An Approach for Developing Groupware Product Lines Based on the 3C Collaboration Model

Bruno Gadelha[1], Ingrid Nunes[1], Hugo Fuks[1], Carlos J. P. de Lucena[1]

[1] Department of Informatics, Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
R.M.S Vicente, 225, Gávea, Rio de Janeiro - RJ, Brazil, 22453-900
{bgadelha, ionunes, hugo, lucena}@inf.puc-rio.br

Abstract. Software Product Lines (SPLs) are a new software engineering technology that aims at promoting reduced time and costs in the development of system families by the exploitation of applications commonalities. Given that different Groupware applications typically share a lot of functionalities, Groupware Product Lines (GPLs) have emerged to incorporate SPL benefits to the Groupware development. In this paper, we propose an approach for developing GPLs, which incorporates SPL techniques to allow the derivation of customized groupware according to specific contexts and the systematic reuse of software assets. Our approach is based on the 3C Collaboration Model that allows identifying collaboration needs and guiding the user to select appropriate features according to their collaboration purpose. A GPL of Learning Object repositories, named FLOCOS GPL, is used to illustrate the proposed approach.

Keywords: groupware development, software product lines, learning objects.

1 Introduction

Over the past decades, the use of applications to support the collaboration among groups, namely Groupware, is gaining significant popularity. This popularity is due to the fact that sharing the same space and time is becoming a hard task for people with busy schedules. This kind of applications shares a particular set of common requirements [1], resulting in the need of specific software development techniques to support groupware development. One key issue in groupware development is the huge amount of time wasted on implementing infrastructure aspects like protocols, synchronism, session management and others, leaving little time for implementation of innovative solutions [2]. In addition, there are several common functionalities shared by different groupware applications, such as forum and chat, resulting in the development of applications with a common core, but differ in some functionalities, which are according to specific contexts. This scenario calls for approaches that take advantage of these common functionalities, while allowing the customization of applications.

This scenario has been explored by Software Product Lines (SPLs) [3, 4], a new trend in the context of software reuse. SPLs investigate methods and techniques to build and customize families of applications through a systematic method. This approach exploits the commonalities among family members promoting several benefits, such as a reduced time-to-market, lower development costs and quality improvement. A SPL [3] refers to a family of systems sharing a common, managed set of features to satisfy the needs of a selected market and that are developed from a common set of core assets in a prescribed way. According to Czarnecki et al. [5], a feature is a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate among products in a SPL.

The issues previously mentioned found on groupware development have also been addressed by several groupware component-based approaches [6, 7]. These approaches have used components [8] mainly for two reasons: product lines and tailorability. Despite accelerating groupware implementation, we have identified some deficiencies in the proposed groupware componentization approaches: (i) they do not cover the domain analysis phase. Domain analysis is essential to capture commonalities and variabilities in a domain, and therefore to identify the features to be supported by the components. In addition, domain analysis defines the scope of the SPL, stating the possible products to be derived from the SPL; (ii) the approaches propose to assemble service components to build customized products; however they do not provide means to deal with fine-grained variabilities allowing customization into the services; and (iii) they do not provide feature traceability. It is essential in SPLs to trace features, because it allows one to know which components implement which features, thus enabling the selection of appropriate artefacts of a SPL in the product derivation process.

In this paper, we present an approach that addresses these identified deficiencies in the development of families of groupware, which we refer to as Groupware Product Lines (GPLs) from now on. The main goal of the approach is to develop GPLs leveraging practices of the SPL development in order to incorporate the benefits provided by SPLs. A GPL is composed of a reusable infrastructure that provides common groupware services and has a production plan based on feature traceability to allow the derivation of applications customized to users' specific needs. The features to be addressed by the GPL are identified and defined in the domain analysis, which is based on the 3C Collaboration Model [9, 10], whose purpose is to classify features into three categories (Cooperation, Communication and Coordination) and to guide the user to select appropriate features according to his needs. Our approach is illustrated with the FLOCOS GPL, which consists of a product line of Learning Object repositories.

The remainder of this paper is organized as follows. Related work is presented in Section 2. Section 3 presents and details the proposed approach for developing GPLs. In Section 4, we describe the FLOCOS GPL development in order to illustrate our approach. Section 5 presents and discusses relevant issues that arose from our experience with the GPLs. Finally, conclusion and directions for future work are presented in Section 6.

2 Related Work

Several component-based approaches have been proposed in the literature in order to minimize the efforts in the groupware development. The main idea of these approaches is to encapsulate many of difficult technical aspects, e.g. session management and communication protocols, which are typically present in the groupware development. Among the motivations of component-based development, these proposals are mainly concerned with tailorability [11, 12] and product lines [7, 13, 14]. The first is related to the idea of assembly by the final user. In the latter, components are developed with the aim of reusing them in various systems, reducing the total investment and maintenance costs.

Examples that illustrate these approaches are: (i) GroupKit [13], which addresses the development of synchronous groupware by providing a toolkit that hides from the programmer complexities intrinsic to this type of application; (ii) FreEvolve [11], which was designed to enable groupware customization through composing components or tailoring existing applications to final users; (iii) DreamTeam [14], which is a component-based platform to support building synchronous groupware; and (iv) Gerosa et al. [6, 7] propose structuring collaborative systems using components that encapsulate the technical difficulties of distributed and multi-user systems and reflect the concepts of collaboration modelled by the 3C model.

