Using CVL to Operationalize Product Line Development with Reusable Aspect Models

Benoit Combemale, Olivier Barais, Omar Alam, Jörg Kienzle

Using CVL to Operationalize Product Line Development with Reusable Aspect Models

Benoit Combemale and Olivier Barais
University of Rennes 1, IRISA
Rennes, France
{benoit.combemale, barais}@irisa.fr

Omar Alam and Jörg Kienzle
McGill University
Montreal, QC, H3A 2A7, Canada
Omar.Alam@mail.mcgill.ca
Joerg.Kienzle@mcgill.ca

ABSTRACT

This paper proposes a software design modelling approach that uses the Common Variability Language (CVL) to specify and resolve the variability of a software design, and the aspect-oriented modelling technique Reusable Aspect Models (RAM) to specify and then compose the detailed structural and behavioural design models corresponding to the chosen variants. This makes it possible to 1) exploit the advanced modularization capabilities of RAM to specify a complex, detailed design concern and its variants by means of a set of interdependent aspect models; 2) use CVL to provide an easy-to-use product-line interface for the design concern; 3) automatically generate a detailed design model for a chosen variant using a custom generic CVL derivation operator and the RAM weaver.

1. INTRODUCTION

A well-established and convenient practice in variability management is to provide a specification of the variability in terms of features separately from the associated artifacts that provide an implementations of the actual reusable assets. While some de-facto standards such as feature diagrams [9] are widely used to represent commonality (i.e., common properties) and variability (i.e., differences) of a system in terms of features, they still have to rely on specific operators to modularly implement and then compose the reusable aspects.

In this paper, we propose an approach combining the *Common Variability Language* (CVL) [4] to specify and resolve the variability of a software design, and the aspect-oriented modelling technique *Reusable Aspect Models* (RAM) [10] to implement and compose reusable object-oriented software design aspects. We use RAM to describe and compose the assets, while the feature model and its resolution (which are currently not explicit in RAM) are made explicit using CVL.

The contribution of this paper is therefore twofold: On the one hand we show how CVL can be used to extend an existing approach for AOM with well-established practices coming from the variability management community. On the other hand we illustrate the use of CVL with specific modularity and composition operators tailored to work with an aspect-oriented modelling technique. The derivation operator of CVL is specialized to work with RAM, resulting in an implementation of a generic opaque variation point that can produce the composition directives allowing the RAM weaver to produce a woven model corresponding to the chosen configuration.

The remainder of the paper is structured as follows: section 2 presents an overview of the proposed approach; section 3 illustrates the details of the approach by means of a software design concern product line for workflow executions; section 4 presents related work and the last section draws some conclusions.

2. APPROACH OVERVIEW: COMBINING CVL AND RAM

The approach we propose is based on **RAM** and **CVL**.

Reusable Aspect Models (RAM) [10] is a modelling approach that allows a designer to specify models that define the structure and behaviour of recurring design solutions. RAM models are inherently reusable, which means that it is possible to customize the generic design solution models to application-specific needs when applying them in a software design of a specific application. Currently, the RAM tool comes with a growing library of reusable design models, including models for low-level utility concerns, design patterns, network communication, workflow definition and execution, and transactions.

In general, there is *no one single good way* to solve a specific design problem. This is why RAM supports the definition of families of interrelated design models – in RAM terminology called *concerns* – that describe different variations of how to address a design problem. Typically, at the core of such a design concern is at least one aspect model that encapsulates the structure and behaviour common to all variations. Additional structure and behavioural properties covering variations of the design are modelled within *extensions* to that core model.

