Evolving an M-learning Software Product Line in an Industry Practitioner Perspective

***Abstract*—The development of software that unifies the expertise from industry with theoretical techniques from academia may providing better and more quality results, with high level of reuse and lower costs. In this perspective, this paper integrates industry practice with academic research to evolve a software product line (SPL) for the development of mobile applications for the teaching of programming. The proposed SPL was evaluated in a previous study and, as a result, we noticed a lack in the integration of practitioners in the performed evaluation. Furthermore, as the set of features identified for the definition of the SPL scope is very broad, there is a need for adopting strategies that can reduce efforts in their development process. In this perspective, experts in development were considered in our research to allow: (i) the evaluation of the SPL conceptual architecture, (ii) the evolution of the SPL architecture, and (iii) the support in modeling of the diagrams that will be considered in implementation of both domain and test realization. The performed analyses provided discussions about: (i) the decisions taken, (ii) the new models of the SPL architecture evolved, (iii) the strategies and techniques adopted to conduct the implementation of each features, and (iv) lessons learned. At the end, considerations about the integration of practical and theoretical perspectives and future work are argued as well.**

I. Introduction

The adoption of information and communication technologies (ICTs) and their learning modalities have providing significant changes on the educational process. Teachers have adopting such technologies as a supporting mechanism in the classroom and to promote the continuity of studies in the learners' houses. Mobile learning (m-learning) is one of these modalities. It enables through mobile devices the learning at anyplace and everywhere. Moreover, such devices are low-cost and are widely present in the routine of young, adult and elder worldwide, creating a propitious scenario for its adoption in the educational process for several domains [**?** **?**].

On the other hand, the development of educational software and applications can hinder the process of adoption of ICTs' based modalities, due to issues involved, i.e., introducing pedagogical and didactic features in the solutions for supporting both teachers and learners; and guarantee the low-cost of the development of software that really met the needs of teachers and educators. An example of educational domain that need to overcome several problems is the teaching of programming [**?**].

The majority of applications in the teaching of programming domain supports informal learning and do not provide a good level of feedback [**?**]. Informal learning is the process of learning which there is no support of a human mediator, such as a teacher or tutor. The application itself is responsible for providing feedback, which difficult its adoption since it do not support widely teachers in classroom. The feedback gave by the applications do not show exactly the students' limitations, consequently teachers still need to monitoring in person their students in the learning process, being the main mediator between content and the learners' difficulties.

Additionally, the reduced mobile devices keyboards and the few use of sensors and touchscreen interactions may demotivated the learners to program in such platform. In this perspective, there is a need of researches that consider the adoption of interactions seen in applications of social network and games for providing better educational applications, besides to identifying the features that really fit as an real improvement of learning for this modality [**?**].

In this perspective, the software product line (SPL) approach was selected for supporting the creation of mobile learning applications for the teaching of programming. The several issues and concerns involved in the educational area and in learning applications can benefit themselves from this methodology, since the teachers have the power to decide which features they expect in the applications for supporting them in classrooms, e.g., through an SPL it is possible to select features of content and learning activities, or only activities, supported by one or more programming languages. Besides, the applications can integrate different means of interaction and formats of learning content and activities [**?** **?**]. Marcolino and Barbosa [**?**] presented an SPL conceptual architecture, conducting the three first sub–processes from the SPLE (software product line engineering) [**?** **?**]: product management, domain requirements engineering and domain design.

The conceptual model [**?**] was evaluated by 31 experts from academic area (software engineers, programming teachers, m-learning experts and mobile developers),

suggesting the architecture design is feasible. To complete such domain design activities, this study aims at: (i) evolving the conceptual architecture in a industry practitioners perspective through a re-analysis of the evaluated architecture, (ii) discussing the proposed improvements based on pedagogical and didactic issues; (iii) representing the improved architecture with UML (Unified Modeling Language) and SMarty (Stereotype-based Management of Variability); and (iv) presenting technique and methodology adopted for the conduction of domain realisation and domain testing SPLE sub-processes.

The research contributes towards: (i) easing SPL adoption for the development of mobile learning applications; (ii) discussing new technologies for supporting a better integration and reuse of pedagogical and didactic features; and (iii) improving the approaches and frameworks adopted considering mainly variables as developers team and time available.

This paper is organized as follows. Section II presents the main concepts about m-learning and the teaching of programming, the proposed m-learning SPL and the SMarty approach. Section III presents the evolution of the SPL conceptual architecture process. Section IV presents and discusses the architecture represented with UML and SMarty. Finally, Section V presents the conclusions and perspectives for future work.

II. Background

A. M-learning and the Teaching of Programming

Mobile devices can be used for fun, entertainment, communication and learning, among many other possibilities. They provide a more personal experience, besides being cheaper than desktops computers [**?**]. Moreover, they can be adopted as a way to carry the classrooms lessons for learners' houses [**?**].

The number of applications for education has growing significantly. However, most of such applications support only informal learning [**?**]. Besides that, there are many issues to overcome and to address for allowing the use of mobile devices' potential in the education in several areas, such as in teaching of programming [**? ?**].

Despite the programming domain has growing and that many countries are changing their primary and secondary curricula for including programming disciplines, such domain still faces several problems [**?**]. Based on the analysis of applications for the teaching of programming, we noticed the need of improvement of the support for teachers in classrooms, the providing of more accurate feedback for the learners and the creation of more engaging environments [**? ? ?**].

