REQUIREMENTS ENGINEERING OF WEB APPLICATION PRODUCT LINES

Hernán Casalánguida and Juan Eduardo Durán
Facultad de Matemática, Astronomía y Física, Universidad Nacional de Córdoba, Medina Allende s/n, Córdoba, Argentina

Keywords: Web engineering, Requirements engineering, Product line engineering.

Abstract: Application families (AF) are usually developed to reduce time to market and development costs of applications. Therefore, it is attractive to investigate the development of web AF (WAF) and to have an adequate requirements engineering (RE) method for WAF. A problem little studied is how to classify use cases (UC) for rich internet applications (RIA); to give better guidance to the developer we define a taxonomy for RIA UCs. UCs are described using UML activity diagrams (AD) in some web methods, but the taxonomies for actions proposed in them can be improved, because there are not enough action classes or they are too monolithic; for this reason we define a new action classification for RIA ADs. Studying the AD variability notations, we found a set of requirements for them; we define a notation for ADs satisfying these requirements and fulfilling some of them in a better way than in the literature. Non-functional requirements (NFR) for WAF must be documented. There are some goal-based approaches with variabilities; but they do not consider the modeling of NFRs and only consider examples for hard goals. We extend the NFR framework for the description of NFRs adding to it modeling elements for expressing variabilities.

1 INTRODUCTION

In general, AFs when well implemented help enormously reduce time to market and decrease development costs of applications. For several business areas there are many web applications that are similar, in terms of the functionality they offer; in addition, a software company could develop several web applications for a specific business area. Therefore it is interesting to investigate the development of WAF. As a first step we consider it is important to have an adequate RE process for WAF. To the best of our knowledge the field of RIA RE has not been properly exploited and we did not find a paper about RE of RIA families; for this reason these are topics of study in this work.

A problem little studied is how to classify UC for RIAs; only WebRE+ (Luna et al., 2010) considers a stereotype <<RIA specification>> for RIA behaviors; and we think that more than one stereotype should be used for different kinds of RIA behavior. We have decided to base UC variability modeling on (Bragança, 2007) and to define a UML profile in order to have a taxonomy for RIA UCs.

The presence of asynchronous events and asynchronous requests in a RIA UC introduces concurrent behaviors in UCs which can lead to too much scenarios and describing all of them can be costly; for this reason we think it is more convenient to describe a RIA UC using a notation with constructs for concurrency. UML AD notation has modeling elements for concurrency and was used in some web approaches that consider RE: UWE (Koch et al., 2008), OOWS (Valderas, 2004), WebRE+ (Luna et al., 2010). The taxonomies for actions proposed in these approaches can be improved, because there are not enough action classes or they are too monolithic.

There exist some approaches for defining AD with variabilities (Riebisch et al., 2000, Robak et al., 2002, Reuys et al., 2006, Bragança, 2007, Korherr and List, 2007; Schnieders and Weske, 2007, Razavian and Khosravi, 2008, Heuer et al., 2010), studying them in depth we found the following requirements for AD notations with variabilities:

R1: Describe variation points inside an AD.

R2: Avoid the use of stereotypes for variant elements.

R3: Use cardinality or interval notation for defining variation points in AD.

R4: Avoid the use of AD elements to represent variabilities, and do not clutter AD modeling elements with variability information.

R5: Provide well-defined and simple specialization semantics for ADs with variabilities.

R6: Describe dependencies between variants inside the AD.

R7: Consider data flow variability.

Another problem is the appropriate documenta-tion of NFRs for WAF. Feature models (FM) - see (Schobbens et al., 2006) - do not take into account several aspects in the area of NFR like: the impact of non-functional qualities on the functional part of a system, contributions of softgoals to some other softgoals, correlations between quality goals, argu-mentations, goal prioritization, evaluation pro-cedures. The use of aspect-oriented FMs (Kulesza et al., 2005) improves the situation, because it allows expressing how a feature for a non-functional quality affects other features; but the other aspects men-tioned are not contemplated. Goal-based approaches with variabilities recently have arisen (Semmak et al., 2008; António et al., 2009); but they do not deal with NFRs modeling and only consider case studies for hard goals.