The **Common Variability Language** (CVL)[1] [4] is a domain-independent language for specifying and resolving variability over any instance of any MOF-compliant metamodel. Inspired by feature model, CVL contains several layer. The Variability Abstraction Model (*VAM*) is in charge of expressing the variability in terms of a tree-based structure. The core concepts of the VAM are the variability

[1]CVL is currently a proposal submitted to OMG. Cf. `http://variabilitymodeling.org`.

specifications (*VSpecs*). The *VSpecs* are nodes of the VAM and can be divided into three kinds: Choices, Variables and Classifiers. The Choices are *VSpecs* that can be resolved to yes or no (through *ChoiceResolution*), *Variables* are *VSpecs* that requires a value for being resolved(VariableValue) and *Classifiers* are *VSpecs* that imply the creation of instances and then providing per-instance resolutions (*VInstances*). In this paper, we mainly use the *Choices VSpecs*, which can be intuitively compared to features, which can or cannot be selected during the product derivation (yes/no decision). Besides the VAM, CVL also contains a Variability Realization Model (VRM). This model provide a binding between the base model and the VAM. It makes possible to specify the changes in the base model implied by the *VSpec* resolutions. These changes are expressed as Variation Points in the VRM. The Variation Points capture the derivation semantics, i.e. the actions to perform during the Derivation. Finally, CVL model contains resolution models to fix the variability capture in the VAM.

In this paper, we propose an approach combining CVL to specify and resolve the variability, and RAM to implement and compose reusable object-oriented software aspects. We use RAM to describe and compose the assets while the feature model and its resolution (which are currently not explicit in RAM) are made explicit using CVL.

The global approach is a two-level process: first the reusable aspects are capitalized and their possible combinations are captured in a variability model. Second, the variability model is used to select an expected set of features (aka configuration) from which a woven model is produced by composition of the suitable reusable aspects.

In practice, as illustrated in Figure 1, the approach is divided into the following five steps:

① implementation of the reusable aspects using RAM;

② specification of the variability (called *variability abstract model*) using the choice diagram proposed by CVL;

③ resolution of the variability by selecting a set of features (called *resolution model*) using CVL;

④ derivation of the composition directives using the generic CVL derivation operator, with a dedicated opaque variation point that we propose (and include in the *variability realization model*);

⑤ composition of the corresponding reusable aspects as describe in the composition directives with the RAM weaver.

From a methodological perspective, we also distinguish two roles for users of our approach:

- **Design Concern Expert**. The design concern expert knows the domain captured in the reusable aspects and knows their possible combination. This person is thus in charge of leveraging this domain to model one or several reusable aspects with RAM (step ① in Figure 1), and the respective variability with CVL (step ② in Figure 1).

- **Application Engineer**. The application engineer create models in the application domain. These users, through their modelling activities, can select the expected features with CVL (step ③ in Figure 1), and then automatically derive the composition directives (step ④ in Figure 1) used by RAM to automatically generate the corresponding woven model (step ⑤ in Figure 1).

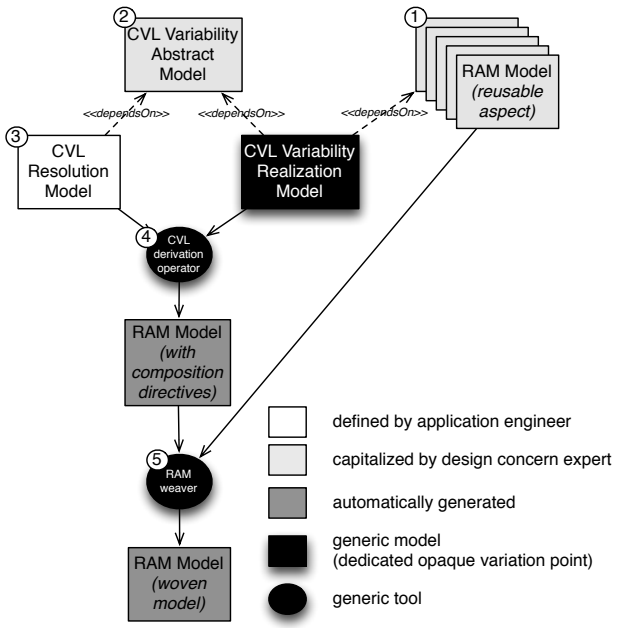

Figure 1: Approach Overview Combining CVL and RAM

3. APPROACH DETAILS: THE WORKFLOW CASE STUDY

A workflow is a set of operations that need to be completed in a certain order to fulfill a goal or task. For example, workflows have been used in software engineering to describe how a system under development is to interact with its environment.

At run-time, a system that is supposed to behave according to a workflow specification needs to incorporate a workflow execution engine in its design. To facilitate this, we have designed a reusable workflow design concern in RAM that provides such a functionality. Since workflows can be of varying sophistication, we designed a product line of workflow execution engines, which allows the designer to choose the most appropriate configuration for his specific application.

