An ontology-driven software product line architecture for developing gamified intelligent tutoring systems

Diego Dermeval*
Penedo, Campus Arapiraca,
Federal University of Alagoas (UFAL),
Penedo, AL, 57200-000, Brazil
Email: diego.matos@penedo.ufal.br
*Corresponding author

Glauber Leite, João Almeida,
Josmário Albuquerque and Ig I. Bittencourt
Computing Institute,
Federal University of Alagoas (UFAL),
Maceió, AL, 57072-900, Brazil
Email: grl@ic.ufal.br
Email: jvas@ic.ufal.br
Email: jas@ic.ufal.br
Email: ig.ibert@ic.ufal.br

Sean W.M. Siqueira
Department of Applied Informatics,
Federal University of the State of Rio de Janeiro (UNIRIO),
Rio de Janeiro, RJ, 22290-240, Brazil
Email: sean@uniriotec.br

Seiji Isotani
Institute of Mathematics and Computer Science,
University of São Paulo (USP),
São Carlos, SP, 13566-590, Brazil
Email: sisotani@icmc.usp.br

Alan Pedro da Silva
Computing Institute,
Federal University of Alagoas (UFAL),
Maceió, AL, 57072-900, Brazil
Email: alanpedro@ic.ufal.br
Abstract: Intelligent tutoring systems (ITSs) are effective to provide instruction for students in several situations. Many works have been using gamification by adding game elements to learning contexts aiming to engage students and to drive desired learning behaviours. However, the design of gamified ITS should deal with a huge variability. Software product lines (SPLs) promise to offer rapid product development and more affordable development costs to build software from the same family. A key factor to successfully implement a product-line approach is to structure commonalities and variabilities into a product line architecture (PLA). In this paper, we propose a PLA for developing gamified ITSs that uses an ontology-driven feature modelling strategy. We illustrate how our architecture could be applied to instantiate a product on the basic math domain. We also discuss a set of implications of using it as well as how it could support the evolution/changing of gamified ITSs.

Keywords: ITSs; intelligent tutoring systems; gamification; gamified intelligent tutoring systems; SPL; software product line; ontologies.


Biographical notes: Diego Dermeval is a Professor at the Federal University of Alagoas (Penedo Educational Unit) and co-leader of the Center of Excellence for Social Technologies (NEES). He received his PhD from the Federal University of Campina Grande (UFCG), Brazil, in 2017. His research interests include computers and education, ontologies and software engineering.

Glauber Leite is an undergraduate student in Computer Engineering at the Federal University of Alagoas. He is a member of the Center for Excellence in Social Technologies (NEES) research group. His research interests include computers and education and software engineering.

João Almeida is an undergraduate student in Computer Science at the Federal University of Alagoas. He is a Member of the Center for Excellence in Social Technologies (NEES) research group. His main research interests include computers and education and software engineering.

Josmárió Albuquerque is an MSc student at the Federal University of Alagoas. He is a member of the Center for Excellence in Social Technologies (NEES) research group. His main research interests include computers and education and software engineering.

Ig I. Bittencourt is an Associate Professor at the Federal University of Alagoas (Brazil), the President of the Special Committee on Computers and Education from Brazilian Computer Society (leading around 2500 researchers) and the Editor of IEEE Transactions on Learning Technologies. He co-founded an awarded company called MeuTutor (more than 50,000 students already used) and he stands out from his peers by creating one of the most innovative companies in the field of educational technology in Brazil. He believes in innovative social entrepreneurship as a model for promoting a sustainable economic and social development to humankind.
An ontology-driven software product line architecture

Sean W.M. Siqueira is a Professor at the Federal University of the State of Rio de Janeiro. He received his PhD from the Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) in 2005 and his Post-doctoral Degree from the Federal University of Alagoas in 2016. His main research interests include research, development and innovation in web science, information systems and computers and education.

Seiji Isotani is a Professor at the Institute of Mathematics and Computational Sciences of the University of São Paulo. He received his PhD from the Osaka University (Japan) in 2009 and his Post-doctoral Degree from the Carnegie Mellon University (USA) in 2011. His main research interests include research, development and innovation in intelligent tutoring systems, CSCL, ontologies, open linked data, semantic web and educational data visualisation/mining.

Alan Pedro da Silva is a Professor at the Federal University of Alagoas (Brazil). He received his PhD from the Federal University of Campina Grande, Brazil, in 2011. His main research interests include research, development and innovation in software engineering, semantic web and artificial intelligence in education.

1 Introduction

Evidence from several empirical studies suggests that intelligent tutoring systems (ITSs) (Sleeman and Brown, 1982) can successfully complement and substitute other instructional models in many situations (Boulay, 2016). These situations exist at all educational levels and in many common academic subjects (Ma et al., 2014).

However, the traditional development of ITSs does not take into account or does not make efforts to engage and motivate students. Meanwhile, VanLehn (2011) argues that motivated, challenged and intrigued students tend to have better learning results. In this way, relying on several theories and models of motivation and human behaviour, for instance, Fogg’s behaviour model, need theories and Skinner’s reinforcement theory, many works have been using persuasive technologies (e.g., gamification) in connection with education (Hamari et al., 2014; Borges et al., 2014; Seaborn and Fels, 2015).