The objectives of this paper are: to define an AD notation for capturing variabilities considering (possibly in a better way) the above requirements; to define a more detailed taxonomy of actions to give better guidance to the developer; and the definition of a NFR notation with variabilities that considers the facilities in the field of NFR.

We define in Subsect. 2.1 a profile that extends the UC model for RIAs and in Subsect. 2.2 an action taxonomy to be used for RIA UCs description. In Subsect. 2.2 we present an AD with variabilities notation fulfilling requirements R1 to R7 (some of them in a better way than in the literature – see Sect. 4). In Sect. 3 we extend the NFR framework (Chung et al., 2000) adding to it some modeling elements for variability expression.

2 FUNCTIONAL REQUIREMENTS

2.1 A Use Case Model for WAF

To maintain the original UML 2.0 UCMs, we only use from (Bragança, 2007) the additions to UML 2.0 UCMs to model variabilities. A *variability annota-tion* is represented as a note to be linked to *include* or *extend* relationships and represents a variability point with a name, a minimum and a maximum cardinality and the respective options. The family UC metamodel presents two new elements used to annotate variability: *Extend-Variability* and *Include-Variability* (see Fig. 1).

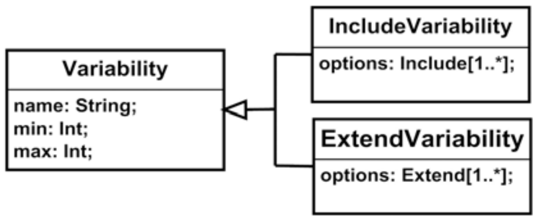

Figure 1: Variability Notation for UC Models.

In addition to use well known UC classes for Web applications it is necessary to find other classes of UC that contemplate typical RIA features like (some of them found in (Wright and Dietrich 2008)): run-time interface update, auto-suggest, real-time form validation, hover detail for items of information.

A *job* is an execution of one or more operations (i.e. input validation, information modification, or a calculation) by the system without user intervention. A *Job support step* (JSS) consists of the collection of inputs, the use of them for the execution of a job, and the possible presentation to the user of the result of the job. A *content visualization step* (CVS) is a presentation of content (possibly as a consequence to user inputs and/or search of information).

We consider the following four classes of UC:

1. «**Task**»: A *task* consists of a sequence of JSS. A «task» UC is a UC whose instances are tasks (i.e. each UC execution performs a task).

2. «**Navigation**»: A *navigation* consists of a se-quence of CVS, possibly decided by a user. A «navigation» UC is a UC whose instances are navigations performed by a user (i.e. each UC execution consists of a navigation).

3. «**RIA Navigation**»: A RIA navigation involves one or more CVS taking into account specific RIA interface features or the execution of asyn-chronous search (i.e. a search request is made and the system continues with its execution in-stead of waiting for the end of the search to continue); in addition some content visualize-tion steps can be concurrently performed.

4. «**RIA Task**»: A *RIA task* involves one or more JSS that take into account specific RIA interface features or the execution of asynchronous jobs (i.e. a job execution request is made and the sys-tem continues with its execution instead of wait-ing for the end of the job to continue); the JSSs can be

concurrently performed. A «RIA task» UC is a UC whose instances are RIA tasks.

The first two UC stereotypes are taken from (Casalánguida and Durán, 2009).

Fig. 2 shows a UCM for an AF of libraries. There is an includeVariability element saying that the «task» UC *reserveItem* is optional. There is an extendVariability element expressing that the developer has to choose one between *registerLoan with credits* and *registerLoan with payment*. The UC *findBooks* is of stereotype «RIA navigation» because we decided it will consider autosuggestion and live search. The UC *loanItem* is of stereotype «RIA task» because it uses real time form validation.

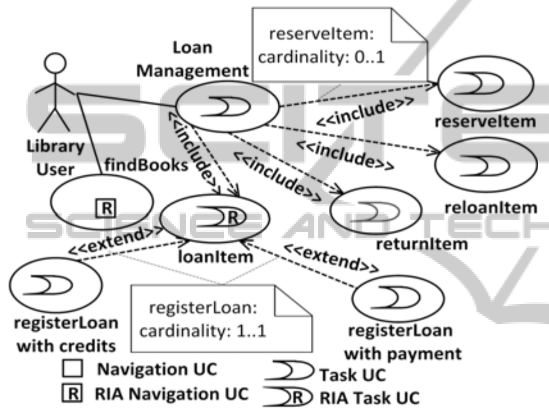

Figure 2: UC Model for a Family of Online Libraries.