Based on these issues and on the need of evolving the learning applications, exploring their features systematically, an SPL to create mobile learning applications for the teaching of programming foundations has been proposed [**?**].

B. The M-learning SPL

The proposed SPL follows the SPLE (Software Product Line Engineering) framework [**?**]. Composed of nine-sub process, the SPLE describes the main activities and artifacts of each subprocess, easing the conduction of the SPL life-cycles, namely: domain engineering and application engineering. Marcolino e Barbosa [**?**] specified each phase in the creation of an M-learning SPL, its conceptual architecture and proposed improvements in the first three SPLE sub-process. Figure 1 presents the SPL conceptual architecture.

In Figure 1, the m-learning applications architecture presents the mobile client application, that provides a macro view of Presentation Layer, Service Layer, Business Layer (Educational and Programming), Data Layer (persistence) and Cross-Cutting/Orthogonal Services Layer and their interactions with external elements, External Infrastructure and External Sources.

The Domain design differs from design of single systems mainly because it incorporates configurations mechanisms into the SPL reference architecture to support the variability of product line, this mechanisms are presented in [**?**] and are considered in the final SPL solution. Furthermore, it allows the development of an adaptable architecture for future inclusion of new features and designates the reusable parts with no loose of rules for the development of specific applications based on the SPL reference architecture.

However, the evaluation conducted with 31 main stakeholders involved in the process did not consider the industry practitioners. Such stakeholders were from academic area, having know–how in software engineering, programming, m-learning and mobile development. Therefore, a re-analysis in practitioners perspectives is needed before the SPL architecture representation in the development and process views, in which both will be supported by the SMarty approach. The aim is evolve the already evaluated architecture to attenuate possible future problems in development phase.

C. The SMarty Approach

The SPLE framework designates the adoption of the orthogonal variability model [**?**]. However, this model adopt a specific graphic notation, requiring the understanding of the notation to be applied. Additionally, the creation of such models needs specific modeling tools for the conception of them. To minimize the inclusion of a new notation for developers and to easy the process of understanding for industry practitioners, the SMarty approach was adopted in the conception of the SPL models [**? ?**].

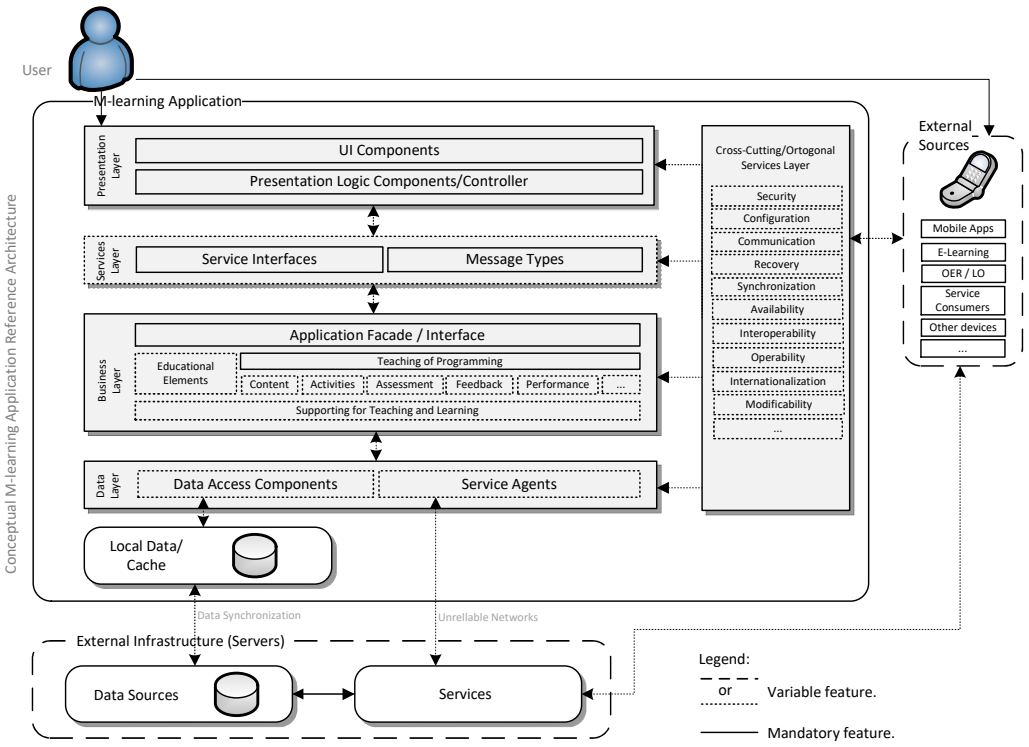

Figure 1. M-learning SPL Conceptual Architecture Excerpt [**?**].

SMarty supports the representation and management of variability in UML models. It supports use case, sequence, class, activity and component UML models through an UML 2 profile, named *SMartyProfile*. UML profile allows an easier integration of their notation in any modeling tool that enables UML profiles import. SMarty also provides the *SMartyProcess*, a set of guidelines that guide users in its application, providing also the support for the detection of (new) features in UML models.

The *SMartyProfile* comprises the following stereotypes: ≪variability≫, ≪mandatory≫, ≪optional≫, ≪alternative_OR≫, ≪alternative_XO≫; ≪mutex≫ and ≪requires≫. It also has a set of attributes for presenting extra and fundamental information such as the rastreability among the variabilities, the maximum and minimum number of variants, all of them included in an UML comment element.