Gamification can be defined as “the use of game elements and game design techniques in non-game contexts” (Werbach and Hunter, 2012). In particular, many works have been using gamification in the context of online courses. They add game elements (e.g., levels, points, badges, progress, avatars and so on) to learning contexts aiming to engage students and to drive desired learning behaviours (Kapp, 2012). This way, in addition to ITSs’ technological and pedagogical perspectives, we need to consider learner’s motivational perspective in ITS design. This new perspective increases even more the complexity for developing ITSs with explicit learner’s motivation focus.

However, the design of gamified ITSs should deal with a huge variability of features. Besides considering the variable software requirements (technological perspective) and different educational strategies (pedagogical perspective) of systems, several gamification elements could be combined with aspects of each one of the other perspectives.

To illustrate the high variability presented in gamified ITS, we mention an example of a system named Meu Tutor (in English, My Tutor). It is a gamified ITS that aims to help
high-school Brazilian students to be prepared to take the high-school national exam (called ENEM\(^3\)). Regarding the technological perspective, there are more than 50 features (e.g., login, register, social integration, evaluation, reports and so on) provided by the system, where at least 15 of them can be optionally included in a particular configuration of the system. Considering the pedagogical perspective, it uses a problem-based learning (PBL) strategy and takes into account 10 courses (related to the high-school exam domain, for instance, math, physics, biology and so forth), 20 subjects per course and a minimum of 30 educational resources (e.g., content, problem and so on) for each subject. Moreover, in the current configuration of Meu Tutor, there are six gamification elements (e.g., points, levels, badges, mission, leaderboard and progress bar) that could be combined with each other resulting in a total of 63 possibilities (using the equation, where \(n\) is the number of gamification elements) for using such elements in a combined way. Hence, multiplying the number of optional features, disciplines, subjects, resources and number of gamification elements combinations leads to a total of 5,670,000 possible combinations of configurations that a system like Meu Tutor could have to manage – this is the maximum number of combinations.

Otherwise, the concept of SPL (Clements and Northrop, 2001; Pohl et al., 2005), from software engineering research, promotes to offer characteristics such as rapid product development, reduced time-to-market, quality improvement and more affordable development costs. An SPL is a set of software systems that have a particular set of common features and that satisfy the needs of a particular market segment or mission (Clements and Northrop, 2001). SPL uses in a more efficient way a set of software reuse techniques (e.g., frameworks, services and components) since it provides a systematic methodology for planning reuse of component or service-based systems and there is a way to customise the production of software from a same family. In this context, considering the huge variability presented in gamified ITS, the use of SPL appears to be appropriate and promising to aid the development of gamified ITSs.

In this context, feature modelling (Kang et al., 1990) is one of the key activities involved in the design of SPLs. In general, a feature model is produced in this activity to represent the commonalities and variabilities of SPLs. To manage the variability of gamified ITSs, such activity should be performed. However, a gamified ITS in a specific domain may require different requirements, pedagogical strategies and gamification elements. Thus, allowing a particular gamified ITS to be reconfigured at runtime to change, for instance, a gamification element, can improve the flexibility of a system to be adapted to fluctuations in user’s (e.g., learners) and author’s (e.g., teachers) needs. In this way, enabling the automatic analysis of feature models and hence providing the reconfiguration of an ITS that takes into account motivational aspects of learners is required. Achieving these characteristics could allow, for example, to monitor learner’s motivational levels and to reconfigure the system with a different combination of gamification elements that could improve the engagement of students. Thus, in comparison with other mechanisms for automatic analysis of features models (Benavides et al., 2013), description logic (DL)-based methods promise to provide improved automated inconsistency detection, reasoning efficiency, scalability and expressivity (Wang et al., 2007; Benavides et al., 2013).

Ontology (Gruber, 1993) is the most common way to represent feature models knowledge based on DL reasoners. Ontology languages are used for domain formalisation by defining classes and properties for these classes, individuals (that instantiate the classes), properties of individuals and statements on these individuals.
An ontology-driven software product line architecture

They are generally represented using the web ontology language (OWL). Previous works (e.g., Wang et al., 2007; Tenório et al., 2014) use OWL to formally represent feature diagrams, aiming to provide means for automatic reasoning. Empirical results (Dermeval et al., 2015) indicate that using an ontology-based feature modelling style based on OWL individuals (i.e., the work of Tenório et al. (2014)) is more flexible and demands less time for changing than the one based on OWL classes and properties (e.g., the work of Wang et al. (2007)).

In the context of SPLs, PLAs should incorporate a variety of similar products in terms of their architectural elements. The more variable these architectural elements are, the more different products can be generated from them. Software variability (van Gurp et al., 2001) can be reached by delaying architectural design decisions, and it can be described through variation points and variants. Therefore, a key factor to successfully implement an architectural product-line approach is to structure commonalities and variabilities into a PLA in terms of variable architectural components and connectors, and variable interfaces associated to variants of variation points.