2.2 Describing Use Cases

We consider the following stereotypes for actions:

- **«Search» Actions:** they represent database queries (i.e. relational, object-oriented, XML) or information retrieval. They could have an input pin for the parameters needed to make the query and an output pin for the result of the query.
- **«Job» Actions:** they are call behavior actions, whose activity performs a job. They could have input pins for the parameters of the job and an output pin for the result of the job.
- **«Input» Actions:** they represent the provision of an input by an human actor. They could have as output pins the values provided by the user.
- **«Output Request» Actions:** they represent the request by the system for the provision by a human actor of some inputs.
- **«Output Content» Actions:** they represent the system displaying content (for instance, the result of a job, the result of queries). They could have as input pins the values to show.

If an action needs to be executed as a transaction, the stereotype «*transaction*» must be used. For the execution of «transaction» actions, the ACID properties are valid.

To consider *control flow variability*, we need to be able to express that a sequence of actions inside an AD is optional or variant; for this purpose we use UML *activity groups*. An *ActivityGroup* is an abstract class for defining sets of nodes and edges in an activity. A *VariabilityActivityGroup* (VAG) is an ActivityGroup where grouped nodes and edges represent the execution of a sequence of actions inside an activity; in addition, there exists at most one node in the VAG (which is not a connector node) that is the destination of an activity edge which originated outside the VAG. A VAG is denoted by a dotted line rectangle with a name in the upper part and the grouped nodes and edges inside.

An *ActivityEdge* can be optional or variant with the meaning that the action pointed by the Activity-Edge will be chosen if the ActivityEdge is chosen.

For *data flow variability* we need to be able to express that a data item is optional or variant; for this purpose, we use from UML 2.0 *ObjectNode* and its specializations *InputPin* and *OutputPin*.

We propose the use of variability comments that are assigned to activity edges, to VAGs, and to object nodes. For this purpose, we add a metaclass called *Variability* which is described in Fig. 3. A *control flow variability* is a variability that takes into account activity edges (not leading to final nodes) and VAGs and a *data flow variability* is a variability that considers object nodes. A control flow vari-ability with interval *a..b* (min to max), optionsAE *AE* and optionsAG *AG* has the meaning: choose *n* arcs in *AE* and *m* activity groups in *AG* where $a \leq n + m \leq b$.

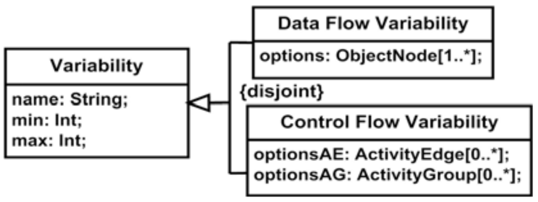

Figure 3: Variability Metaclass.

In a variability element there is at least one option to be chosen.

If a variability involves more than one variant, then such variants must be preceded by a fork node.

If a variability element has as variants all the nodes/VAGs following a fork node, we use an interval near the fork node and not a variability annotation.

Elements ActivityEdge, VAG and objectNode not included in variability objects are mandatory.

Fig. 4 shows an AD for a part of a *purchase* UC of an electronic commerce AF; this AD contains the optional VAG called *add discount*; inside this VAG there is a variability saying that exactly one of the actions below the fork must be chosen.