Despite the fact that these approaches provide useful techniques to reduce groupware development efforts and to tailor them to specific contexts, they do not cover the whole GPL development process. They lack some essential activities such as the domain analysis, which is responsible for eliciting common and variable requirements of the GPL, and tracing features, which provide means of systematically derive customized groupware from a GPL, based on a feature selection. In summary, the approaches aim at developing reusable software assets or even provide them, but these assets are reused in an *ad hoc* way.

Based on these identified deficiencies, we propose our approach for developing GPLs in next section. The approach incorporates activities of the SPL development, resulting in the evolution of the reuse from an *ad hoc* to a systematic way.

3 An Approach for Developing Groupware Product Lines

In this section, we present and describe our approach for developing GPLs. It incorporates activities and model from both groupware and SPL development. The approach not only provides means of developing reusable assets to build groupware, but also concerns with identifying the GPL features and providing feature traceability, whose main purpose is to allow a systematic derivation of members of the groupware family.

As a motivation for our approach, we present a typical scenario of families of groupware. The family is a set of Learning Management Systems (LMSs). Several LMSs can be seen available on the market or in academic environments. The members of the LMS family have a set of common collaborative services, such as e-mail, chats, discussion forums, links and repositories. However, other services are present in just some or even in only one LMS. These commonalities and variabilities are depicted in Table 1 (adapted from [7]). It shows collaboration services found in the following LMSs: AulaNet (http://www.eduweb.com.br), TelEduc (http://teleduc.nied.unicamp.br), AVA (http://ava.unisinos.br), WebCT (http://www.webct.com) and Moodle (http://www.moodle.org). The use of a GPL approach allows the development of reusable assets, and to derive each one of these LMS configurations in a systematic way. Moreover, any valid combination of the features can be derived from this reusable infrastructure to meet specific user needs.

Table 1. Collaborative Services in LMS [7].

	Communication Services							Coordination Services									Cooperation Services												
	Mail	Discussion List	Forum	Mural	Brainstorming	Chat	Messenger	Agenda	Activities Report	Participation Monitoring	Questionnaire	Tasks	SubGroups	Resource	Guidance	Voting	Repositories	White Board	Search	Glossary	Links	Cooperative Journal	Classifier	Wiki	Contact Manager	Peer Review	FAQ	Notes	RSS
AulaNet	X	X	X			X	X		X	X	X	X	X			X	X				X							X	
TelEduc	X		X	X		X		X	X	X	X	X	X				X				X						X	X	
AVA	X		X	X		X		X	X	X	X	X			X		X			X	X						X	X	
WebCT	X		X			X		X	X		X	X				X	X	X	X	X	X							X	
Moodle			X			X	X	X	X	X	X	X	X			X	X		X	X	X	X		X		X	X		X

Analyzing groupware families, like LMS for example, as a whole (domain analyses) and its possible variations makes possible to deploy customizable groupware according user's needs.

Next sections describes the two main steps of our approach, which are: (i) Domain Analysis, which analyzes the domain using the 3C Collaboration Model to guide the requirements elicitation and identifies common and variable features (Section 3.1); (ii) Domain Design and Implementation, which aims at defining an architecture that supports the defined variability and the derivation of applications in a systematic way (Section 3.2).

3.1 Domain Analysis

In the domain analysis, the main concepts and activities in a domain are identified and modelled using adequate modelling techniques. The common and variable parts of a family of systems are identified, defining the scope of the GPL indicating which products can be derived from it. The GPL domain analysis differs from the SPL domain analysis because of the need of the collaboration analysis. The analysis of Collaboration in our approach is made in accordance with the 3C Collaboration Model [9, 10] as illustrated by Figure 1.

During communication there is an exchange of messages targeting future common action. Coordination deals with the resources and their interdependencies needed for accomplishing the accorded plan of action. Cooperation is the action realized by the joint operation of group members in the shared space. Collaboration, therefore, depends on the interplay of these dimensions. Using this model as a guide to develop groupware means developing tools to support each "C" in the model.

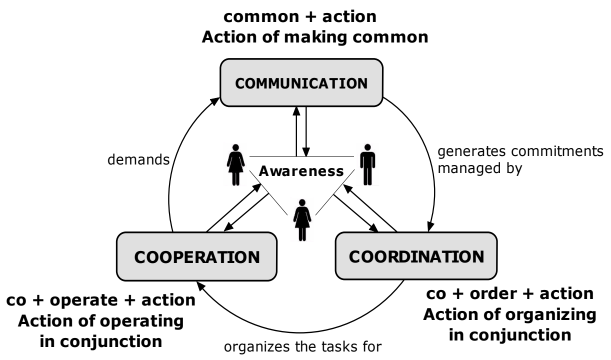

Fig. 1. 3C Collaboration Model

To support communication, [15] says that the designer of communication tools defines the elements that will set the communication channel between the interlocutors, considering the specific groupware user needs like purpose, dynamics, time and space. According to group needs, aspects like privacy and information overload must be analyzed.