Finally, other reason for the selection of SMarty was the good results of a set of experimental evaluations, where it was compared the process of identification of variability with UML-based Software Engineering Method (PLUS) and Ziadi et al. approach [**? ?**]. Even with a better representation of PLUS method in Class models, mainly by the set of reduced stereotypes, SMarty showed to be the best choice because of its set of guidelines. Moreover, as the approach allows the management of their stereotypes in a profile, the addition of new syntactic elements is facilitated [**?**], enabling the constant evolution of SPL core assets.

III. Evolving the SPL Conceptual Architecture

The SPL conceptual architecture (Figure 1) was evaluated by 31 participants from different areas of expertise, considering the stakeholders' perspective [**?**]. Based on their profiles, we noticed that all of them are from academic area, implying in a possible bias in the evaluation of the architecture in a industry perspective.

Other issue refers to the results related with *conceptual aspects*, *infrastructure aspects* and *SOA aspects*. Comparing the results with the *general aspects* and *educational domain aspects*, the percentage of results that ranked the architecture as *partially complies* suggests possible issues to be reanalyzed.

It was also identified the need of evaluating the architecture and the requirements catalog together, since they had been evaluated separately, what could imply on issues such as imprecise understanding of the software solutions. If the *domain design* and *domain realization* SPLE sub-processes are conducted without proper analysis, the cost and effort to re-factor the project and the sub-processes involved could be higher. Thereby, a reevaluation of the proposed architecture was conducted.

A. Methodology

The methodology applied to support the new analysis of the conceptual architecture follows an industry protocol in which the requirements and needs were deeply analyzed for proposing the architecture.

Such industry protocol involves the process of rigorous analysis of (i) each requirement (functional and non - functional) independently, for the identification of it possible impact and development technique to be adopted; (i) the analysis of set of requirements and possible expect and unexpected behaviors, when combined with other sets of requirements; (ii) architectural designing decisions for each set of requirements and after, for the entire solution; (iv) technical debt issues and; (v) human resources available in the development team and time.

Then the following steps were conducted:

- the previous architecture model (Figure 1) and its publication were analyzed;
- the requirements catalog was discussed considering the ideas of the first architecture model and the learning and didactic objectives to minimize problems [**?**] in programming domain considering mobile learning applications; and
- based on the previous analysis, each package/layer in the conceptual architecture was studied and restructured when necessary.

The analysis occurred during one month, where two practitioners from the industry supported the evolution of the model. Both have an average of eight years of expertise in mobile development in industry. One of them have also expertise in software product line, three years, and the other one in development of hybrid applications, five years.

It is highlighted that the number of practitioners is small, but considering the availability and the reality of Brazilian academy agreements with software industries, this interaction, even in with a small participants, allows the identification of possible lack of know-how in academic perspective, where the theoretical knowledge is bigger than the practice. Finally, it is also emphasized that the participants also collaborate with the development of the UML models with SMarty.

B. Discussions and Evolved Conceptual Architecture

After the execution of each step previously discussed, the conceptual model was updated as shows Figure 2.

Domain Engineering, *Application Mechanism* and *Application Engineering* do not changed (Figure 1). All of them continue with the same roles and sub–processes, with the exception from the way in which the applications are generated. This change implies in most of the changes in the m–learning application architecture.

For the user, the *application mechanism*[**?**] will work in the same way: the user answers some questions or selects the desired features and the support mechanism generates the application for download. However, with the updates, the choices made by the users will be stored in a database (or metadata files) and will be considered for delivering an application instantiated with only the selected features. So, if the institution privilege is considered, only a set of features allowed for it will be available for the users' institution. This software architecture strategy is called multi–tenancy.

The mult–tenancy architecture refers to an architecture in which a single software solution is executing on a server, serving multiple groups of users who share common access in such instance of software. Each of these groups is called tenant. Looking at cloud computing models, the multi–tenancy approach is perfectly adherent to Software as a Service (SaaS) [**?**].

The change in the application mechanism implies in a delivery of a complete application instance (presentation and business layers), no matter the selected features. However, at the end, the application will solve its variabilities in execution time based on the set of selected features, providing only features and services related to the current user, the owner of the configuration defined in advance.

In the multi–tenancy proposal, the code and database packages are installed on a single server, simplifying the release and scalability processes [**?**].

At this point, many concerns arise:

Q1 – Why was this decision made?

Imagine the hypothetical scenario where a teacher generated an application for two classes to be used in one semester. After finished the period, he receives in the next semester four more classes, but now he wants to use the application with a new feature, not selected in the previously semester. Until now there is no problem. He can generate another application and use it with his new classes but, and the data from the other application? Additionally, if the teacher decides to change his choices for the application in the middle of semester, he will need to create a new application and re–install each student's application. Furthermore, if the new configuration is not concise with the previous features – in a software product line constraint perspective – teachers and their students will not be able to correctly use the application with the new changes.