In this section of the paper we illustrate our process in detail by means of the workflow design concern case study. The following subsections correspond to the steps outlined in section 2.

3.1 Designing Reusable Assets using RAM

This subsection outlines the detailed design of the aspect models that are part of a RAM design concern. For space reasons, all presented models are simplified versions of the real RAM workflow design concern models: only the classes relevant to the definition of the different kind of nodes found in a workflow are shown. Structure related to the execution of the workflow, as well as all the sequence diagrams describing the behaviour of the design have been omitted. The in-

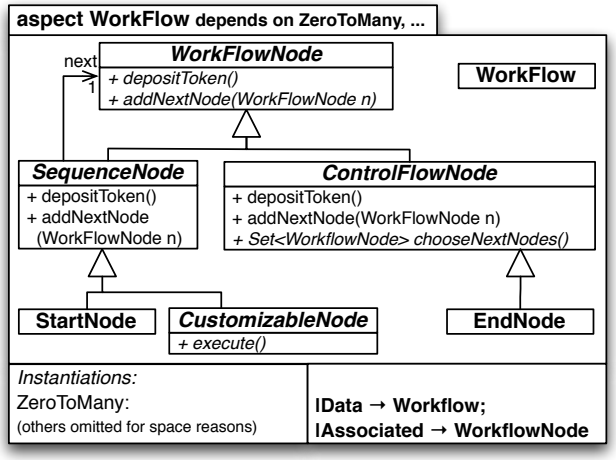

Figure 2: The Base Workflow Aspect Model

terested reader can download the complete models from our website [2].

3.1.1 The Core Workflow Aspect Model

Figure 2 shows the *WorkFlow* aspect which defines the minimal model elements found in every workflow. It states that a basic workflow is composed of nodes, which can be sequence or control flow nodes. A special sequence node is the `StartNode`, a special control flow node is the `EndNode`. The `WorkFlowNode` has two abstract methods, `depositToken()` and `addNextNode(WorkFlowNode n)`, which are implemented differently by the two subclasses. This allows other parts of the system to treat workflow nodes in a uniform way. For example, the work flow execution engine (not shown for space reasons) can deposit a token into any kind of node in order to execute it, whether it is a `SequenceNode` or a `ControlFlowNode`. The design that encodes that a `Workflow` is composed of zero or more `WorkflowNode`s is implemented by another reusable aspect model, *ZeroToMany*. The instantiation directive on the bottom of Figure 2 instantiations that tell the RAM weaver how to compose *ZeroToMany* with *Workflow*.

With this base workflow aspect, a user can build very simple, sequential workflows. Application-specific actions are to be designed by extending `|CustomizableNode`, a class that executes the `execute` method when a token is deposited before scheduling the next node.

3.1.2 Workflow Extensions

The RAM workflow concern provides additional aspect models that define more elaborate control flows. For instance, there are control flow nodes that have multiple successor nodes, where each outgoing path is named using a string. Figure 3 shows an aspect called *OutPath* that extends the *WorkFlow* aspect and defines the structure needed for control flow nodes with more than one named outgoing path in the class `|CFNWithOutPath`. A new kind of named sequence node is introduced, `OutpathNode`, and a new kind of control flow node, `|CFNWithOutpath`. Internally, the *Map* aspect is reused to define a hash table that maps strings to `OutpathNode` as shown by the instantiation directives at the bottom of the figure.