In this paper, we propose a gamified intelligent tutoring systems product line (GITS-PL) architecture that relies on the use of ontologies for reasoning on feature models. We apply Pohl’s SPL engineering process (Pohl et al., 2005) to define our architecture. First, we conduct the requirements engineering (Kotonya and Sommerville, 1998) of GITS-PL, which results in two artefacts: use case models and a feature model. Moreover, we use the ontology proposed by Tenório et al. (2014) to represent the feature model of GITS-PL. Then, we specify the modules and component and connectors views (Garlan et al., 2010) of our SPL architecture taking into account the variability represented in GITS-PL feature model (i.e., which is represented in an ontology). Afterwards, we illustrate how we could apply GITS-PL architecture to instantiate a product on the math domain. Then, we also discuss the implications – considering development costs, quality and time-to-market variables – of using the GITS-PL architecture to derive new products as well as how our architecture could support the variability evolution of gamified ITSs.

2 Gamified intelligent tutoring systems product line architecture (GITS-PL)

This section explains how GITS-PLA has been developed. We have followed the high-level process presented by Pohl et al. (2005) with the aim of constructing a platform that takes into account the commonalities and variabilities of ITSs that use game elements. Pohl et al. (2005) define a process that specifies a set of activities of software development that support the systematic creation of software artefacts aiming to manage the commonalities and variabilities of an SPL. There are two specific sub-processes to each one of the essential aspects of an SPL: Domain Engineering and Application Engineering. In this work, we are relying on the high-level steps defined by Pohl to develop the GITS-PL architecture. As such, among the four steps (i.e., domain requirements engineering, domain design, domain realisation and domain testing) defined by Pohl’s Domain Engineering, we consider only the first two: requirements engineering and design. Note that, as further explained, we are using specific methods and techniques, for instance, Gomaa for use case modelling (Gomaa, 2004), Feature-Oriented Domain Analysis (Kang et al., 1990) for feature modelling and Junior’s approach (Junior et al.,
2010) for aiding architectural design, rather than Pohl’s methods such as the orthogonal variability modelling (Pohl et al., 2005).

In the following sections, we present how the Requirements Engineering and the Architectural Design sub-processes were performed. The last two activities are beyond the scope of this paper, since we have followed the process until the Architectural Design sub-process.

2.1 Requirements engineering

The classic requirements engineering process is composed by several activities: Requirements Elicitation, Requirements Analysis and Negotiation, Requirements Documentation and Requirements Validation (Kotonya and Sommerville, 1998).

To conduct the requirements engineering of our SPL for gamified ITSs, we have followed those activities. We have elicited the requirements of GITS-PLA by:

- analysing the literature about ITS (e.g., Woolf, 2010) and gamification applied to education (Kapp, 2012; Werbach and Hunter, 2012; Hamari et al., 2014; Borges et al., 2014; Seaborn and Fels, 2015)
- identifying the functionalities presented of real ITS that uses gamification in industry (i.e., the Meu Tutor ITS)
- interviewing the Meu Tutor requirements engineering team to gather information about new requirements that would be considered in future versions of Meu Tutor systems.

In the Requirements Analysis and Negotiation phase, we have analysed the elicited requirements and identified common and variable requirements for GITS-PL, and then we have negotiated which of them were actually common or variable in a meeting with the Meu Tutor company requirements engineering team. In the Requirements Documentation phase of GITS-PL, we have produced two main requirements artefacts that encompass the commonalities and variabilities of our SPL: use cases model and feature model. Use cases represent in a finer granularity the functionalities of GITS-PLA, whereas features represent in a coarse grain the SPL functionalities, i.e., every feature incorporates one or more use cases within its representation. In this paper, we rely on Gomaa’s use case extended notation (Gomaa, 2004) to represent the variable functional requirements of GITS-PL. However, for the sake of space, we do not present the diagram.

After producing the GITS-PLA requirements artefacts, we have validated the use cases models and the feature model with Meu Tutor requirements engineering team. The validation was conducted by two rounds of discussions, feedbacks and modifications in these artefacts aiming to result in a final list of features to be included in GITS-PL. In the following, we describe how we have conducted the feature modelling of our SPL.

2.2 Feature modelling and ontology-based feature modelling for GITS-PL

As previously explained, the variability of SPLs is commonly expressed through features represented in feature models. In this paper, we use the well-known and broadly adopted feature-oriented domain analysis (FODA) technique (Kang et al., 1990) for modelling GITS-PL features.
The diagram presented in Figure 1 depicts the features that could be incorporated in products derived from GITS-PL. For the moment, ignore that some features are green as it will be further explained. As seen in Figure 1, the Register (along with a Student Model feature), Login, Strategy, Evaluation, Gamification and Domain Model are mandatory features, i.e., they must be included in all gamified ITSs that are derived from GITS-PL. These features are mandatory since they are supported by a great part of ITSs presented in the literature (Woolf, 2010). Additionally, the Course Management, Social and Report features are optional, i.e., they can be included or not in an ITS product.

Figure 1  Feature model of GITS-PL (see online version for colours)
or her performance with other users. The Progress Bar feature may present to the users/learners their current state in some ITS content.

