Modelling and Analysis with Featured Modal Contract Automata

Davide Basile
University of Florence, Italy
ISTI–CNR, Pisa, Italy
basile@isti.cnr.it

Maurice H. ter Beek
ISTI–CNR, Pisa, Italy
terbeek@isti.cnr.it

Stefania Gnesi
ISTI–CNR, Pisa, Italy
gnesi@isti.cnr.it

ABSTRACT

Featured modal contract automata (FMCA) have been proposed as a suitable formalism for modelling contract-based dynamic service product lines. A contract is a behavioural description consisting of offers and necessary and permitted service requests with different levels of criticality, to be matched with corresponding offers of other FMCA. Each contract is equipped with a feature constraint, whose features are offers or requests, and characterises a valid product orchestration. A safe orchestration of a product fulfils all necessary and the maximum number of permitted requests, such that all enabled features are available and none of its disabled features is. The entire product line orchestration can be computed from a subset of valid product orchestrations, by exploiting their (partial) ordering. The open-source prototypical toolkit FMCAT supports the specification and orchestration of FMCA, and it interfaces with FeatureIDE for importing feature models and their valid products. In this experience report, we show how to model a Hotel service product line with FMCA and how to analyse it with FMCAT.

CCS CONCEPTS

• **Software and its engineering → Formal methods**; **Software product lines**;

KEYWORDS

Service product line, Contract automata, Orchestration, Variability

ACM Reference Format:
Davide Basile, Maurice H. ter Beek, and Stefania Gnesi. 2018. Modelling and Analysis with Featured Modal Contract Automata. In *22nd International Systems and Software Product Line Conference - Volume B (SPLC '18), September 10–14, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3236405.3236408

1 INTRODUCTION

Service computing is a distributed computing paradigm concerned with the creation, publication, discovery and orchestration of services, which are autonomous, platform-independent and reusable pieces of software [20, 32]. In last year's Service Computing Manifesto [14], service design is listed as one of the four emerging research challenges in service computing for the next 10 years, together with a call for formal models.

Web applications typically reuse services in different configurations over time to adapt to changing environments or resources. Therefore the idea to organise them into dynamic service product lines was explored roughly a decade ago in the SOAPL (Service-Oriented Architectures and Product Lines) workshop series at three consecutive SPLC conferences (cf. [21, 27, 30]) and at a workshop on Web systems [10]. Recently, interest has been revived. At last year's VaMoS workshop, the Web application JHipster, with a microservice architecture, was lifted to a product line, while at last year's FormaliSE (Formal Methods in Software Engineering) workshop at ICSE an e-Government public licensing payment service was modelled and orchestrated as a service product line [17, 22].

Service contracts [2] offer a specification framework to formalise the behaviour of services in terms of obligations (i.e. service offers) and requirements (i.e. service requests) to be matched. Contracts that are fulfilled characterise an agreement among services as an orchestration (i.e. a composition) based on the satisfaction of all requirements through matching obligations. Orchestrations can dynamically adapt to the discovery of new services, to service updates and to services that have become unavailable. In [3], contract automata were introduced as a formal model for service contracts. A contract automaton represents a single service (called a principal) or an (orchestrated) composition of services. A principal's goal is to reach an accepting (final) state by matching its requests with offers of other principals. Their interactions are implicitly controlled by an orchestrator synthesised out of the principals, which directs them such that only finite executions in agreement can occur. Thus, service contracts allow to model the behaviour of an ensemble of services and the (verifiable) notion of agreement characterises safe executions of services (i.e. all requests are matched by offers).

At last year's VaMoS workshop, contract automata were equipped with modalities to distinguish necessary from permitted requests; these mimic uncontrollable and controllable actions, respectively, known from supervisory control theory (SCT) [15, 31]. In SCT, a (supervisory) controller is synthesised from models of (components of) a system and a set of requirements such that the ensemble of component models controlled by the supervisor behaves correctly, i.e. it satisfies a number of desirable properties, like the possibility to reach stable local states. In the resulting (modal) contract automata [5], only requests and matches can be classified as necessary, whereas offers are by definition permitted, based on the assumption that a service contract may always withdraw offers not needed to reach agreement. A synthesis algorithm was defined to orchestrate services in agreement, guaranteeing to fulfil all necessary requests and negotiating the maximum number of permitted requests that can be fulfilled without ruining the orchestration. The orchestration is indeed the maximally permissive controller from SCT.