[2] `http://www.irisa.fr/triskell/perso_pro/obarais/pmwiki.php?n=App.VARY2012`

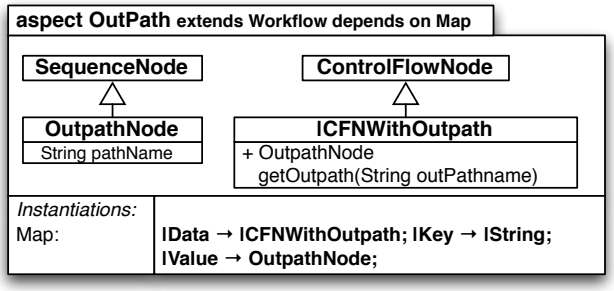

Figure 3: The Outpath Aspect Model

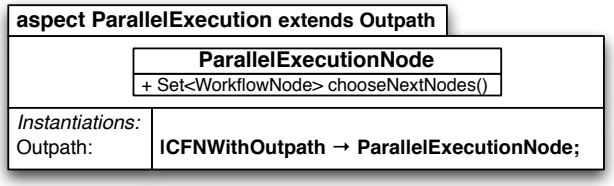

Figure 4: The ParallelExecution Aspect Model

The *ParallelExecution* aspect shown in Figure 4 is an example of an aspect that uses the *Outpath* aspect to define a control flow node that allows a workflow to continue execution of several following nodes in parallel. To reuse *Outpath*, `ParallelExecutionNode` is composed with `|CFNWithOutPath`.

The RAM workflow design concern defines many other workflow extensions, which can unfortunately not be shown here for space reasons. They are:

- *ConditionalExecution*, which allows for selective execution of workflows;
- *Synchronization*, which allows concurrent workflows to wait for each other;
- *Conditional Synchronization*, which allows concurrent workflows to wait for each other conditionally;
- *Timed Synchronizaton* which allows concurrent workflows to wait for each other until a timer expires;
- *Input*, which allows workflows to wait for input messages coming from the network;
- *Output*, which allows workflows to send output messages to the network;
- *Nesting*, which makes hierarchies of workflows possible.

Each extension is designed in one aspect model that extends the base workflow model, and optionally depends on other models to provide lower-level functionality. The left hand side of Figure 5 shows an overview of all the aspect models of the workflow design concern and their dependencies (shown using black straight arrows). The aforementioned extensions are above the *Workflow* aspect, since they add structure and behaviour to the latter. At the bottom of the figure, below the *Workflow* aspect, are all the low-level design models implementing design patterns (e.g. *Singleton*), recurring data structures (e.g. *ZeroToMany*, *Stack* or *Map*), or utility aspects (e.g. *NetworkCommand*).

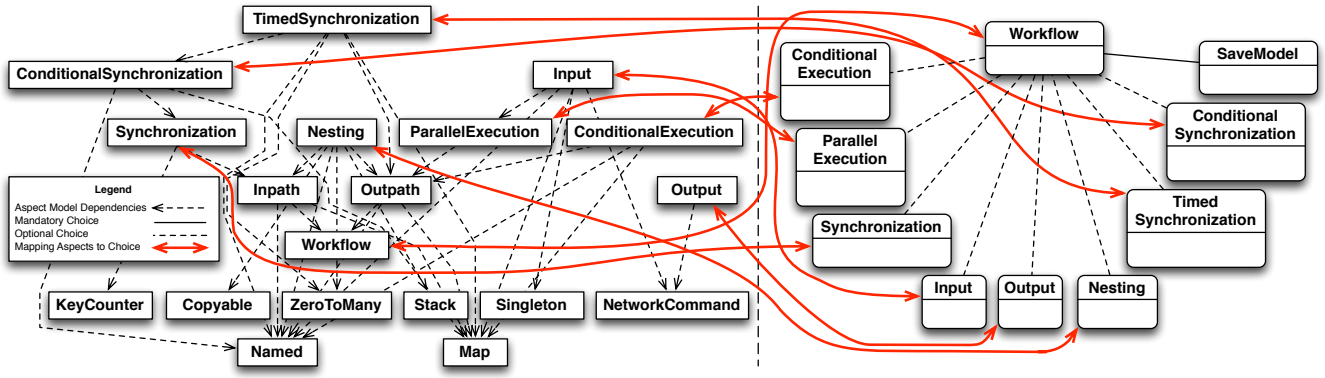

Figure 5: The Workflow Design Concern (RAM Models and Dependencies) and the CVL Variation Model

3.2 Specifying the Variability using CVL

To use a RAM design concern within an application model, the designer first needs to perform careful tradeoff analysis that takes into account the functional and non-functional requirements of the application under development to determine the desired concern variation. Then the designer needs to add instantiation directives into the application model that instantiate the RAM aspect models that correspond to the desired design concern variation. As a result, the RAM weaver can then compose the design concern models with the application model to yield the complete design model.