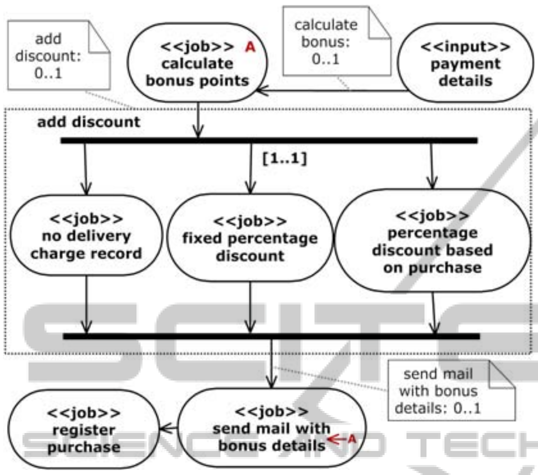
add discount: 0..1
<<job>> A calculate bonus points
calculate bonus: 0..1
<<input>> payment details
add discount
[1..1]
<<job>> no delivery charge record
<<job>> fixed percentage discount
<<job>> percentage discount based on purchase
send mail with bonus details: 0..1
<<job>> register purchase
<<job>> send mail with bonus details A

Figure 4: AD for part of Purchase UC.

Fig. 5 shows an example of data flow variability for the *Payment* UC.

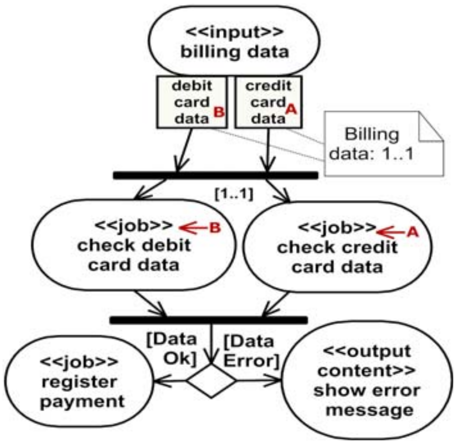

Figure 5: Data flow variability in Payment UC.

Another kind of variability not explained in variability approaches for AD is *business rule variability*: sometimes for a job to be executed a business rule must be valid; such a business rule can be variable among different WAF members. This fact is expressed with a decision node whose branches point to the same action and a variability with cardinality [1..1] is related to all the branches.

Fig. 6 shows an example of business rule variability in *addMember* (to a library) UC. There is a decision node with all conditions leading to the *Add member* action, and there is a variability saying that only one of these conditions must be chosen.

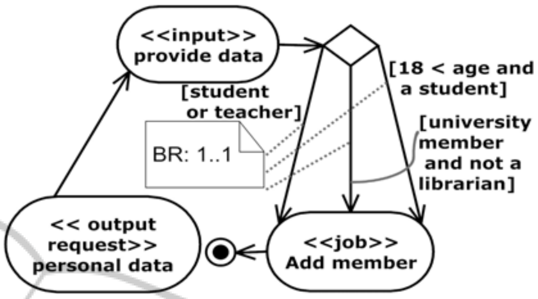

Figure 6: Business Rules in addMember UC.

In Fig. 7 metaclasses for dependency constraints (DC) are presented. A *variant* consists of a variability name and of either an objectNode or an ActivityEdge or a VAG or a UC. A DC can be of type *excludes* (denoted with a left-right arrow) or *requires* (denoted with a left arrow) and has a *name*. We have two kinds of DCs:

- *DC involving variants of an AD*: Such DCs have a *dependent,* which can be a control flow vari-ant or an ObjectNode. The dependent depends on a non-empty set of variants inside the AD (obtained by navigating the roles *in* and *depend-ence on* of the corresponding metaassociations)
- *DC involving an AD variant and a UC variant*: Such DC has a *dependent* that is a control flow variant. The dependent depends on a UC variant (obtained through navigating the roles *in* and *dependence on* of the corresponding meta-asociations). For instance: an AD for the UC *loanBook* contains an optional action called *Check for Reserved Item* and there is a DC of type *requires* between the action *Check for Reserve Item* and the UC *reserveItem*.

In Fig. 4 there is an example of a DC called *A* of type *requires* between *calculate bonus points* action and *send mail with bonus details* action.

Fig. 5 provides an example of two DC of type *requires*: one called *B* between outputPin *debit card data* and action *check debit card data*, and the other called *A* between outputPin *credit card data* and action *check credit card data*.