To support coordination, the designer must think about what must be coordinated. If the aim is to coordinate people, the focus of the coordination must be the tools that provide communication and the context. Coordinating resources is related to the shared space where actions are made (cooperation). Coordinating tasks consists in managing interdependencies between tasks that are carried out to achieve a goal [15]. So, the designer must keep these different aspects of coordination when designing tools to support this task.

To support cooperation, the designer must think on how the actions on the shared space will take place. A set of tools for storage and maintenance of artefacts (documents, spreadsheets, presentations and others) must be present. Depending on the objective of the group, it is necessary some tools for online synchronous edition of these artefacts.

After identifying GPL requirements in accordance to the 3C Collaboration Model, variabilities inside the domain must be identified. For instance, a GPL of groupware whose purpose is cooperation may need a feature to provide communication. In one product of the GPL, a synchronous communication is need and in another product, an asynchronous one. This results in a feature “Communication” that has two different alternatives for instance chat and e-mail. Feature modelling is the activity of analyzing and capturing the common and variable features of SPLs and their respective interdependencies, which is introduced in the GPL development. It was originally proposed by the FODA method [16], and has been used in several SPL approaches in order to analyze the domain and its variabilities. According to Czarnecki et al. [5], a feature is a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate among products in a SPL. Features are essential abstractions that both customers and developers understand.

The features are organized into a coherent model referred to as a feature model, which models the features of a GPL as a tree, indicating mandatory, optional and alternative features. This tree representation is named feature diagram. Mandatory features are the ones that must be present in any derived product of the GPL, and is part of the identity of the product. For instance, in a Learning Object repository, there must be functionalities that allow storing and retrieving them. Optional features aggregate functionalities to a derived product that are necessary just in certain context, so it can or not be present in a product of the product line. Alternative features in turn are features that vary from one product to another. An example in collaborative systems is that communication can be achieved by exchanging asynchronous messages, e.g. e-mail, or synchronous, such as chat.

A feature model refers to a features diagram accompanied by additional information such as dependencies among features, and it represents the variability within a system family in an abstract and explicit way. The constraints provide information about valid configurations of the feature model. For instance, two different features can be mutually exclusive, or the inclusion of a certain feature requires the inclusion of another.

In order to model GPL features, we propose an extension of the feature model to provide information of both collaboration and variability concerns. We have named this extended version 3C-Feature Model. In collaboration systems, there are several supporting features that are intrinsic to this kind of system. As a consequence, the feature model is split into two layers: functionality layer and infrastructure layer. The first refers to the features that provide functional behaviour to the user; however features are represented by different symbols to indicate the purpose of the feature (Communication, Cooperation and Coordination) according to the 3C Collaboration Model. These notations are complementary to the ones already used in traditional feature models, which are: (i) filled circles to represent mandatory features (present in all products of the GPL); (ii) empty circles to represent to represent optional features (present in some product of the GLP); and (iii) a line connecting alternative features (one of them must be chosen for the product being derived). The latter is composed of features that support some functional features. A feature model with our proposed adaptations is presented in Figure 4.

3.2 Domain Design and Implementation

The design and implementation of a GPL must result in an architecture that supports the identified variability, which characterizes the main difference of GPL and single groupware architectures. The GPL architecture is modelled using typical UML diagrams; however they are adapted to provide explicit variability information. We adopted the stereotypes <<*kernel*>>, <<*optional*>> and <<*alternative*>>, proposed by the PLUS method [17], to indicate which design elements are part of the GPL core, are present in some products and varies among products, respectively.

Supporting variability is typically achieved by taking into account feature modularization and adopting techniques to promote it. The main benefit of having features modularized is that it allows an easy (un)plugging of a feature; therefore it improves the systematic derivation of products and consequently reduces time and costs of the derivation process. Different implementation techniques can be used to modularize features in the code [18], e.g. polymorphism, design patterns, frameworks, conditional compilation and aspect-oriented programming. A particularity of

Groupware is that functionalities typically rely on common services. For instance, chat, e-mail and forum are three different options to support communication. Therefore, we propose the idea of having a Layered architecture [19], in which one of the GPL architecture layers provides the business services, which uses services of the next layer, which in turn provides infrastructure services.

In addition to these techniques, component-based SPL development approaches propose the decomposition of the SPL architecture into components with well-defined interfaces, and therefore they can be easily changed. These approaches have the same ideas component-based groupware development approaches. Consequently, this convergence of interests calls for studies that investigate how these approaches can be used together and how they can be used in the GPL development; however this has not been addressed in this paper yet.

Besides modelling and implementation the GPL reusable assets, it is important to keep traceability links between features and the elements that implements them. This information allows choosing the appropriate implementation elements according to the selected features during the product derivation process.

Figure 2 summarizes the idea of our approach. The main result of the domain analysis is the 3C-Feature Model. Later, the GPL is designed considering the identified variability and, in sequel, the reusable assets are implemented. Links among the GPL artefacts ensures the traceability of the feature in order to allow an effective product derivation process. In our approach, these traceability links are expressed by means of tables, which map features to classes that implement them.