Unfortunately, this reuse process is quite cumbersome for the designer. In the current version of RAM, a design concern family does not have a well-defined user interface. The designer is confronted with a collection of interdependent aspect models as shown on the left hand side of Figure 5. Once the designer has determined which features of the design concern are relevant to her, she must determine which RAM models contain the design of these features, and then she must manually instantiate them.

To ease the task of the application developer, we propose to use the CVL Variation Model (VAM) to present a simple-to-use, feature-oriented view of a RAM design concern to the user. It encodes the set of choices and the constraints between choices. For the workflow example, we obtain the choice model as depicted on the right hand side of Figure 5. This choice model is quite simple, it contains a root choice $Workflow$ with eight optional sub-choices and one mandatory choice $saveModel$.

The mapping between the choice model and the RAM aspects, illustrated using red arrows in Figure 5, is designed in the CVL Variability Realization Model (VRM). The VRM contains ten *Opaque Variation Points* (OVP). An OVP is a black box variation point whose behaviour is defined with an action language expression specified in the CVL model. In our CVL implementation, we currently support OVPs defined in Groovy[3], in Javascript or in Kermeta [8]. With these action languages, the designer can modify the base model directly. Each variation point has access to a context that contains the list of objects to remove ($toRemove$), the list of *objectHandles* associated with this variation point (ctx), the list of variables and their associated value defined in the the resolution model ($args$), and a map of key/value pairs that variation points can use to pass data to subsequent variation points (map).

[3]http://groovy.codehaus.org/

Among the ten OVPs, there are only three different types. The first one is bound to the root choice and contains the action language expression to create a RAM aspect that defines the composition directives. The second one is bound to the $SaveModel$ choice and contains the action language expression to save the final model. The eight remaining OVPs are bound to RAM aspects and contains the code to create the composition directive in the root aspect previously created. The action language expressions for these three types of OVPs are shown in Listing 1.

Listing 1: Worflow OVPs in Groovy

```
1  //Expression for CreateAspectWorkflow OVP
2  Aspect aspectcreate=ca.mcgill.cs.sel.ram.
       RamFactory.eINSTANCE.createAspect();
3  aspectcreate.setName(args.get("name"));
4  maps.put("__NewAspect", aspectcreate);
5  ------------------------------------------
6  //Expression for the eight OVP associated to
       RAM apsects
7  Aspect newaspect = maps.get("__NewAspect");
8  Instantiation inst = ca.mcgill.cs.sel.ram.
       RamFactory.eINSTANCE.createInstantiation();
9  inst.setType(ca.mcgill.cs.sel.ram.
       InstantiationType.EXTENDS);
10 inst.setExternalAspect(ctx.get(0));
11 newaspect.getInstantiations().add(inst);
12 ------------------------------------------
13 //Expression for SaveModel OVP
14 Aspect aspecttoSave = maps.get("__NewAspect");
15 ResourceSet resourceSet = new ResourceSetImpl()
       ;
16 resourceSet.getResourceFactoryRegistry().
       getExtensionToFactoryMap().put(
17   "ram", new XMIResourceFactoryImpl());
18 URI fileURI = URI.createFileURI(new java.io.
       File(args.get("modelname").getAbsolutePath
       ());
19 Resource resource = resourceSet.createResource(
       fileURI);
20 resource.getContents().add(aspecttoSave);
21 resource.save(java.util.Collections.EMPTY_MAP);
```

3.3 Specifying the Resolution using CVL

With the CVL VAM model, the application engineer can do the selection according to the variability captured in the VAM model. For this point, we automatically generate an initial solution for the resolution model according to the choice model constraints (cardinalities, $isImpliedByParent$, $DefaultResolution$, ...). The application engineer can change this choice resolution decision using a graphical tool as shown in Figure 6. In this example, we take the decision to select