Now we describe a specialization algorithm for ADs. Assume that decisions of variants according to compile time variabilities have been taken and DCs are respected; the specialization of an AD is another AD obtained by applying the following rules:

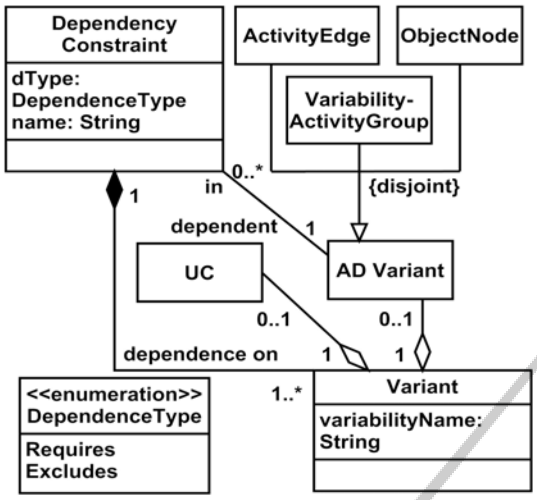

Figure 7: Metaclasses for DC Modeling.

1. If a variability involves only one variant, then the variant is deleted from the AD if this variant has not been chosen. Next, remove the variability.
2. If a control flow variability takes into account more than one variant, then each variant that has not been chosen is deleted from the AD. After that, delete nodes fork and join of the variability if only one activity edge points to the join node of the variability. Next, remove the variability.
3. If a data flow variability considers more than one variant, then each variant that has not been chosen is deleted from the AD. Next, remove the variability.
4. For a business rule variability each activityEdge variant that has not been chosen is deleted from the AD. Next, remove the variability.

By space reasons, it is not explained in detail how to delete a variant. The idea is to remove the variant element in case of being a VAG or an object node, and to remove both the variant element and the action pointed by it in case of being an ActivityEdge. In addition, it may be necessary to remove some arcs leading to or going out of the variant element in case of being a VAG or an object node). In several cases it is necessary to replace the elements deleted by an activityEdge.

In case of nesting of some control flow variabilities, rule 3 is systematically applied from outer to inner levels of nesting.

Within a UC description actions with stereotype «include» are activity behavior actions that represent the execution of an included UC. The name of an «include» action is a name of an included UC, whose description is externally made to the including UC with an AD.

Each extension UC is described with an AD. For each extension UC, *extension rules* are defined which consist of the following parts:

- **Base UC:** name of the extended UC.
- **Condition:** condition that must hold for the insertion of the extension UC.
- **Extension Points:** this is a list of *extension point descriptions*, specifying the places in the UC's AD where the UC extension execution is inserted; a place can be before or after a specific action occurrence. An extension point descript-tion has the following syntax: *(after | before) (action name | extension point name)*. An action name is used if the action occurs only once in the base UC's AD; otherwise an extension point name is used.

Each extension point name is added as the value of a tagged value named *extension point name,* which is assigned to the arrow leading to the desired activity occurrence in the base UC's AD.

In an AF all the extension point names should be unique (i.e. the same extension point name cannot be used in different UCs or in different action occurrences within a base UC's AD).

3 NON-FUNCTIONAL REQUIREMENTS

The NFR framework (Chung et al., 2000) presents softgoals and their relationships inside *softgoal interdependency graphs* (SIG). The modeling elements of a SIG are softgoals - that can be: NFR softgoals, operatinalizing softgoals (OS), or claims - and contributions - that can be: AND-decomposition, OR-decomposition, degree of contribution - MAKE (++), HELP (+), HURT (-), BREAK (--).

To be able to describe SIGs of an AF that will be specialized to application SIGs respecting the NFR framework, we define a metamodel called NFRV (see Fig. 8) that extends the NFR framework to take into account variabilities.

A OS or a NFR softgoal may participate in one or more variabilities. A Variability element has a name and a cardinality given by two integers called *min* and *max*.

We define the following variability constraints: A NFR-softgoal or OS cannot have two variabilities with the same name. If two softgoals (NFR or OS) have a common variability, then they are offspring of the same father.