As mentioned, the Evaluation feature is mandatory. It includes a Test feature that can be alternatively used by students in a Quick or customised (Custom) way. In addition, the Domain Model feature contains the curriculum (Curriculum feature) of a particular domain and a set of resources (Resource feature). A learner may use different types of resources, such as Problem, Essay, Activity, Help and Content. The Problem feature has an or-feature group representing the types of problems (True or False, Relate Columns, Fill the Gaps and Multiple Choice features) that could be selected in an ITS instance. The Help feature contains an or-feature group indicating the types of help that a user could ask in a particular product, i.e., Resolution, Diagnosis and Hint features.

The Course Management feature contains the functional requirements related to the administration of courses. This feature includes requirements such as create classes and associate teachers to classes. This feature has a Requires dependency with the Secretary feature. Besides this, as mentioned in the previous section, the Social feature can be optionally included in an ITS product. It has an or-feature group indicating that within this feature a product could have at least one of the Links to Social Network, Friends Management and Message Management features. The Report feature may be optionally included in ITS products. It has an or-feature group representing the different types of reports that could be selected, for instance, Student Report, Curriculum Report, Domain Report, Exercises Report, Gamification Report and Test Report.

After defining the feature model of GITS-PL, we specify it by using the ontology-based feature model strategy presented by Tenório et al. (2014). The OntoSPL ontology proposed by such authors represents the FODA notation by defining a set of classes and properties as well as the relationships of them. To use such ontology for specifying the feature model of a particular SPL, there is a need to import the OntoSPL ontology and create a set of instances to represent the feature model of an SPL. In this way, we created the GITS-PL.owl^3 ontology by importing the OntoSPL.owl^4 to contain the instances that represent the feature model of GITS-PL (see Figure 1).

In addition, the feature classes represented in OntoSPL ontology contain an OWL dataproperty that may represent if a particular feature instance is selected or not in a product. Each product that is based on a GITS-PL may have a specific configuration that is instantiated in a specific product ontology. The configurations of the products that can be derived by using GITS-PL are also specified in OWL files, which import the GITS-PL.owl ontology to allow the selection of the state of the feature for that specific product.

### 2.3 Architectural design

According to the recommendations for documenting software architecture (Garlan et al., 2010), we specify our PLA in terms of architectural views. In the following sections, we present the modules view and the components and connectors view models produced in the architectural design activity applied to generate the SPL architecture of GITS-PL. These architectural models have been mainly driven by the non-functional requirements (e.g., variability, reusability, flexibility and evolvability) identified in the Requirements Engineering phase as well as the functional requirements detailed in the feature diagram of Figure 1.
2.3.1 Modules view

Figure 2 presents the diagram that represents the modules view of our GITS-PLA. It describes at a high level the main modules of GITS-PLA and also illustrates how these modules are interconnected. This diagram contains the main architectural decisions that were made:

- use of a hybrid architectural style combining the layer, the kernel and satellites and the client/server styles to deal with the different non-functional requirements
- use of the layer architectural style to manage the complexity of the system and to separate the concerns involved in gamified ITSs
- use of the kernel (i.e., to include mandatory features) and satellites (i.e., to include the management of the variation points along with their variants) architectural style
- use of the client/server style since we are building web-based ITSs
- use of web services to enable the interoperability of distributed components and to also enable dynamic reconfiguration of components
- use of ontologies to manage the configuration of the variability of products because we are relying on it to guide the configuration of GITS-PL products (this mechanism will be further explained).

As seen in Figure 2, GITS-PL architecture contains four main layers: Interface Model, Web Services, Business Model and Persistence. The Interface Model layer contains the graphical user interfaces of the system and is located in the client side of the architecture whereas the other three layers are located on the server side of the architecture. The Web Services layer intermediates the access to the business rules of gamified ITS from the Interface Model layer. Moreover, the kernel portion of the architecture includes the Business Model layer. In this layer, we can find all subsystems that are related to the common business rules identified for gamified ITSs, for instance, the Student Model, Domain Model and Pedagogical Model. Besides this, this layer also manages the variable components that should be included in an arbitrary product of GITS-PL. The selection of the variable components occurs by binding the variants attached to the variation points (components that are subject to some selection, i.e., Gamification, Report, Social and so on) represented by the modules placed in satellites portion of the architecture (next to the kernel portion). The satellites modules are represented in the right side of the kernel. The variability management component is responsible for binding the variation points that are placed on such module.

To ‘know’ which configuration of a product is based on GITS-PL, the Business Model depends on the Persistence layer, particularly, on the Ontologies Management subsystem. This subsystem retrieves the selected variants for a specific product in an ontology that represents feature models. Recall that we are using the OntoSPL ontology since it presents an ontology for representing feature models that are more flexible and demand less time (Dermeval et al., 2015) to change in comparison with a well-known ontology for representing feature models (Wang et al., 2007). The Persistence layer is also responsible for managing the access to the data about the system, such as student modelling and resources.
2.3.2 Components and connectors view

The UML components diagram presented in Figure 3 represents an excerpt of the components and connectors view of GITS-PL architecture. It is a more detailed view of the modules and illustrates how the main components are connected (using connectors) with each other to support the commonalities and variabilities of gamified ITSs. The components of the diagram are represented by a UML component, and a connector is represented by a UML component annotated with a `<<stereotype>>`.