A final issue is the number of applications to be controlled based on several teachers and classes in the same institution. Data for the educational administrator will be always broken in parts, making the process to manage and to improve the learning and teaching processes difficult, mainly by each specificity of the applications. It is highlighted that this requirement was not change, just new considerations were made, allowing a deeper understanding of the features, besides, this is only one

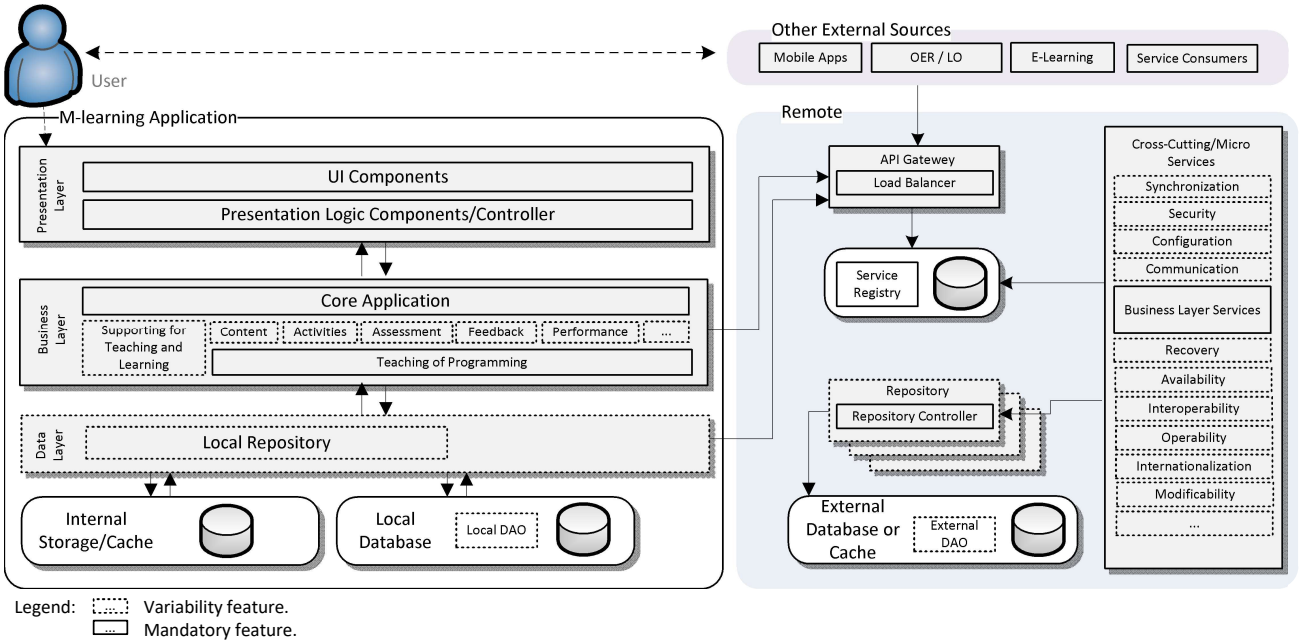

Figure 2. Updated M-learning SPL Conceptual Architecture.

of several implications that were identified, leading to the proposed changes.

Q2 – Do these changes not imply in significantly changes on the functionality of the applications?

Yes, mainly about the connection-dependency. Adopting multi–tenancy implies in a connection, since this connection will be required for the presentation layer shows only the features selected for the current user. However, cache and local database can be used to allow the use of the application when a connection is not available, mainly when a mobile device is the platform in use. Such usage is the reason for a new update in the architecture: the adoption of repository pattern, later discussed in this section.

Q3 – Which trade-off may these types of application have?

- **Positive factors:** one instance in a server for multiple users, centralizing scalability issues and simplifying the release management.
- **Negative factors:** Larger development effort, mainly to maintain and to use per–tenant data allowing the customization of the applications. Furthermore, multi–tenancy increases the risks and impacts inherent in applying a new release version. As there is a single software instance serving multiple tenants, an update on this instance may cause downtime for all tenants even if the update is requested and useful for only one tenant [**?**].

Based on the adoption of multi–tenancy, a new challenge appears: guarantee the correct execution of applications in mobile devices in areas where network connection is not available. For this, as mentioned in **Q2**, the repository pattern was adopted as a solution in the SPL architecture proposed. Although, before the understanding of this pattern, it is important to make an observation related to the pedagogical and didactic concern: the constantly feedback in applications.

According to Marcolino and Barbosa [**?**], there is a problem in the given feedback in mobile applications for the teaching of programming. As such applications generally support informal than formal learning, feedback features are given only for one of the categories of the users, teachers or students. Thereby, the need of a more concise feedback to support students to learn by themselves and to notify teachers about the students' performance is a mandatory pedagogical and didactic feature to support formal learning. The feedback allows the improvement of the curricula, changes on didactic strategies adopted in classroom, changing on teaching topics and other improvements [**?**]. Thus, the adoption of a mechanism to store and retrieve the data of the learning process is fundamental, meeting the need for the adoption of mechanisms to enable it, as the repository pattern does.

The repository pattern avoid the directly access of data from the business logic layer, services or other entities, reducing the duplicate code, weak types of business data, the difficulty in centralizing data from cache and several data bases and, the inability to easily test the business logic [**?**]. The pattern manages

several data repositories, common in a microservices architecture and it also enables caching and storing data in mobile applications. Based on these benefits, the repository pattern was selected to deal with the multi–tenancy concerns, integrating a DAO (data access object) strategy when convenient, and easing the manage of data from the microservices – other update in the proposed SPL architecture.

The repository is shown in Figure 2. It is available in two perspectives, directly in the client application or in the server side. The *local repository* can realize the persistence through internal storage, i.e., private data in the device memory; or in local database, i.e., structured data in private database and; with network connection, i.e., calling a microservice. On the other hand, the repository on the server side is used by the microservices only, and realize its persistence through a external storage, i.e., a database or cache [**?**].