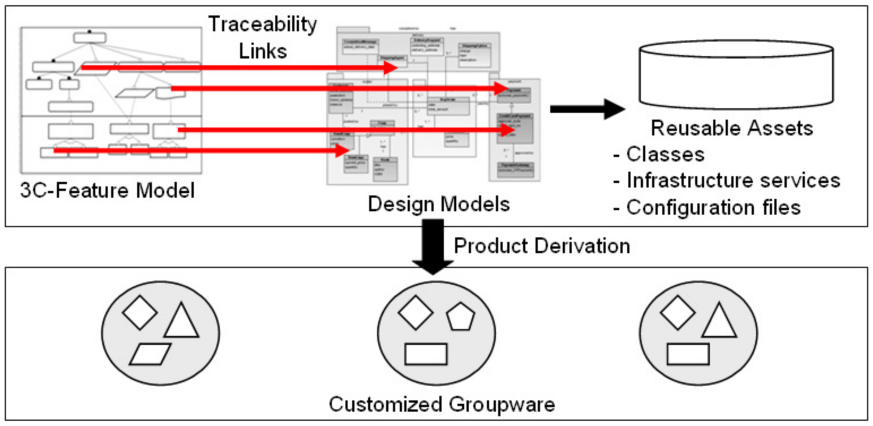

Fig. 2. Approach Overview

4 Illustrating our Approach with the FLOCOS GPL

This section presents and describes the FLOCOS (Functional Learning Object Collaborative System) GPL, using the approach previously presented. FLOCOS consists of a GPL of Learning Object (LO) repositories based on 3C Model. FLOCOS products are a particular kind of LO repositories because it supports a special kind of LOs, not dealt with by other works: Functional Learning Objects (FLOs) [20]. In order to familiarize the reader with the FLOCOS GPL domain, we first present relevant concepts related with FLOCOS (Section 4.1), and later we describe how the FLOCOS GPL was developed using our approach (Section 4.2).

4.1 FLOCOS Overview

According to SCORM [21], LOs have the following characteristics (the "ilities"): (i) Accessibility: the ability to locate and access instructional components from one remote location and deliver them to many other locations; (ii) Adaptability: the ability to tailor instruction to individual and organizational needs; (iii) Affordability: the ability to increase efficiency and productivity by reducing the time and costs involved in delivering instruction; (iv) Durability: the ability to withstand technology evolution and changes without costly redesign, reconfiguration or recoding; (v) Interoperability: the ability to take instructional components developed in one location with one set of tools or platform and use them in another location with a different set of tools or platform; and (vi) Reusability: the flexibility to incorporate instructional components in multiple applications and contexts.

Considering LOs software artefacts that have the aforementioned "ilities", Java applet applications, web services, web applications, software components and software agents, all fall into this category. Gomes et al. [22] propose the definition of Functional Learning Objects, stating that they are computational artefacts having functionality that enables interaction between entities, whether digital or not, being used/reused in the mediation of the teaching-learning process. They propose a Functional Learning Object Metadata (FLOM) to properly describe the technical characteristics of this class of objects.

Even though FLOs adequately support teaching and learning processes, they tend to require a major implementation effort demanding a skill rarely found in docents. So, in order to successfully meet the increasing demand for these objects, promoting reusability and tailorability is essential.

Therefore, FLOCOS products promote accessibility, reusability and affordability by the use of FLOM for describing FLOs. The products provide a shared space in which users can collaborate by posting, maintaining, searching and discussing FLOs. These functionalities provided by FLOCOS can be customized according to specific user needs, or even whole functionalities can be needed just in some contexts. In next section, we present the development of the FLOCOS GPL.

4.2 Developing FLOCOS GPL

The FLOCOS GPL developing process encompassed mainly two steps: (i) the domain analysis, where the features provided by GPL were identified and classified in two dimensions: first according each C of 3C Model, and after they were classified as mandatory and optional (Section 4.2.1); (ii) the design and implementation, in which the FLOCOS GPL architecture was designed and implemented supporting the variability identified in the previous step and then implementation elements were mapped onto GPL features for traceability purposes (Section 4.2.2).

4.2.1 Domain Analysis

FLOCOS GPL purpose is to give support to the cooperation among members of a group in a shared space. However, even when focusing on a specific "C" of the 3C Collaboration Model, in this case cooperation, communication and coordination should also be treated, given that there is also an intra-relationship between them [23].

In this context, each FLOCOS GPL product is a shared space comprising the FLOs, and the discussion, experience reports, message logs, recommendations, user notifications and action histories related to each one of them. Figure 3 illustrates members cooperating on a Functional Learning Object.

