. Sci. Eng.,* vol. 30, no. 4, pp. 1071-1085, 2014.

[15] T. Vogel and H. Giese, "Model-driven engineering of self-adaptive software with eurema," *ACM TAAS,* vol. 8, no. 4, p. 18, 2014.

[16] Y. E. Gelogo and H. Kim, "LotG: A Design of Adaptive u-learning System," *Asia-pacific Journal of. Multimedia Services Convergent with Art, Humanities, and Sociology,* vol. 5, no. 3, pp. 239-249, 2014.

[17] P. J. Muñoz-Merino, C. D. Kloos, M. Muñoz-Organero, and A. Pardo, "A software engineering model for the development of adaptation rules and its application in a hinting adaptive e-learning system," *Comput. Sci. Inform. Syst.,* vol. 12, no. 1, pp. 203-231, 2015.

[18] P. A. Akiki, A. K. Bandara, and Y. Yu, "Adaptive model-driven user interface development systems," *ACM Comput. Surveys,* vol. 47, no. 1, 2015.

[19] D. L. Moody, "The 'physics' of notations: toward a scientific basis for constructing visual notations in software engineering," *IEEE Trans. Softw. Eng., ,* vol. 35, no. 6, pp. 756-779, 2009.