Subsequently, at last year's SPLC conference, featured modal contract automata (FMCA) and its associated toolkit FMCAT were

SPLC '18, September 10–14, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5945-0/18/09.
https://doi.org/10.1145/3236405.3236408

introduced for modelling and analysing contract-based dynamic service product lines [4, 6]. FMCA extend modal contract automata in two dimensions: with the possibility to define feature constraints on service actions (requests and offers) and to distinguish between urgent, greedy and lazy necessary service requests, reflecting a decreasing level of criticality. Features are identified as service actions and each FMCA represents a behavioural product line of services equipped with feature constraints. A feature constraint can be any of the constraints used in feature models, defined as a corresponding propositional formula (cf. [7, 26]). As usual, a product is identified as a truth assignment satisfying the feature constraints.

Composition and orchestration synthesis from modal contract automata were carefully revisited to handle these two additional variability dimensions. This resulted in an algorithm for synthesising an orchestration of services in agreement, either for one product or for the entire service product line, such that it is a maximally permissive controller satisfying all feature constraints, all variants of necessary requests and the maximal number of permitted requests. Furthermore, based on a (partial) order of products of a product line, the orchestration of the entire service product line can be computed from only a subset of products. Since the number of valid products of a product line is in general exponential in the number of features, the subset of products used for such computations is typically much smaller than the set of all products.

The theory of FMCA was implemented in the open-source prototypical toolkit FMCAT [1] and applied to a toy example (with a mere 3 products) in [6]. In this experience report, a Hotel service product line is modelled and analysed with FMCAT, which for this occasion has been extended with an interface that allows to import feature models and their valid products from FeatureIDE [28]. It is also showed how the approach scales to handle many valid products.

Outline. In Section 2, we describe our modelling and analysis experience with FMCA and FMCAT, by applying the theory and its tool to a Hotel service product line. We first discuss its feature model, valid products, partial order generation, behaviour and orchestrations, followed by a few lessons learned from this experience. Section 3 concludes the paper, including a brief overview of related work and several directions for future work.

2 EXAMPLE: HOTEL SERVICE PRODUCT LINE

To illustrate the modelling and analysis capabilities of FMCA and FMCAT, consider a simple franchise of hotel reservation systems conceived as a Hotel service product line. Systems consist of three service contracts: a Hotel and two types of clients, called BusinessClient and EconomyClient. Each contract is described by an FMCA, i.e. a behavioural description equipped with a corresponding feature model (interpreted as a propositional logic formula) characterising a family of possible service contracts. FMCAT can analyse the interactions after computing a composition of FMCA. Its feature model is the conjunction of those of its operands (Fig. 5, Appendix).

2.1 The Feature Model

We describe the feature models. Next to a HotelRoom reservation service, implemented by optional sharedRoom or singleRoom, the hotel reservation system offers Payment and Notification services; the former is implemented by two alternative features for card and cash payments, the latter by optional features invoice and receipt, representing informal and formal proof of payment, respectively.

A BusinessClient is expected to request a (business) room, with customisable options: a BusinessBathroom, optionally with privateBathroom, a BusinessRoom, optionally a singleRoom, or a BusinessCancellation, optionally with freeCancellation. Also an EconomyClient is expected to request an (economic) room, with customisable options: an EconomyBathroom, optionally with sharedBathroom, an EconomyRoom, optionally a sharedRoom or a singleRoom, or an EconomyCancellation, optionally with noFreeCancellation.

Furthermore, cross-tree constraints impose any hotel reservation system to offer notifications via invoice only for cash payments. Likewise, if a sharedRoom is booked, then so is a sharedBathroom. These two constraints are specified in the Hotel feature model and the EconomyClient feature model, respectively.