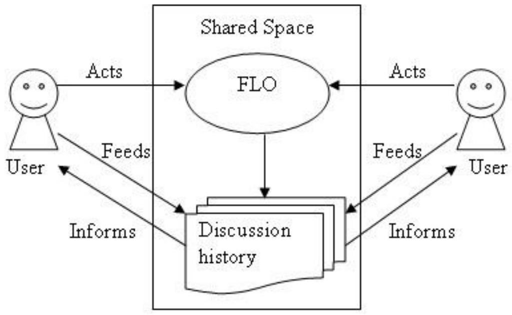

Fig. 3. FLO Cooperation Model

The first step consisted on identifying the features that the GPL must provide to support the FLO Cooperation Model presented in Figure 3, as follows:

- **Communication:** the feature identified to support this “C” is the Discussion. Two kinds of communication are available for FLOCOS products: chat (synchronous) and forum (asynchronous). Therefore, the best communication form can be chosen for a certain context;
- **Coordination:** The features identified to support this “C” are *User registration* and *Notifications*. The *User registration* feature is related to the coordination of the shared space once it controls who will operate in it. The feature *Notifications* is related to the coordination of the tasks once it alerts for events occurred in the shared space and is expected some reaction of the user after the notification;
- **Cooperation:** The features identified to support this “C” are *FLO registration*, *Experience reports*, *Actions history* and *Recommendations*. These features are related to the actions taking place in the shared space. Users act over FLOs registered in the system, relate their experience on FLO usage, the system recommends FLOs to users according their preferences and users can view what other users do with the FLOs registered.

Table 2 shows the classification of FLOCOS GPL features on the 3C Model.

Once the features were identified, they were analyzed and classified into three different categories: mandatory, optional and alternative. Mandatory features are essential to any LO repository, optional features are necessary just in some specific contexts, and alternative features varies from one product to another.

Table 2. FLOCOS GPL Features and 3C Model.

	Communication	Coordination	Cooperation
User registration		▨	
FLO registration			▨
Discussion	▨		
Experience reports			▨
Actions history			▨
Recommendation			▨
Notifications		▨	

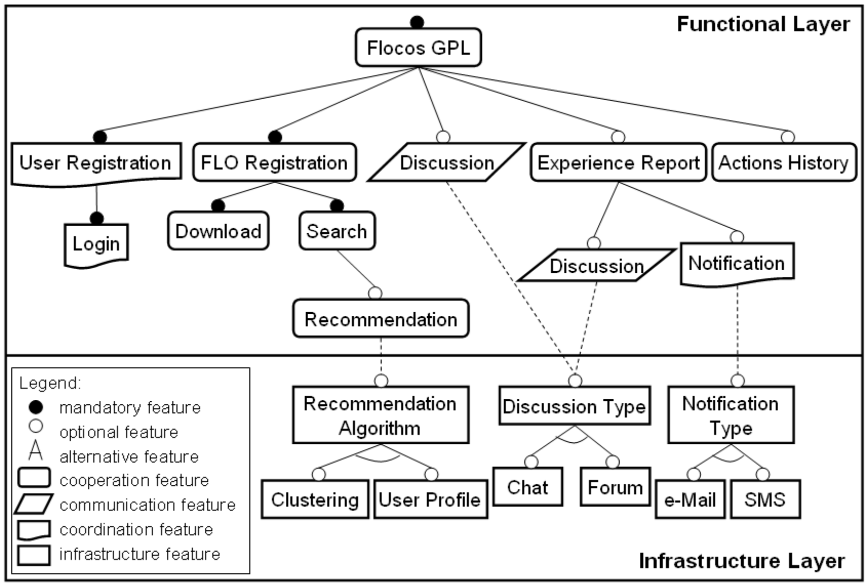

Fig. 4. FLOCOS GPL 3C-Feature Model

The identified functional features were then organized in FLOCOS feature model, which is depicted in Figure 4. The feature model not only shows features with their variability information (mandatory and optional), but also their purpose. The purpose of the products derived from the FLOCOS GPL can be seen be the root of the feature model (Cooperation). Additionally, some constraints were specified in order to indicate dependencies among the features. These are the constraints:

1 Report discussion is available only if the Discussion feature is selected for a product;
2 User notification feature depends of the presence of Experience reports feature on the product;
3 FLO recommendations feature is available only if Actions history feature is selected for a product.

The information provided by the feature model and its constraints indicate the possible products that can be derived of the FLOCOS GPL.

Next, we describe the functional features that comprise the FLOCOS GPL:

- **User registration.** This is a mandatory feature. To access FLOs deployed in FLOCOS, the user must access the system. To do so, he must sign up through this feature. Once registered, the user is able to access other functionalities.
- **FLO registration.** That is the core functionality of LO repositories, and therefore a mandatory feature of FLOCOS GPL. Users, duly registered, can post FLOs, making them available to other system users.
  - **Recommendation**. FLOCOS enables its users to assign ratings for FLOs evaluation. These ratings provide feedback to the creators and maintainers of the FLO and are used to guide the search for FLOs in the system. The search engine ranks the results based on these ratings plus access and download statistics provided by the Actions history. From the 3C Model viewpoint, it is a functionality with a cooperation purpose. In addition, this feature presents an autonomous behaviour considering that it automatically monitors the user actions to evaluate FLOs. Two options for recommendation algorithms are provided: clustering and based on user profile.
- **Discussion.** A discussion is associated with each FLO registered in the system through the Discussion functionality. FLO users may report their experiences and discuss deficiencies in the development and application of the object in different contexts. This is an optional feature in the GPL, given that a customer may require a simple repository needing no discussion. The Discussion feature has an alternative infrastructure feature associated to it – chat and forum – therefore, a derived product can have the most appropriate discussion type.
- Experience report. Similarly to the Discussion, each FLO registered in the system is associated with experience reports. Such reports inform experiences of successes and failures in the use of a specific FLO available in the system. This is another optional feature. For example, in the early stages of a FLO development, a simple repository associated with discussions would suffice the developers' needs. This feature has two sub-features. The Discussion is similar to the one described previously.
  - **Notifications**. In order to inform users about actions taken on FLOs posted by them, they receive a notification for every message sent to a forum, or to each experience report. This is also an optional feature, considering that some users may consider these notifications irrelevant or feel overloaded by the massive amount of messages generated by the feature. From the 3C Model viewpoint, it is a functionality with a coordination purpose. As the system must be monitored to send messages notifications, this feature also provides an autonomous behaviour.