To derive the diagram, we relied on some steps of the process presented by Junior et al. (2010). The work proposed by Junior et al. (2010) presents a variability management approach for UML-based SPLs. In such approach, some activities (e.g., feature model definition, architectural design and so on) are defined to aid the development of UML-based SPL. Junior’s approach takes into consideration similar approaches such as Braganca and Machado (2006), Gomaa (2004), Korherr and List (2007) and Ziadi et al. (2003) to define a more comprehensive systematic approach to specifying SPL component models from the use case and feature models. Hence, we decided to use the guidelines presented in such approach to aid the definition of the GITS-PL component model following the activities presented in their systematic method.
An ontology-driven software product line architecture

with regard to the component model specification. In summary, the method defines a one-to-one relationship between a feature and a component, i.e., every feature of the feature diagram derives a component in the component diagram.

Figure 3 Excerpt of the GITS-PLA component diagram (focus on the ontology-driven variability management) (see online version for colours)

In the following, we depict some elements shown in the diagram. For the sake of clarity, some of the components and connectors of this architectural view are suppressed in the diagram, for instance, the components and connectors related to the Register, Resource, Evaluation, Social, Report and Strategy variation points (features) along with their variants. We also omit some elements of the Business Model layer (e.g., Student Model, Domain Model, Pedagogical Module and Login), Persistence layer (e.g., components related to the access to database), Service layer and Interface layer.

In Figure 3, the component diagram is presented aiming to focus on the configuration of a product receiving as input the choices of an author/user (e.g., teacher) on which gamified ITS features should be included. Thus, the component (UserInterface) of the Interface layer requires the operations provided by the Services Layer (WebServices component). The communication between these layers is intermediated by a connector (ServicesConn) that is connected to the components of both layers. The WebServices component provides to the Interface layer operations of the Business Layer. As seen in Figure 3, the ConfigurationMgt component is connected to the Services Layer (intermediated by the ConfigurationMgtConn connector) and to the Persistence Layer. It happens because this component may update (e.g., from a user input) or retrieve the current configuration of a product that is represented in the product ontology, which imports the GITS-PL ontology, as previously mentioned. Once the ConfigurationMgt components know the current configuration of a product, it provides to the VariabilityMgt connector the information that such element needs to fetch the variation points and variants according to the configuration of product instantiated in the product ontology.
Figure 4 presents a UML sequence diagram illustrating how the ontology-driven variability mechanism works in the GITS-PL architecture. As can be seen in the figure, the mechanism may start when a user configures a desired product in the Interface Model, then the interface calls an operation encapsulated in a Web Service. Then, the ConfigurationMgt component invokes the OntologiesMgt component to update the ontology of that gamified ITS product, which is based on GITS-PL. Afterwards, the ConfigureMgt component sets this new configuration to the VariabilityMgt component to fetch the variation points and to select the variants for each variation point according to the configuration required by a user.

2.3.3 Mapping features into components

On the basis of Junior et al. (2010), we mapped the features from the feature model (Figure 1) into the components included in the component diagram of GITS-PLA. As previously explained, each feature from the feature model is represented by a stereotyped component from the component diagram. Table 1 describes this mapping. As Figure 3 is an excerpt of the complete component diagram, some features, for instance, Evaluation and Course Management that are mapped in this table, are not shown in this figure.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Component</th>
<th>Stereotype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>Register</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Register &gt; Principal</td>
<td>Principal</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Register &gt; Secretary</td>
<td>Secretary</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Register &gt; Coordinator</td>
<td>Coordinator</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Register &gt; Teacher</td>
<td>Teacher</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Register &gt; Student</td>
<td>Student</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Student &gt; Student Model</td>
<td>Student Model</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>
Table 1  Mapping features into components (continued)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Component</th>
<th>Stereotype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy</td>
<td>Strategy</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Strategy &gt; PBL Aleatory</td>
<td>PBL Aleatory</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Strategy &gt; Static Sequencing</td>
<td>Static Sequencing</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Strategy &gt; Lesson Plan</td>
<td>Lesson Planning</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Course Management</td>
<td>Course Mgt</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Social</td>
<td>Social</td>
<td>Optional</td>
</tr>
<tr>
<td>Social &gt; Link to social network</td>
<td>Link to social network</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Social &gt; Friends management</td>
<td>Friends Mgt</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Social &gt; Message management</td>
<td>Message Mgt</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Evaluation</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Evaluation &gt; Test</td>
<td>Test</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Test &gt; Quick</td>
<td>Quick</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Test &gt; Custom</td>
<td>Custom</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Domain Model</td>
<td>Domain Model</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Domain Model &gt; Curriculum</td>
<td>Curriculum</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Curriculum &gt; Resource</td>
<td>Resource</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Resource &gt; Problem</td>
<td>Problem</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Problem &gt; True Or False</td>
<td>True Or False</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Problem &gt; Relate Columns</td>
<td>Relate Columns</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Problem &gt; Fill the Gaps</td>
<td>Fill the Gaps</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Problem &gt; Multiple Choice</td>
<td>Multiple Choice</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Resource &gt; Essay</td>
<td>Essay</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Resource &gt; Activity</td>
<td>Activity</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Resource &gt; Help</td>
<td>Help</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Help &gt; Resolution</td>
<td>Resolution</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Help &gt; Diagnosis</td>
<td>Diagnosis</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Help &gt; Hint</td>
<td>Hint</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Resource &gt; Content</td>
<td>Content</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Report</td>
<td>Report</td>
<td>Optional</td>
</tr>
<tr>
<td>Report &gt; Student Report</td>
<td>Student Report</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Report &gt; Curriculum Report</td>
<td>Curriculum Report</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Report &gt; Domain Report</td>
<td>Domain Report</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Report &gt; Exercises Report</td>
<td>Exercises Report</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Report &gt; Gamification Report</td>
<td>Gamification Report</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Report &gt; Test Report</td>
<td>Test Report</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Gamification</td>
<td>Gamification</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Gamification &gt; Points</td>
<td>Points</td>
<td>Or-Feature</td>
</tr>
</tbody>
</table>
Table 1  Mapping features into components (continued)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Component</th>
<th>Stereotype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gamification &gt; Level</td>
<td>Level</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Gamification &gt; Badges</td>
<td>Badges</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Gamification &gt; Mission</td>
<td>Mission</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Gamification &gt; Leaderboard</td>
<td>Ranking</td>
<td>Or-Feature</td>
</tr>
<tr>
<td>Gamification &gt; Progress Bar</td>
<td>Progress Bar</td>
<td>Or-Feature</td>
</tr>
</tbody>
</table>