It is not possible to define a variability for a subset of softgoals of an AND/OR decomposition, because all possible softgoals of an AND or OR decomposition must be present on any family member to satisfy the decomposition; if we allow to choose a proper subset of the decomposition's softgoals for a product line member, then the semantics of AND/OR decompositions is not respected. For this reason, we define new kinds of decompositions for supporting variabilities:

- *ANDVariability*: One or more variabilities comprise the components of the decomposition and all the softgoals chosen according to such variabilities are necessary to achieve the father of the decomposition.
- *ORVariability*: One or more variabilities comprise the components of the decomposition and at least one of the softgoals chosen according to such variabilities is necessary to achieve the father of the decomposition.

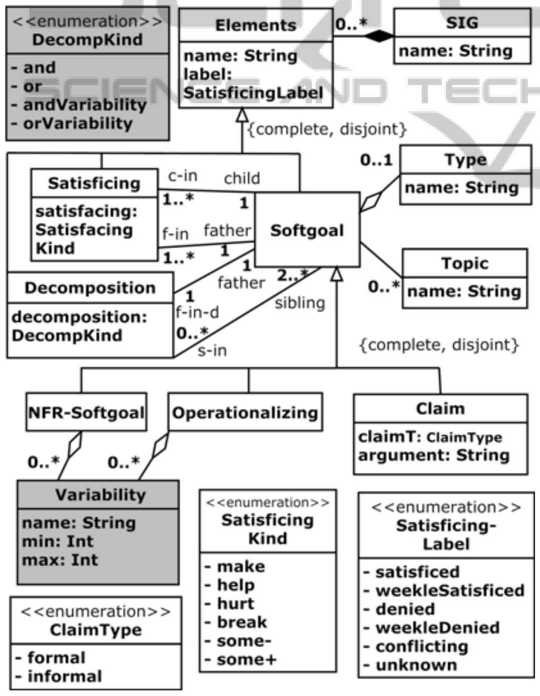

Figure 8: NFRV Metamodel.

A variability is represented with a UML annotation for variant softgoals. An ANDVariability element is represented in the same way as an AND decomposition. An OrVariability element is represented in the same way as an OR decomposition. If in a variability *V* with cardinality [0..1] participates only one softgoal that participates only in *V*, then this softgoal is called *optional* and is represented putting a question mark inside the softgoal cloud. If there is one variability with cardinality a..b comprising all the members of an ANDVariability or ORVariability, then omit the variability annotation and instead put "[a..b]" near the decomposition.

The graph formed by a softgoal *S* and its descendants (contributions and softgoals) in the SIG is denoted by *DG(S)*. The set obtained by removing from *DG(S)* all the *DG(T)* such that T is a descendant of *S* and *T* contributes to a softgoal outside of *DG(S)* is denoted by *DGO(S)*. Assume that decisions of variants according to variabilities have been taken. The specialization of a SIG NFRV is a SIG obtained by applying the following rules:

- If a softgoal *S* is optional, and *S* has not been chosen, then the relations of *S* with its parents and *DGO(S)* are deleted.
- If an ANDVariability with father *F* contains a variability *V*, then remove *V* and for each vari-ant *M* of *V* that has not been chosen delete the relation between *F* and *M* and *DGO(M)*. The selected variants are offspring of *F* and are related to it via an AND decomposition.
- If an ORVariability with father *F* contains a variability *V*, then remove *V* and for each vari-ant *M* of *V* that has not been chosen delete the relation between *F* and *M* and *DGO(M)*. The selected variants are offspring of *F* and are related to it via an OR decomposition.
- If *V* is a variability with its variants contributing in some degree to a father softgoal *F*, then remove *V* and for each variant *M* of *V* that has not been chosen delete the relation between *F* and *M* and *DGO(M)*.

Fig. 9 Shows a SIG NFRV for *usability*. This softgoal is decomposed using an AND-Variability into *Errors*, *Efficiency* and *Learnability* using a variability of cardinality 2..3 (i.e. at least two of the three softgoals must be chosen in specialization time). The OS *Show related data* and *Use site maps* are optional, and can be selected or not when *Efficiency* is chosen. The mandatory OS *Use help* is decomposed using an ORVariability into OS *Help general index*, *Help tips* and *Help Agent*. *Help tips* and *Help agent* is a group of variant softgoals with cardinality 1..1 (i.e. one of them will be chosen in specialization time). The optional OS *Suggest fields* also contribute to *Use help*.