- **Action history**. FLOCOS keeps a history of the actions of its users in the environment. Through this functionality, it is possible to know what FLOs are more accessed those with the more active discussions, more active users in the environment and so on. This is an optional feature of FLOCOS GPL, given that keeping log register may cause an overhead in the system, which is a cost that some users would not like to pay.

4.2.2. FLOCOS GPL Design and Implementation

In order to provide a flexible architecture to support the variability provided by the FLOCOS GPL, the architecture of the GPL adopts several software engineering practices, such as design patterns and agent technology, to bring low coupling among the different features of the GPL.

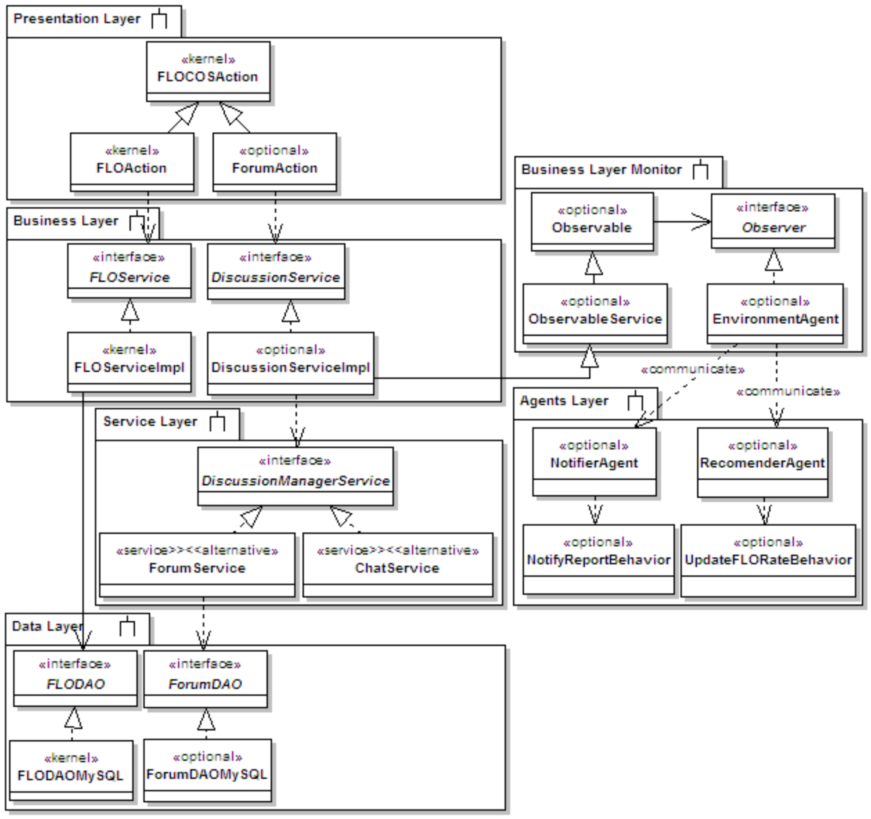

Fig. 5. FLOCOS GPL Architecture (partial)

Giving that FLOCOS products are web-based applications, a pattern widely used to structure this kind of application is the Layer architecture pattern [19]. The layers that comprise the architecture are: (i) Presentation – responsible for processing the web requests submitted by the system users; (ii) Business – responsible for structuring and organizing the business services; (iii) Service – responsible for providing infrastructure services that can be used by the business services; and (iv) Data – aggregates the classes of database access of the system, which was implemented using the Data Access Object (DAO) design pattern. Figure 5 shows a partial view of FLOCOS GPL class diagram illustrating its architecture. The four presented layers are present at the left side of the figure, being composed of classes whose purpose is according to which layer it is part of. As it can be seen, each feature is implemented by a set of classes (Action, Service, and DAO) that is independent of other features. Moreover, the <<*kernel*>> and <<*optional*>> stereotypes indicate which classes are part of the GPL core and which are not.