3  On the use of GITS-PLA to build new products: an illustrative scenario on the math domain

This section aims to present how the software artefacts specified in the domain engineering process may be used to construct one or more gamified systems as well as implications for generating products based on GITS-PLA.

3.1  Application engineering by using GITS-PLA

To present how to use the GITS-PL architecture to construct gamified ITSs, we focus on the steps that should be performed by the main factors involved in gamified ITSs:

- authors (i.e., teachers)
- users (i.e., learners)
- developers.

In general, teachers are responsible for planning courses and managing their execution along with selecting the educational resources that a course takes into account. In this context, we may consider that a teacher could be an author who would configure a gamified ITS according to his or her preferences. An author may define the main elements (e.g., curricula structures of a domain, educational resources and so on) related to the ITSs. Moreover, they have to make decisions in which elements should be considered in a product regarding the three perspectives previously mentioned in this paper, i.e., the technological, pedagogical and motivational.

To create a gamified ITS product relying on GITS-PL architecture, an author may specify the domain of a tutor, the hierarchy of curricula within such domain as well as selecting specific configurations for the ITS. In the product configuration step, an author may select the features that he or she would incorporate in the gamified ITS according to the variabilities that have been foreseen in the feature model of GITS-PL (Figure 1) and, hence, in the GITS-PL architecture. In this way, he or she may define the educational resources to be included in the system, select general ITS features and select gamification elements of the system.

To illustrate the application of these steps, in the following we apply them in the basic math domain. Figure 5 represents an example of curricula within this domain in the Brazilian high-school context. Each curriculum has a set of educational resources that could be, for instance, a problem, essay, activity, help or content as previously explained.
In this way, after defining the domain, curricula structure of tutor and the educational resources, the next step to be performed is the selection of ITS features and gamification elements.

Figure 5 Excerpt of the gamified ITS domain model on basic math (see online version for colours)

The green features highlighted in the feature model of Figure 1 depict a possible configuration for the gamified ITS on the math domain. Thus, for enabling the GITS-PL architecture to reason on the feature model of a particular gamified ITS product and to fetch the proper variation points and variants, the product configuration of the gamified ITS should be represented in a specific ontology. Hence, the gamified ITS on the basic math domain is instantiated in the Math.owl ontology, which imports the GITS-PL.owl ontology setting the selected features for the product. Figure 6 illustrates the Math.owl product configuration ontology showing only features that may be selected in the application engineering phase. Since the mandatory features are already included in all products derived from the platform, they are not shown in the figure. For the basic math tutor, all green features highlighted in Figure 1 receive a true value with respect to their currentState dataproperties in the ontology, whereas, for the other features, they receive a false value in their dataproperty.

This way, since GITS-PLA may receive the selection of features from a particular author, the architecture may create a product ontology (e.g., Math.owl) containing the desired configuration requested by the author. The architecture configuration is driven by the ConfigurationMgt component, which is responsible for receiving the features selection by authors, creating the product ontology following their preferences (keeping a
reference to the selection of features for the derived product), and supporting the VariabilityMgt connector to fetch the components related to the selected features and to generate a specific gamified ITS following such configuration. Note that, for creating such ontology, the product ontology should have to import the GITS-PL.owl ontology. Then, with the new product generated, a learner may interact with the gamified ITS, which is intended to be personalised according to the author’s preferences.

**Figure 6** Ontology instances of the gamified ITS features on the basic math domain (see online version for colours)

3.2 Development costs, quality and time-to-market implications for GITS-PL

Thus, in this section, we discuss the implications of using the GITS-PL application engineering process, as well as the domain engineering one, under the view of the development team to construct gamified ITSs using GITS-PL – regarding the development costs, impact on quality and time-to-market.