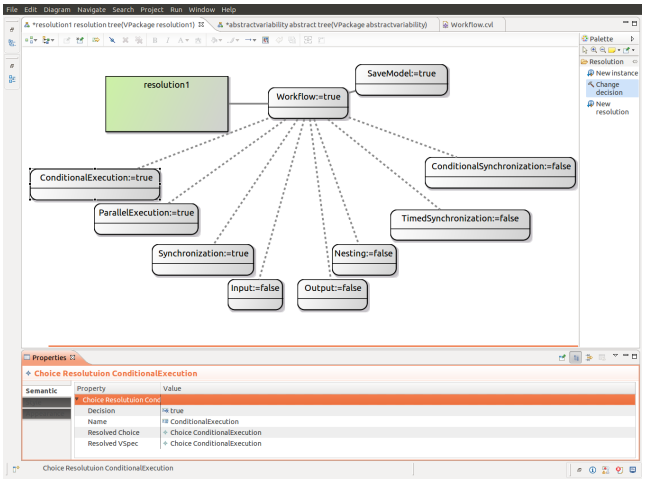

Figure 6: The Workflow Resolution model

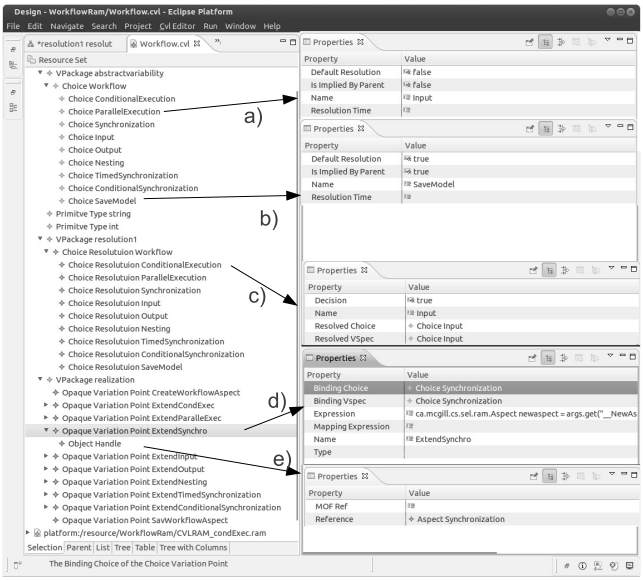

Figure 7: The Workflow CVL model

the *ParallelExecution*, *ConditionalExecution* and *Synchronization* choices. The variable values are also set during resolution. In our case, the application engineer must defined the name of the composition directive aspect and the URI of its resource [4].

3.4 Derivation of a Software Design using CVL

The tool screenshot shown in Figure 7 illustrates the entire derivation process. Arrows *a*) and *b*) depict the VAM model. Arrow *c*) shows a choice resolution. Arrow *d*) shows an OVP which specifies the mapping to a specific RAM model (object handle shown in arrow *e*)). Based on this model, the derivation engine executes the code of the OVPs selected by the choice resolution. The expression for the OVP bound to the RAM aspect is generic and can be used as such for new aspects than can be composed. Only the object handle that maps to the concrete aspect must be respecified. The CVL derivation engine and the workflow models can be downloaded on our website [5].

3.5 Composing the Reusable Assets using RAM

Using the generated aspect model as input, the RAM weaver recursively composes all dependent aspect models to generate a complete design concern model that corresponds to the selected configuration. In our case we selected the *ParallelExecution*, *ConditionalExecution* and *Synchronization* variants. As a result, the RAM weaver will compose *ParallelExecution*, *ConditionalExecution*, *Synchronization*, as well as the dependent aspects *Outpath*, *Inpath*, *KeyCounter*, *Named*, *ZeroToMany*, and *Map*.

The resulting structural model is shown in Figure 8[6]. Since *Outpath* is instantiated both by *ParallelExecution* and *ConditionalExecution*, there are also two instances of *Map* that map `Strings` to `WorkflowNodes`. *Map* is used a third time in *Synchronization* to map `InpathNodes` to `Integers`.