Some features in FLOCOS GPL provide an autonomous behaviour. Software agents are used to implement these features, because they are a powerful abstraction to model this kind of behaviour. A software agent enjoys mainly the following properties: autonomy, reactivity, pro-activeness and social ability. Nunes et al. [24] proposed the Web-MAS architectural pattern, based on the Layer architecture pattern, which allows the extension of web applications to incorporate autonomous behaviour. So, we adopted this pattern, adding the Business Layer Monitor and Agents Layer components to FLOCOS GPL architecture (see Figure 5) to implement the *User Notifications* and *Recommendation* features. These two components are at the right side of Figure 5. The Business Layer Monitor contains the *EnvironmentAgent*, which monitors and propagates the execution of business operations performed in the Business Layer. The Agents layer, in turn, is composed of two agents (*NotifierAgent* and *RecommenderAgent*), which were implemented to provide functionalities needed for the *User Notifications* and *Recommendation* features.

Table 3. FLOCOS GPL Features/Implementation Classes Mapping (partial).

Feature	Classes
User registration	`UserAction; UserService; UserServiceImpl; UserDAO; UserDAOMySQL`
FLO registration	`FLOAction; FLOService; FLOServiceImpl; FLODAO; FLODAOMySQL`
Discussion	`DiscussionAction; DiscussionService; DiscussionServiceImpl`
Experience Reports	`ExpReportAction; ExpReportService; ExpReportServiceImpl; ExpReportDAO; ExpReportDAOMySQL`
…	…

Finally, the last activity to be performed in this step is to provide traceability links between the FLOCOS GPL features and the implementation elements. Each feature is implemented by a set of classes. As a consequence, we built a table that maps each feature onto the classes that implements it. Table 3 shows a partial view of this mapping between features and their implementation classes. As a result, when one wants to derive a product that contains a certain feature, the related classes must be selected to be part of the product.

5 Discussion

In this section, we present and discuss some lessons learned and challenges that we have identified with our experience with the development of GPLs.

Providing mass customization. The use of the SPL practices and techniques for developing groupware enabled its mass customization, and in the specific case of FLOCOS GPL, in a semi-automatic way. The configuration of a specific product of FLOCOS GPL is made through the edition of a Spring configuration file (XML file) and the selection of classes and other implementation artefacts related to the selected features for the product. The Spring framework offers a model to build applications as a collection of simple components (called beans) that can be connected or customized using dependency injection. Spring container uses the XML configuration file to specify the dependency injection on application components.

Furthermore, there are some model-based tools, such as GenArch [25] and pure::variants, which automate the derivation process, reducing even more the time and effort needed to generate new products.

It is important to notice that a GPL is different from adaptable systems in the sense that its purpose is to accelerate groupware development enabling the product derivation according to user's needs. A common architecture allows to deriving different system versions. On the other hand, adaptable systems are single systems, which can be configured. One direction of future work is to explore how GLP architectures can benefit building adaptable systems.

Easier introduction of new features. Software systems typically evolve by the adding of new features that provide new functionalities to the users or change/improve the existing ones. The adoption of an architecture that has modularized features and low coupling among components helped on the introduction of new features to FLOCOS GPL. The techniques used in SPL engineering, and now incorporated to GPLs, make a system better maintainable as stated in [26]: "The same design techniques that lead to good reuse also lead to extensibility and maintainability over time". In addition, one way of developing Groupware is by a prototyping approach. It is based on the experience with new features, receiving users' feedback and adapting collaborative systems to meet user needs. As a consequence, we believe that SPL techniques, which improve feature modularization and therefore allow (un)plugging features, make sense for prototyping Groupware.

Use of software agents to tasks automation. Several systems have evolved with the addition of functionalities or features that automate tasks that require human intervention. Examples are monitoring the system to check user interaction and reducing searching time by the use recommendation techniques, which can take into account user preferences. Due to the pro-active and autonomous nature of software agents, they are an appropriate technology to implement this kind of feature. This approach was adopted in FLOCOS GPL to notify users about the posting of messages in forums and experience reports and to FLOs recommendation. Moreover, using the Web-MAS architectural pattern brought flexibility in the adding of these features with a low impact.

6 Conclusion

Groupware software development has to deal with common technical issues, such as communication and security. In addition, different groupware applications typically share a lot of functionalities and this fact can be exploited by the development of reusable assets that can be used to build customized applications. Several approaches have been proposed in order to address particularities of this kind of applications. However, they failed to provide complete solutions to the groupware development.

In this paper, we proposed an approach that addresses Groupware Product Lines (GPLs) and aims at solving identified issues in the groupware development. Our approach advances the groupware development by incorporating Software Product Line (SPL) techniques into the GPL development process, in order to build not single applications, but families of groupware. The groupware family is first analyzed in terms of features based on the 3C Collaboration Model to capture the commonalities and variabilities in the domain, while identifying the features' purpose according to the model. This information is documented in our extended 3C-Feature Model. Variabilities may have different degrees of granularity, such as a whole functionality (e.g. Discussion) or a functionality customization (e.g. Recommendation). Later, the family is designed and implemented taking into account the identified variability. In addition, features are mapped onto the implementation elements in order to allow an effective product derivation process. Our approach is illustrated with the FLOCOS GPL, which allows to systematically deriving customized versions of Learning Object repositories. We also discussed some lessons learned with our experience in the GPL development.

This paper addresses ongoing research on the GPL development. We are currently investigating how an existing component-based approach for developing groupware [6, 7], also based on the 3C Collaboration Model, can be used in a complementary way with our approach. In addition, we aim at adopting product derivation tools in order to automate the GPL derivation process.