To build a reusable platform, an organisation has to make an up-front investment in such creation before it can reduce the costs per product by reusing platform artefacts (Pohl et al., 2005). However, literature revealed that the accumulated costs needed to develop systems from an SPL platform pay-off around three systems (called break-even point) in comparison with the costs for building the same number of systems in a single way – i.e., without a reusable platform. Thus, we expect that the investment required to perform the domain engineering of GITS-PL be returned after creating at least three gamified ITS products. Moreover, as our architecture intends to support the application engineering in an automatic way through the use of ontology-based feature modelling, it could decrease even more the costs for instantiating new products after the break-even point.

In the context of SPLs, the software artefacts produced may be reviewed and tested in many products since they have to prove their proper functioning in more than one kind of product. In this way, the extensive quality assurance implies a higher chance of detecting faults and correcting them, hence increasing the quality of products. In view of that,
we also expect that gamified ITS products derived from GITS-PLA have a higher quality in comparison with creating products using individually other reuse strategies (e.g., frameworks). As previously explained, PLAs include a variety of similar products in terms of architectural elements and variability might be achieved by delaying architectural design decisions, which are described through variation points and variants (van Gurp et al., 2001). Thus, for example, in comparison with frameworks, which might require the implementation of a new feature/component for a specific product, applying architecture-based customisation through SPL might add value to the quality of a product development by enabling delayed decisions of already built components.

In addition, a very critical success factor for a product is the time-to-market. If we consider single-product development, the time-to-market mostly comprises the time to develop the entire product. In SPL, the time-to-market is initially higher, as all software artefacts, which consider the variabilities and commonalities, have to be built first (Pohl et al., 2005). However, since the domain engineering is finished, the time-to-market is considerably reduced, as many artefacts can be reused for each new product. Hence, we believe that this characteristic could be offered by GITS-PL. Moreover, since the application engineering is automatic – which is expected to be supported by the GITS-PLA – it could significantly reduce the time-to-market to generate new gamified ITSs.

3.3 Support for evolving/changing gamified intelligent tutoring systems

In SPLs, it is possible to better organise development for evolution of the product range and reduce the effort compared with single-system engineering. In the context of GITS-PL, the automatic application engineering can be achieved by the use of ontologies as well as by the use of an ontology-driven mechanism to reconfigure products based on GITS-PL architecture. Thus, the automatic reconfiguration capabilities provided by our architecture could enhance the evolution of SPL platform at development time and the changing of products at runtime.

Aiming to support the evolution of GITS-PL, our architecture may favour the inclusion of new features in the platform. GITS-PL architecture foresees the specific components that are necessary to be changed to support such evolution. For instance, if there is the need to include a new variation point, the development team should perform the following activities:

- implementing the components of the new variation point and its variants along with implementing a connector to manage the variants
- modifying the VariabilityMgt component (see Figure 3) to consider the new components as well as to connecting the ConfigurationMgt component to it
- creating new instances – this creation could also be automated using SPARQL queries – in the GITS-PL.owl ontology
- creating new operations in the WebServices component to consider the inclusion of new features aiming to provide to the interface a way to select the new features by a user.
Afterwards, the new features could be selected in every product derived from GITS-PLA.

In a similar way to the evolution of the platform with new variation points, the evolution in GITS-PL architecture to include new variants associated to variation points that were already considered in the platform is even more straightforward. For instance, to include a new alternative feature (e.g., a new gamification element such as an Avatar) within an existing variation point (in the case of avatar, the Gamification variation point), the number of activities to perform the inclusion is reduced. It would be necessary to:

- implement the component related to the new variant and modifying the connector of the variation point that the new variant belongs to
- create a new instance in the GITS-PL.owl ontology representing this new variant in the ontology
- modify the operation related to the variation point in WebServices component to take into account the new variant to be included in the platform.

Moreover, GITS-PLA may also support the reconfiguration of products derived from the platform at runtime. Our ultimate intention (not necessarily in the scope of this paper) with GITS-PLA is to provide a single instance that might enable features reconfiguration/changing at runtime. Developing a dynamic SPL in the e-learning domain (e.g., gamified ITS) can be considered an open challenge (Capilla et al., 2014; Bittencourt et al., 2012) since it may consider different product requirements, several domains, large-scale instantiation, products authoring and quick instantiation of products by authors. Thus, in the context of developing a dynamic SPL for gamified ITS, evolution might be achieved by supporting reconfiguration of products derived from the platform at runtime, following the approach proposed by Tenório et al. (2014). And then, by fetching the software components related to the new configuration. The modification of an ontology can be performed by using, for example, SPARQL queries. An SPARQL query is an SQL-like language that allows to execute operations (i.e., create, retrieve, update and delete) in ontologies. To illustrate the implementation of an SPARQL query that could be automatically executed within our architecture, we present a reconfiguration scenario where an optional feature (i.e., True or False feature) that was not previously selected (see the green features of Figure 1) is selected in the product configuration of the gamified ITS on the math domain. The SPARQL query in Figure 7 specifies that an arbitrary instance, called ?x, in which its current state is the ‘eliminated’ value and that has the name ‘True or False’ should have changed its current state to the ‘selected’ value. Hence, since a new configuration is represented in the ontology of a specific product, it is possible for the GITS-PLA to retrieve such a new configuration and to enable the new feature that was already implemented in the architecture but was disabled for the previous configuration of the product at runtime.