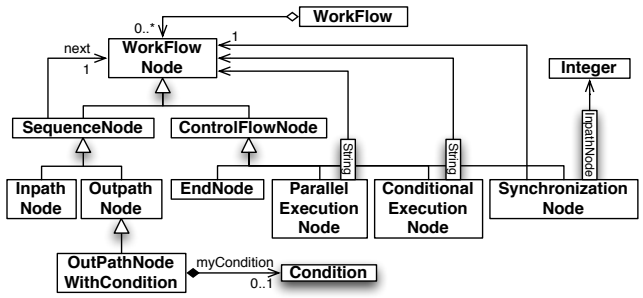

Figure 8: A Woven Workflow Design Concern Model

3.6 Discussion on the Use of OVPs

We could probably design the VRM to use other kinds of variation points such as *ObjectSubstitution* or *FragmentSubstitution*. Nevertheless, in combining AOM and CVL, the composition complexity is primarily in the aspect weaver. Consequently, the derivation engine remains simple. Consistency checking would, for instance, be part of the aspect weaver, undisturbed by the presence of the OVP. Additionally, we were able to build a generic OVP to create the aspect composition directives. Checking this OVP only needs to be done once. The CVL model could be further improved by using *ConfigurableUnit* and *ConfigurableUnitUsage* to avoid the eight repetitions of the same OVP. This will be investigated in future work.

4. RELATED WORK

This section discusses related work on aspect-oriented modelling and variability modelling approaches.

Because composing models by hand is a cumbersome task, tools and approaches have been proposed that automate significant parts of model composition [7]. The early aspect [15] and aspect-oriented modelling [1] workshop series introduced many aspect-oriented modelling (AOM) approaches that differ on (i) model composition activities, (ii) the types of inputs required other than the 2 models to be composed and (iii) the types of models that can be com-

[4]An EMF resources is a container of persisted model elements

[5]`http://www.irisa.fr/triskell/perso_pro/obarais/pmwiki.php?n=App.VARY2012`

[6]Again, only the classes directly related to the different kind of workflow nodes are shown. The structure pertaining to workflow execution (executors, execution contexts and parameters) have been omitted for space reasons.

posed. We chose RAM in this work because of its ability to compose structural and behavioural models and because the existence of large aspect-oriented models that capture variabilities. Indeed, RAM has been applied to model many software designs concerns. The biggest design concern is the AspectOPTIMA case study, a transaction support middleware product line [10, 11]. AspectOPTIMA offers support for multiple transaction models (flat, nested, multithreaded and open multithreaded transactions), different concurrency control strategies (pessimistic lock-based and optimistic time stamp-based), and different update strategies (inlace and deferred update). However, even if we used RAM in this paper, the proposed approach should also work for other AOM approaches.

Many formalisms were proposed in the past decade for variability modeling. For an exhaustive overview, we refer the readers to the literature reviews that gathered variability modeling approaches [16, 6, 2, 17, 3]. All formalisms for variability modeling could be used following the approach we introduce in this paper. In our case, we use the choice diagram proposed by CVL, very similar to an attributed feature diagram with cardinalities.

More recently, few works deal with the binding between the feature in the variability modeling and the actual assets. Let us cite for example *FeatureMapper* [5], which is a tool for combining SPL and MDE that makes it possible to bind a feature to a design model. Relying on CVL, we use in our approach the provided action language to describe in the realization model the binding between the features in the choice model and the actual RAM assets. A dedicated derivation operator is automatically obtained by implementing a dedicated opaque variation point in the CVL generic derivation operator.

Recently, several works have shown the benefits of coupling aspect-oriented modelling approaches and variability approaches. Voelter *et al.* [18] was the first to combine AOM and MDE techniques to achieve an explicit separation of concerns in software product lines. In the domain of software architecture, we cite Morin *et al.* [12] and Parra et al. [13] that combine architecture aspect models and feature models to ease the design of adaptive systems. In [14], Perrouin *et al.* proposes to specify variants by means of model fragments and the product derivation process consists in merging those fragments together.

5. CONCLUSION

In the *Reusable Aspect Models* approach, a design concern is a collection of interrelated aspect models describing a family of design solutions for a specific design problem. This paper showed how we used CVL to specify an easy-to-use product line interface for RAM design concerns. When faced with a specific design problem for which a RAM design concern exists, an application developer can consult the variation model provided by CVL to get an overview of all possible design choices. After making her choice, she passes the resulting resolution model to the CVL derivation engine, which knows about how to map features to RAM aspect models. Based on this knowledge, the derivation engine automatically creates an aspect that instantiates all the aspect models that are needed for the designer. Using this aspect, the RAM weaver generates a complete model of the chosen design concern configuration.