Acknowledgments

This research is partially supported by project "Desafios" from CNPq. Bruno Gadelha, Ingrid Nunes, Hugo Fuks and Carlos José Pereira de Lucena receive grants from CNPq. Carlos José Pereira de Lucena and Hugo Fuks also receive grants from FAPERJ project "Cientistas do Nosso Estado."

References

1. Tietze, D.A. 2001. A Framework For Developing Component-Based Co-Operative Applications. Ph.D. Dissertation, Technischen Universität Darmstadt, Germany.
2. Greenberg, Saul. Multimedia Tools and Applications. Volume 32 , Issue 2. February 2007. ISBN 1380-7501, pp. 139 – 159.

3. Clements P. and Northrop L. Software Product Lines: Practices and Patterns. Addison-Wesley, Boston, MA, USA, 2002.
4. Pohl K., Böckle G., and van der Linden F. J.. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag, New York,USA, 2005.
5. Czarnecki K. and Helsen S.. Feature-based survey of model transformation approaches. IBM Systems Journal, 45(3):621–645, 2006.
6. Gerosa, M.A., Pimentel, M., Fuks, H. & Lucena, C.J.P. Development of Groupware based on the 3C Collaboration Model and Component Technology. CRIWG 2006, pp. 302-309.
7. Gerosa, M.A., Raposo, A., Fuks, H. & Lucena, C.J.P. Component-Based Groupware Development Based on the 3C Collaboration Model. SBES 2006, Brazil, pp. 129-144.
8. Szyperski, C. (2003) Component technology – what, where, and how? Procedings of the 25th International Conference on Software Engineering (ICSE'03), IEEE, pp 684- 693.
9. Ellis, C.A., Gibbs, S.J. & Rein, G.L. (1991): Groupware - Some Issues and Experiences. Communications of the ACM, Vol. 34, No. 1, pp. 38-58.
10. Fuks, H., Raposo, A.B., Gerosa, M.A. & Lucena, C.J.P. (2005), "Applying the 3C Model to Groupware Development", IJCIS, v.14, n.2-3, 2005, World Scientific, p.299-328.
11. Won, M., Stiemerling, O. & Wulf, V. (2005) "Component-Based Approaches to Tailorable Systems", End User Development, Kluwer, pp. 1-27.
12. Slagter, R.J. & Biemans, M.C.M. (2000) "Component Groupware: A Basis for Tailorable Solutions that Can Evolve with the Supported Task", in ISA 2000, Wollongong, Australia.
13. Roseman, M. & Greenberg, S. (1996) "Building real time groupware with GroupKit, a groupware toolkit". ACM Transactions on Computer-Human Interaction, 3, 1, p. 66-106.
14. Roth, J. & Unger, C. (2000) Developing synchronous collaborative applications with TeamComponents. In Designing Cooperative Systems: the Use of Theories and Models, COOP'00, pp. 353-368.
15. Fuks, H., Raposo, A., Gerosa, M.A., Pimentel, M. & Lucena, C.J.P. The 3C Collaboration Model. The Encyclopedia of E-Collaboration, Ned Kock (org), pp. 637-644, 2007.
16. Kang, K.; Cohen, S.; Hess, J.; Novak, W.; Peterson.: Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-021, SEI, Carnegie-Mellon University (1990)
17. Gomaa, H.: Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison Wesley (2004)
18. V. Alves. Implementing Software Product Line Adoption Strategies. PhD thesis, UFPE, Brazil, 2007.
19. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, November, 2002.
20. Gomes, S.; Gadelha, B.; Mendonça, A. P.; Amoretti, M. Marc, S.. Functional Learning Objects and Actual Metadata Limitations. Proceedings of XVI SBIE – Brasilian Symposium on Informatics on Education. Juiz de Fora- MG, Brazil, 2005.
21. SCORM 2004 2nd Edition Overview. Advanced Distributed Learning. July, 2004.
22. Gomes, S.; Gadelha, B.; Mendonça, A. P.; Castro Jr, A. N.. A Functional Learning Object Metadata Proposal. Proceedings of XVIII SBIE – Brasilian Symposium on Informatics on Education. São Paulo - SP, 2007.
23. Fuks, H., Raposo, A., Gerosa, M.A., Pimentel, M., Filippo, D., & Lucena, C.J.P. (2007) "Inter- e Intra-relações entre Comunicação, Coordenação e Cooperação". Proceedings of IV Collaborative Systems Brasilian Symposium, Brazil, pp. 57-68.
24. Nunes, I. O.; Kulesza, U.; Nunes, C.; Cirilo E.; Lucena, C. Extending Web-Based Applications to Incorporate Autonomous Behaviour. In: WebMedia, 2008, Vila Velha.
25. Cirilo E., Kulesza U., and Lucena C. A Product Derivation Tool Based on Model-Driven Techniques and Annotations. JUCS, 14:1344-1367, 2008.
26. Coplien J., Hoffmann D., and Weiss D.; "Commonality and Variability in Software Engineering", IEEE Software, vol. 15, no. 6, 1998, pp. 37–45.