4 RELATED WORK

WebRE+ (Luna et al. 2010) uses the stereotypes «Search» (for querying contents), «Browse» (for navigation), «UIAction» (for actions performed by the user in a «UIElement») and «userTransaction» (for actions involving a transaction operation). In contrast with our approach, WebRE+ activities performed by the user are not separated from activities performed by the system (for instance «Search» and «UserTransaction» activities need the involvement of both the user and of the system); in addition a «userTransaction» activity may be very complex and could be decomposed into smaller ones.

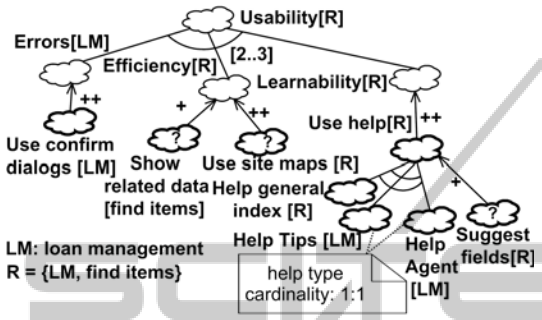

Figure 9: SIG with Variabilities for Usability.

In OOWS (Valderas 2004) the stereotypes «search» (for actions performing information search), «function» (for actions executing functionality), «output» (for output interaction points), and «input» (for input interaction points) are considered.

Table 1 shows the stereotypes for AD provided by our work and related approaches. Our approach permits to express WebRE+ stereotypes for content visualization steps (CVS) and «User Transaction» by using some distinct stereotyped actions.

Table 1: Stereotypes for ADs in Web Methods.

Objective	WebRE+	OOWS	Our work
CVS without search	Browse	Input, output	Input, Output content, Output request
CVS with search	Search	Input, Search, output	Input, Search, Output content, Output request
Activity with a transaction	User Trans-action	Input, Output, Function, Search.	Input, Output Content, Output request, Search, Job, Transaction
Input	UIAction	Input	Input
Input request	-	output	output request
Output content	-	output	Output content
transaction	-	-	Transaction

In WebRE+ (Luna et al. 2010) a UC can be of stereotypes «Navigation», «WebProcess» or «RIA Specification» (for a behavior involving RIA specific features). The first two stereotypes are covered in our work by the «Navigation» and «Task» stereotypes and the third is contemplated by our «RIA navigation» and «RIA task» stereotypes.

Table 2 provides a comparison of AD notations for variability management w.r.t. the requirements from R3 to R7 (presented in the introduction). Below we justify the results in detail.

Table 2: comparison of AD with variabilities notations.

	R3	R4	R5	R6	R7
Bragança.	no	no	no	no	no
Heuer et al.	good	reg+	good	good	no
Razavian, Khosravi	reg	no	no	no	good
Korherr and List	no	no	reg	no	no
Schnieders and Weske	no	no	no	no	good
Robak et al.	no	no	no	no	no
Our work	very good	very good	very good	very good	very good

R3: full use of cardinalities for variabilities: only in (Heuer et al., 2010) and our work. In contrast to our work in (Heuer et al., 2010) this information is captured by Boolean expressions outside the AD.

R4: do not use AD elements to represent variabilities: only in (Riebisch et al., 2000; Heuer et al., 2010) and our work. From these approaches only our work does not put variability information in AD elements.

R5: presentation of some specialization rules: in (Schnieders and Weske, 2007; Heuer et al., 2010) and our work. For an exhaustive specialization algorithm we only found (Heuer et al., 2010) and our work. Do not use other notations in the specialization algorithm: only in our work ((Heuer et al., 2010) uses Petri nets).

R6: use of DCs: in (Heuer et al., 2010) and our work. In contrast to our work (Heuer et al., 2010) is more restricted in the kind of DC that can be expressed (variants represent sets of actions) and DCs are defined outside the AD by means of Boolean expressions.

R7: Use of data flow variability: only in (Razavian and Khosravi, 2008; Schnieders and Weske, 2007) and our work. Use of cardinalities in data flow variabilities and do not use stereotypes in data flow variabilities: only in our work.

5 CONCLUSIONS

In this paper we presented a RE method for WAF considering both RIA and Web 1.0 applications.