Acknowledgements

This work has been partially supported by VaryMDE, a collaboration between Inria and Thales Research & Technology.

6. REFERENCES

[1] Aspect-Oriented Modeling Workshop Series. http://dawis2.icb.uni-due.de/aom/workshop:start.

[2] D. Benavides, S. Segura, and A. Ruiz-cort. Automated Analysis of Feature Models 20 Years Later : A Literature Review 6. *Review Literature And Arts Of The Americas*, 2010.

[3] L. Chen and M. Ali Babar. A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology*, 53(4):344–362, Apr. 2011.

[4] F. Fleurey, Ø. Haugen, B. Møller-Pedersen, A. Svendsen, and X. Zhang. Standardizing variability - challenges and solutions. In I. Ober and I. Ober, editors, *SDL Forum*, volume 7083 of *LNCS*, pages 233–246. Springer, 2011.

[5] F. Heidenreich, J. Kopcsek, and C. Wende. FeatureMapper: Mapping Features to Models. In *Companion Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, pages 943–944. ACM, May 2008.

[6] A. Hubaux, A. Classen, M. Mendonça, and P. Heymans. A preliminary review on the application of feature diagrams in practice. In *VaMoS*, volume 37 of *ICB-Research Report*, pages 53–59. Universität Duisburg-Essen, 2010.

[7] C. Jeanneret. An analysis of model composition approaches. Master's thesis, Colorado State University and EPFL, 2008. http://goo.gl/iYcGy.

[8] J.-M. Jézéquel, O. Barais, and F. Fleurey. *Model Driven Language Engineering with Kermeta*. LNCS 6491, Springer, 2010.

[9] K. Kang, S. C. J. Hess, W. Novak, and S. Peterson. Feature-oriented domain analysis (foda). feasibility study. Technical Report CMU/SE-90-TR-21, Software Engineering Institute, Pittsburgh, PA 15213, Nov. 1990.

[10] J. Kienzle, W. Al Abed, and J. Klein. Aspect-Oriented Multi-View Modeling. In *AOSD 2009, March 1 - 6, 2009*, pages 87 – 98. ACM, March 2009.

[11] J. Kienzle, E. Duala-Ekoko, and S. Gélineau. AspectOPTIMA: A Case Study on Aspect Dependencies and Interactions. *Transactions on Aspect-Oriented Software Development*, 5:187 – 234, Mar. 2009.

[12] B. Morin, O. Barais, G. Nain, and J.-M. Jézéquel. Taming dynamically adaptive systems using models and aspects. In *ICSE*, pages 122–132. IEEE, 2009.

[13] C. A. Parra, X. Blanc, A. Cleve, and L. Duchien. Unifying design and runtime software adaptation using aspect models. *Sci. Comput. Program.*, 76(12):1247–1260, 2011.

[14] G. Perrouin, J. Klein, N. Guelfi, and J.-M. Jézéquel. Reconciling Automation and Flexibility in Product Derivation. In *12th International Software Product Line Conference (SPLC 2008)*, pages 339–348, Limerick, Ireland, Irlande, 2008. IEEE Computer Society.

[15] M. Pinto, R. Chitchyan, A. Rashid, A. Moreira, J. Araújo, P. C. Clements, E. L. A. Baniassad, and B. Tekinerdogan. Early aspects at icse 2008: workshop on aspect-oriented requirements engineering and architecture design. In *ICSE Companion*, pages 1053–1054. ACM, 2008.

[16] K. Pohl and A. Metzger. Variability management in software product line engineering. In *Proceedings of the 28th international conference on Software engineering (ICSE '06)*, pages 1049–1050. ACM, 2006.

[17] R. Rabiser, P. Grünbacher, and D. Dhungana. Requirements for product derivation support: Results from a systematic literature review and an expert survey. *Information and Software Technology*, 52(3):324–346, 2010.

[18] M. Voelter and I. Groher. Product line implementation using aspect-oriented and model-driven software development. In *11th International Software Product Line Conference (SPLC'07)*, pages 233–242. IEEE, 2007.