Among other benefits encompassed with the adoption of the repository pattern, we highlight that: (i) they can adopted different persistence strategies for different contexts, such as relational databases, non-relational databases, cache and services; and (ii) applications can adopt managers for easing the integration and encapsulation of the dependencies strategies.

Finally, the last update made was the adoption of microservices concept, which integrates the SPL proposal for the development of applications as a SaaS [**?**].

The adoption of mobile learning applications has several benefits (see Section II). However, the management of data, the availability of new content, learning activities and other elements presented in a learning environment though the reduced mobile screens and its native keyboard may not be a good way for conducting such tasks. In this perspective, providing different paths of access to those functionalities in different platforms can improve the user experience and, microservices support this strategy.

Microservices allow the use of different platforms and technologies as consumers of them. The responsibility is uncoupled from the business layer, which has only the function of accessing the microservices available on one or more servers and providing for the presentation layer such functionality. Looking at the SPL concept, the microservices meet and facilitates the process of management and definition of variabilities. Microservices provide a set of fully uncoupled services, but highly collaborative. Each service implements a set of narrowly, related functions. For example, an application might consist of services such as the order management service, the customer management service, etc.

Services communicate using either synchronous protocols, such as HTTP/REST (Representational state transfer), SOAP (Simple Object Access Protocol), or asynchronous protocols, such as AMQP (Advanced Message Queuing Protocol) and MQTT (Message Queue Telemetry Transport). Also, they can be developed and deployed independently of one another [**?**]. As each service has its own database, the data consistency is improved. Add a data management standard, such as repository, and you'll have an even more robust solution. And, considering the multi-tenancy architecture, just the features selected in advance will be available and visible for each tenant/user.

Also in Figure ref fig: schema2, other difference from the first model is clear: the relation between *other external sources* with the application. This goes to meet what was discussed about the multi–tenancy and microservices. These external sources will benefit from these adoptions, since its relation with the *API Gateway*, its *load balancer* and the *service library* will allow the consumption of the microservices by other platforms or even by other applications, improving the reuse of features not dependent from the domain of the line.

The extra *service layer* was removed and the *core application* component replaced the *application facade/interface*. Now, the *business layer* has a relation with the *API Gateway*, which balances the requests among the different tenants and applications to the microservices which are, in their turn, registered in a data base managed by the *service registry*. Every microservice that reflects a feature in the SPL and is available to be selected by users, has a register entry that is visible by the *service registry* and available by it for the *API Gateway*, responsible to redirect each requisition received from the *business layer* or *external sources* for its respective microservice.

Among other benefits encompassed with the adoption of microservices, we highlight that: (i) they are independent; if one fails or goes to maintenance, the others will continue available; (ii) they can be implemented adopting different technologies; (iii) they can have one or more repositories; and (iv) they can be encapsulated and works only with the complexity from the business domain, easing the development, since the complexity is distributed among such domains.

Some concerns also emerged with its adoption: (i) it is hard to separate business domains, but for our SPL this difficulty does not occur, since the mentioned m-learning requirements catalog solve this concern; and (ii) the deploy of several microservices and the hardware maintenance need more efforts than the development of monolithic systems.

We also highlight some API Gateway benefits: (i) it is adopted as a facade layer with the objective to facilitate the clients' accesses to the microservices; and (ii) it is an "intelligent" layer that can balance the load of number of requisitions and the caching information.