UML tools can be used for drawing ADs and UCMs for WAFs, because our extensive use of variability annotations.

Our notation for AD variability management satisfies all the requirements stated in the introduction and permits to express both data flow and control flow variabilities.

We proposed the documentation of business rule variability in ADs; we did not find papers considering this for ADs. In addition, we considered a more complete treatment of DCs in ADs than in the literature. Furthermore, because of the use of annotations to model variabilities the documentation of data flow variabilities is simpler than in the approaches found in the literature.

Our method has been tested by considering parts of an electronic commerce WAF and of an online library WAF. Such examples are representative of data intensive Web 1.0 or RIA WAFs.

We defined the specialization algorithms taking preservation of syntactic correctness into account. In the future we plan to check for the AD specialization algorithm the preservation of behavioral correctness. In addition we plan to continue validating our RE process with other interesting RIA families.

REFERENCES

António, S., Araújo, J., Silva C., 2009. Adapting the i* Framework for Software Product Lines. In: *ER'09*. LNCS 5833, pp. 286-295, Springer-Verlag.

Bragança, A., 2007. Methodological Approaches and Techniques for Model Driven Development of Software Product Lines. PHD-Thesis, Universidade do Minho, Escola de Engenharia.

Casalánguida, H., Durán, J., 2009. Modelado Orientado a Aspectos de Navegación para Aplicaciones Web basado en UML. In IEEE Latin America Transactions, Vol. 7, N° 1, pp 92—100.

Chung, L., Nixon, B., Yu, E., Mylopoulos, J., 2000. Non functional Requirements in Software Engineering. Kluwer Academic Publisher, Boston.

Heuer, A., Budnik, Ch., Konrad, S., Lauenroth, K., Pohl, K., 2010. Formal Definition of Syntax and Semantics for Documenting Variability in Activity Diagrams. In *SPLC'10.* LNCS 6287, pp 62--76.

Koch, N., Knapp, A., Zhang, G. Baumeister, H., 2008. UML-Based Web Engineering. An Approach Based on Standards. In *Web Engineering: Modelling and Implementing Web Applications*, Human Computer Interaction Series, Springer, pp. 157--191.

Korherr, B., List, B., 2007. A UML 2 Profile for Variability Models and their Dependency to Business Processes. In *DEXA´07*, pp 829--834.

Kulesza, U. García, A., Bleasby, F., Lucena, C., 2005. Instantiating and Customizing Product Line Architectures using Aspects and Crosscutting Feature Models. In *EA'05, Workshop on Early Aspects*.

Luna, E., R., Escalona, M., J., Rossi, G., 2010. A Requirements Metamodel for Rich Internet Applications. In *ICSOFT 2010*.

Razavian, M., Khosravi, R., 2008. Modeling Variability in Business Process Models Using UML. In *5th Intl. Conf. on Inf. Technology: New Generations*, pp 82-87.

Riebisch, M., Böllert, K., Streitferdt, D., Franczyk, B., 2000. Extending the UML to Model System Families. In *IDPT 2000, Integrated Design and Process Technology*. Society for Design and Process Science.

Robak, S., Franczyk, B., Politowicz, K., 2002. Extending the UML for Modelling Variability for System Families. In *Intl. Journal of Appl. Math. Comput. Science*, Vol.12, No.2, pp 285–298.

Schnieders, A. and Weske, M., 2007. Activity Diagram Based Process Family Architectures for Enterprise Application Families. In: *Enterprise Interoperability 2007*, Part II, pp 67—76.

Schobbens, P., Heymans, P., Trigaux, J., 2006. Feature Diagrams: a Survey and a Formal Semantics. In *RE'06*, pp 139—148.

Semmak, F., Gnaho, C., Laleau, R., 2008. Extended KAOS to Support Variability for Goal Oriented Requirements Reuse. In *MoDISE-EUS'08,* Vol. 341 of CEUR Workshop Proceedings, pp 22-33.

Valderas, P., 2004. A requirements engineering approach for the development of web applications. PHD-thesis, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia.

Wright, J., Dietrich, J., 2008. Requirements for Rich Internet Application Design Methodologies. In *WISE´08*, LNCS 5175, Springer, pp 106—119.