![Figure 7](image-url) Example of a SPARQL query for reconfiguring a gamified ITS product by selecting an optional feature

```sparql
PREFIX gits-pl: <http://nees.com.br/ontologies/gits-pl/GITS-PL.owl#>
PREFIX math: <http://nees.com.br/ontologies/gits-pl/math.owl#>
DELETE {
  ?x gits-pl:current_state "eliminated";
}
INSERT {
  ?x gits-pl:current_state "selected";
}
WHERE {?x gits-pl:name "True or False")
```
4 Related works

There are few studies that use the concept of SPLs for developing ITSs (Mitrovic et al., 2009; da Silva et al., 2011). We also found two papers that propose an ontology-driven architecture in the context of ITS (Mavani, 2010; da Silva et al., 2011).

Mitrovic et al. (2009) present ASPIRE, which is an authoring environment that enhances the speed and facility to develop constraint-based tutors that automatically deploy tutoring systems on the web. The system’s primary goal was to reduce the quantity of preliminary knowledge required from authors (i.e., artificial intelligence and programming) to develop ITSs. Although ASPIRE was developed as an authoring tool architecture to be used even by non-experts – that in some way is similar to the idea of automatically performing the application engineering process that we propose with our architecture – the authors do not consider the motivational perspective of ITS through the use of gamification. Moreover, Mitrovic et al. (2009) do not benefit from the strategic reuse provided by SPLs in their work as well as do not rely on ontology capabilities to generate new tutors.

da Silva et al. (2011) use ontologies in the context of ITS to provide a semantic and consistent description of ITS knowledge. The work describes a model to develop ITSs based on the use of SPLs and ontology. However, even though presenting a platform for constructing ITSs that use SPL and ontology concepts, da Silva’s work do not consider in their architecture the motivational perspective that we take into account in our architecture. As previously mentioned, the motivational perspective through the use of gamification, for example, is of great importance to engage students and to drive desired learning behaviours.

González et al. (2014) propose a conceptual architecture for building ITS taking into account gamification elements that are used in the creation of EMATIC (i.e., an ITS oriented to digital tablets and mobile devices). The gamification elements are integrated in several modules of the system, such as game aesthetic in the student model’s module and game feedbacks in the visualisation module. However, this architecture does not foresee the high variability (i.e., do not use SPL) that could be applied to use ITS together with gamification as well as the need to personalise gamification elements according to some author’s preference. In this way, EMATIC’s architecture does not consider the high level of variability in the three perspectives (technological, pedagogical and motivational) that our SPL architecture does.

Mavani (2010) addresses ITS and proposes an architecture blending ITS and reciprocal tutoring system (RTS). The semantic web is used to build an appropriate infrastructure for the intelligent agents since the system uses an agent-based approach. Although Mavani’s work proposes the concept of dynamic growth of learning objects to increase their reuse, it is not beneficial from the strategic reuse provided by SPLs in their work as well as do not rely on ontologies capabilities to generate new ITSs. Moreover, Mavani’s work does not consider in their architecture the motivational perspective that we take into account in our architecture.

Despite the existence of some works that propose some architectural contributions using, in isolated ways, SPL in conjunction with ITS (and ontology for the case of da Silva’s work) as well as gamification together with ITS, we have found no previous work (Table 1) that uses ontology to drive the automatic configuration of products based on
SPLA, which aims to support the mass construction of personalised ITSs with clear focus on the motivation of students, i.e., using gamification elements.

5 Concluding remarks

The main contribution of this work is to define a platform architecture that considers three perspectives that are of utmost importance to the successful application of web-based ITSs. Our platform takes into account commonalities and variabilities of software requirements, individualised tutoring as well as learner’s motivational aspects in the context of web-based ITS. Moreover, our architecture is designed to provide an automatic application engineering by reasoning on features models (represented by ontologies) and by configuring the components to be included in a specific product according to the configuration represented in the feature model.

The expected results for using GITS-PLA include the possibility to generate personalised gamified ITS with lower development costs, with improved quality and with shorter development cycles (reduction in time-to-market). In addition, adding game elements to online learning contexts could have the power to engage students and to drive desired learning behaviours. Moreover, as we are using an ontology-based feature modelling – which presents a high level of flexibility – it may support the evolution of the SPL platform at development time and the reconfiguration of the features to be included in a particular product at runtime.

We are currently implementing the architecture presented in this paper as well as we intend to test it in real situations. We also plan to design a controlled experiment to empirically compare – regarding the development costs, quality, time-to-market and support for the evolution/changing attributes – the generation of products using our SPL architecture in comparison with the current development process to create a gamified ITS in the context of Meu Tutor Company. Finally, we also intend to develop a dynamic SPL based on the architecture investigating design trade-offs between evolvability and maintainability.

References


An ontology-driven software product line architecture


**Notes**

2]This exam is used by public and private universities in Brazil to select the entry of new students in college.