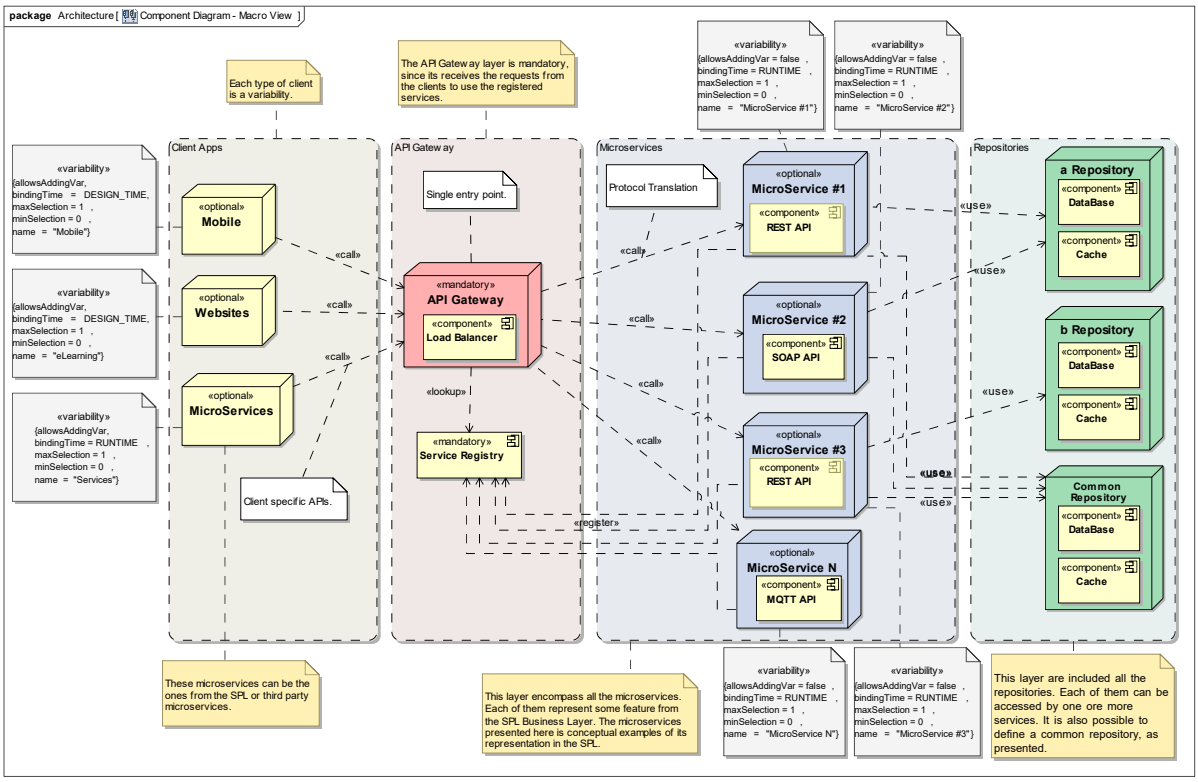

Figure 3. M-learning SPL Component Model.

On the other hand, one concern is the disadvantage in the creation of a single point of failure.

Therefore, with the macro view of the conceptual SPL architecture updated, the *domain design* sub-process is retaken Its artifacts models are presented and discussed next.

IV. SPL Architecture with SMarty

The SPL methodology differs from the development of singular software by the postponement of project decisions. The features are included in the software by convenience and the need of stakeholders. In the proposed SPL, the variabilities are concentrate in the *business layer* features [**?**].

The integration of the teaching of programming domain, mobile platform and teaching and learning processes presents several features. The most of them need to be combined to provide a richer learning environment, allowing the achievement of the educational objective. The several features considered in the proposed SPL come from the analysis of 81 applications and software systems in the programming domain [1]. Consequently, the catalog was represented in a feature model, depicting the relation among features, variabilities, variation points, variants and their constraints. As previously mentioned, SMarty was selected to represent this in UML diagrams.

Among the *domain design* sub-processes, the *logical view* is represented through the feature model [**?**]. The *development view* can adopt the package diagram, component and class diagram. Figure 3 shows the component model of the SPL.

A. Component Architecture

The component diagram depicts how components are wired together to form larger components or software systems. They are used to illustrate the structure of arbitrarily complex systems[**?**]. The SPL representation has four layers: *client apps*, *API gateway*, *microservices* and *repositories*.

Client Apps layer: this layer shows the apps, platforms, websites, services or any potential consumer of the microservices. The main clients considered in the research are *mobile apps*, followed by *websites*. These clients have just the view and controller of the application. All the business logic is inserted in the microservices. However, they are the main way to wire together all the desired features by the tenant/user, previously selected in the *application mechanism*. Additionally, all the clients are considered optional variabilities, since they may or not be part of the SPL.

[1] goo.gl/EvzUpB

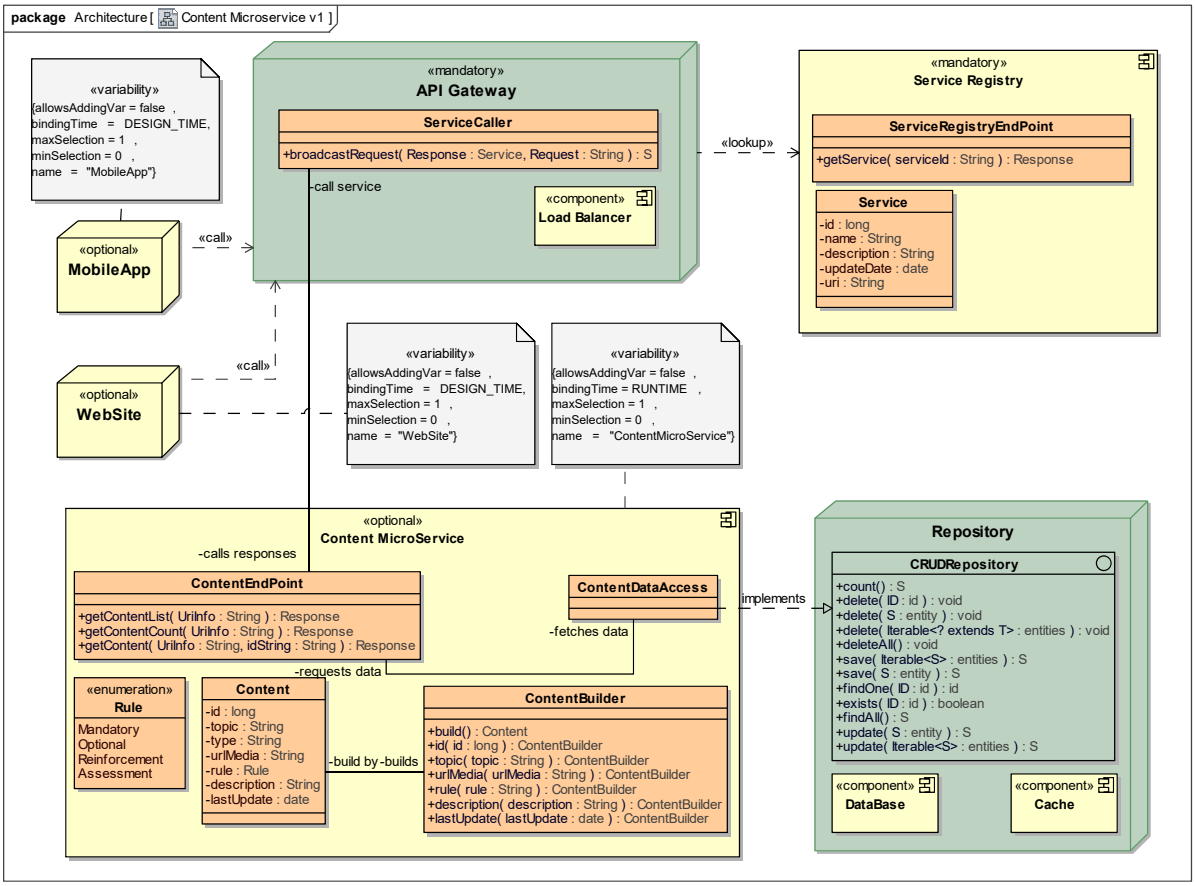

Figure 4. M-learning SPL Content Microservice Class Diagram.

In the representation, the clients are tagged as ≪optional≫ and each one has a UML comment which depicts additional information about them. Those variabilities are solved in design time. Each instance of client calls the *API Gateway* node from the next layer.

API Gateway: this layer identifies the available microservices registered in a data base through the mandatory *service registry* component. The *API Gateway* node, also mandatory, looks up for the microservice requested by the client in the *service layer*. If the microservice is found, the requisition is broadcast for the respectively microservice in the next layer. Additionally, the *load balancer* observes the behavior and quantity of requests from the clients, balancing the load of data traffic and ensuring that all the requests will be answered, controlling them in a row. This component is also responsible to increase the server resources; in other words, it provides the elasticity of the cloud resources [**? ?**].

Microservices layer: this layer encompass all the microservices, although each one can stay hosted in different servers. The microservices has specific functionality, from non-functional to functional ones and each represents the features depicts in the feature model. They are instances of each selected feature in the *application mechanism*. The model presents four samples of services. Everyone has a component which express the type of supported protocol and, based on this protocol, each one receives requests directed by the *API Gateway*. Still in this layer, each microservice is tagged as optional variability, but following the feature model, some of them will be represented as exclusive (≪alternative_XO≫) or be mutually exclusive among the other microservices selected. For instance, educational activities can be available following a fixed order or be performed by convenience, therefore, these two features of the availability of educational activities follow the SPL mutually exclusive constraint. On the other hand, there are features that require others, such as the learning monitoring and performance, that needs feedback features to support both teachers and students.

Other behavior on the microservice layer is the possibility for the microservices to do calls among them, when it depends on the process of another microservices, removing the needs of processing from the client. These

calls among the microservices is also performed by the API gateway.

Repositories Layer: in this layer all repository structure is presented. They can be in different servers and, in some cases, they are not mandatory for some services. The repository is responsible for the management of the persistence process, providing interfaces in DAO model, if convenient, for enabling the access of the data in their registered data base. They also coordinate the cache usage.

B. Architecture Class Diagram

The next architecture artifact from the SPLE framework is the class diagram (Figure 4). For this representation, as each microservice represents a feature, or a small set of them, the class diagram was divided in one representation for each existing artifact. This decision was made based on some issues, particularly: (i) the reduced number of developers, (ii) the reduced time for the development of each feature before its development, (iii) the facility to manage each diagram of the architecture as an artifact from the SPL core asset; and (iv) the the extreme programming (XP) agile process [**?**] with the test driven development (TDD) [**?**] adopted for the conduction of domain realization and domain testing SPLE sub-processes (an investigation about this adoption will be conducted in order to provide evidences of the possible benefits in SPL context) .

Figure 4 shows the content feature microservice represented in a class diagram. The diagram considers two instances of clients: an *MobileApp* and a *WebSite*. Both calls the content *API Gateway* which look up in the *Service Registry* to identify if the requisition for such microservice can be answered. For this, the respective called microservice need to be register on the queried data base and be identified with the support of a *repository*. With the identified service, the *API Gateway* can localize the service through its *uri*, executing the request that comes from the customer and returning to it the retrieved content data being presented in their app interface through the controller of content and its communication with the presentation logic layer.

Three variabilities are represented in the class diagram. The two clients, where both are solved in design time; and the content microservice, that is solved in runtime. The content microservice is considered a variability because some learning applications can be developed only to provide activities for the students, not being mandatory for every application. The difference among the binding time of the variabilities is related with the type of variability. The clients need to have its presentation and business logic layers developed before, since they will consume the available microservices accordingly to the permission of features selected for users on the *application generation mechanism*. The availability of the microservices, in turn, will occur in runtime. All microservices will be available for the client applications, although only the ones that the user logged in the application have permission or were selected to their users will be visible and can be used.

C. Lessons Learned

Based on the tasks and decisions made on this research, the two main lessons learned were:

- Evaluate the requirements catalog and after, the first conceptual model architecture without a integration of both could be resulted in future problems in development phase. This was made based on the order of research phases and this threat was mitigate based on the practitioners analysis.
- Microservices improve the reuse and, in some cases, can be adopted in other software programs independently of the domain. Additionally, the reduced size of functionalities expressed on them enables the creation of a greater number of configurations of application. In this perspective, other microservices can be easier integrated to our solution and even, our microservices can be reused in other contexts. However, the evolution of the line will need more attention, since the communication among these microservices requires more efforts and tests to guarantee that they will work properly.

V. Conclusions and Future Work

In this paper we discussed the evolution of a conceptual SPL architecture for the development of mobile learning applications for the teaching of programming based on the analysis of practitioners from the industry. The first model considered only theoretical and academic views for its evaluation. To provide a more real industrial perspective, two practitioners were integrated to our research, giving contributions to the improvement of the project.

Based on the evolved model, the component architecture diagram was presented. The model encompasses the multi–tenancy concept, microservices and repository pattern. Furthermore, the content feature was represented with UML diagram and the SMarty approach.

All the decisions and models proposed lead to a possible solution for some problems faced on the domain of teaching and programming through mobile learning applications. The discussions presented also allow the adoption of these decisions in other projects which need to deal with several educational features and short releases, besides to promote the reuse of these features.

Our next steps are the finishing of the analysis performed with teachers of programming to define our first set of features to be developed and integrate the core

asset of the line. Additionally, the proposed integration of XP and TDD needs to be carefully registered to allow the identification of its positive and negative impacts on the development process.

Acknowledgment

The authors acknowledge the Brazilian funding agencies (CAPES (DS-8907173DT), FAPESP and CNPq) for the financial support provided to this research.

References

[1] Viviane Cristina Oliveira Aureliano and Patrícia Cabral de Azevedo Restelli Tedesco. Ensino e Aprendizagem de Programação para Iniciantes: uma Revisão Sistemática da Literatura focada no SBIE e WIE. *Simpósio Brasileiro de Informática na Educação*, 2012.

[2] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2Nd Edition)*. Addison-Wesley Professional, 2004.

[3] Caitlin Duncan, Tim Bell, and Steve Tanimoto. Should your 8-year-old learn coding? In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, WiPSCE '14, pages 60–69, New York, NY, USA, 2014. ACM.

[4] V. Falvo Junior, Nemésio F. Duarte Filho, Edson OliveiraJr, and Ellen Francine Barbosa. Towards the Establishment of a Software Product Line for Mobile Learning Applications. *Int. Conf. on Software Engineering and Knowledge Engineering (SEKE)*, 1:678–683, 2014.

[5] Steven Fraser, Kent Beck, Bil Caputo, Tim Mackinnon, James Newkirk, and Charlie Poole. Test driven development (tdd). In *Proceedings of the 4th International Conference on Extreme Programming and Agile Processes in Software Engineering*, pages 459–462, Berlin, Heidelberg, 2003. Springer-Verlag.

[6] Rouven Krebs, Christof Momm, and Samuel Kounev. Architectural concerns in multi-tenant saas applications. *Closer*, 12:426–431, 2012.

[7] Frank J. Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer, Berlin, 2007.

[8] A. Marcolino, Edson OliveiraJr, I. M. S. Gimenes, and J. C. Maldonado. Towards the Effectiveness of a Variability Management Approach at Use Case Level. *Int. Conf. on Software Engineering & Knowledge Engineering*, v. 1.:214–219, 2013.

[9] Anderson Marcolino, Edson OliveiraJr, and Itana Maria de Souza Gimenes. Variability Identification and Representation in Software Product Line UML Sequence Diagrams: Proposal and Empirical Study. In *2014 Brazilian Symposium on Software Engineering, Maceió, Brazil, Set. 28 - October 3, 2014*, pages 141–150, 2014.

[10] Anderson S. Marcolino and Ellen Francine Barbosa. Linhas de Produto de Software no Domínio Educacional: Um Mapeamento Sistemático. *Simpósio Brasileiro de Informática na Educação*, 1:239–249, 2015.

[11] Anderson S. Marcolino and Ellen Francine Barbosa. Softwares Educacionais para o Ensino de Programação: Um Mapeamento Sistemático. *Simpósio Brasileiro de Informática na Educação*, 1:190–199, 2015.

[12] Anderson S. Marcolino and Ellen Francine Barbosa. Towards an m-learning requirements catalog for the development of educational applications for the teaching of programming. *In: 2016 IEEE Frontiers in Education Conference (FIE), 2016, Erie (PA).*, pages 1–5, 2016.

[13] Anderson S. Marcolino and Ellen Francine Barbosa. Towards a software product line architecture to build m-learning applications for the teaching of programming. *In: 50th Hawaii International Conference on System Sciences (HICSS-52), 2017, Kauai (HI) - EUA.*, pages 6264–6273, 2017.

[14] Microsoft. The repository pattern, 2017.

[15] Sam Newman. *Building Microservices*. O'Reilly Media, Inc., 1st edition, 2015.

[16] Edson Alves Oliveira, Junior, Itana Maria Souza Gimenes, and Jose Carlos Maldonado. Systematic Evaluation of Software Product Line Architectures. *Journal of Universal Computer Science*, 19(1):25–52, jan 2013.

[17] K. Pohl, G. Bockle, and F. Linden. *Software Product Line Engineering Foundations, Principle, and Techniques*. Secaucus, NJ, USA: Springer-Verlag, 2005.

[18] Roger Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., New York, NY, USA, 8 edition, 2015.

[19] P. Sanchez Barreiro, D. Garcia-Saiz, and M.E. Zorrilla Pantaleon. Building Families of Software Products for e-Learning Platforms: A Case Study. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, 9(2):64–71, Maio 2014.

[20] D. M. Souza, M. H. S. Batista, and E. F. Barbosa. Problemas e Dificuldades no Ensino de Programação: Um Mapeamento Sistemático. In *Revista Brasileira de Informática na Educação.*, 2015.

[21] Nikolai Tillmann, Michal Moskal, Jonathan De Halleux, Manuel Fahndrich, Judith Bishop, Arjmand Samuel, and Tao Xie. The Future of Teaching Programming is on Mobile Devices. pages 156 – 161, Haifa, Israel, 2012.