Finally, the FMCA formalism allows different feature models (i.e. their logical interpretation) to predicate over the same features, to be treated as the same atom in the corresponding formula. Since this is not directly expressible in FeatureIDE, when generating a composite feature model a renaming is performed such that all feature names are unique and a set of equivalence constraints is used to bind all renamed features to the original ones. Indeed, this is the case for the sharedRoom and singleRoom features contained in the feature models of both the hotel service and its clients.

Let *FM* denote the composite feature model (Fig. 5, Appendix) and $\mathcal{F}$ its concrete features. The formula φ corresponding to *FM* is:

$$\varphi = (\mathsf{card} \oplus \mathsf{cash}) \wedge \bigvee_{f \in \mathcal{F},\ f \notin \{\mathsf{card},\mathsf{cash}\}} f \wedge (\mathsf{cash} \rightarrow \mathsf{invoice}) \wedge (\mathsf{sharedRoom} \rightarrow \mathsf{sharedBathroom}),\ \text{where } \oplus \text{ is } \textit{exclusive or}.$$

2.2 The Valid Products

FeatureIDE generates 288 valid products of *FM*, each representing a possible instantiation of the available offers and requests of the contracts. Our goal is to analyse these products to single out those admitting behaviour in *agreement* (i.e. all requests satisfied by corresponding offers). An orchestration admits all and only behaviour in agreement. As said before, features are in 1:1 correspondence with service actions. The set of features implemented in a product p (atoms interpreted true in φ) are interpreted as actions that are *mandatory* (required to occur) in the behavioural description, while features not implemented in p (atoms interpreted false in φ) are interpreted as actions *forbidden* in the behavioural description. Thus, if the behaviour of an FMCA $\mathcal{A}$ exposes all features mandatory in p and none of those forbidden by p, then $\mathcal{A}$ is said to *respect* p.

Traditionally, in a valid product all variability is resolved, i.e. φ's interpretation function is *total*, but the FMCA formalism considers also *partial* interpretations satisfying φ as valid products, i.e. in which not all variability is resolved (akin subfamilies). This is a key aspect for efficiently synthesising the orchestration of the entire service product line. Indeed, such valid products can be organised as a partial order according to a relation $\preceq$ by which $p_1 \preceq p_2$ iff the mandatory (i.e. required, R) and forbidden (F) features of p_2 are included (i.e. set inclusion) in those of p_1, in which case p_1 is said to be a *sub-product* of p_2 while p_2 is said to be a super-product of p_1. The *maximal products* are those valid products 'greater' than all other products (i.e. for which no super-products exist). For instance,

[1] Available at https://github.com/davidebasile/FMCAT

there are 4860 interpretations satisfying φ (i.e. products considered by FMCA), but only 4 are maximal products (Fig. 6, Appendix). These maximal products only instantiate at most 4 of the Hotel service product line's 10 concrete features, which suffice to satisfy φ.

Results from the theory of FMCA state that if the orchestration of a product p is (1) empty (i.e. no agreement is possible for product p), then also all its sub-products' orchestrations are empty; otherwise (2) if it is non-empty, then the orchestrations of all its sub-products are refinements (i.e. sub-automata) of the orchestration of p. Thus, it is possible to synthesise the orchestration of the entire product line by only considering these 4 maximal products, instead of 4860. It is easy to see that even for real-world feature models of up to millions of valid products the gain is considerable, confirming the scalability of our approach. Moreover, FMCAT quickly detects all valid products via a top-down visit of its partial order of products (ignoring products rooted in a non-valid product due to result (1)).

2.3 Partial Order Generation

FMCAT imports the model description and the valid products generated by FeatureIDE. To automatically generate the partial order of valid products, FMCAT exploits the valid products of FeatureIDE in the following way. Consider, for example, these three products:

P0 R:[card]; F:[cash, invoice, receipt, sharedRoom, singleRoom, sharedBathroom noFreeCancellation, privateBathroom, freeCancellation];
P48 R:[card, invoice]; F:[cash, receipt, sharedRoom, singleRoom, sharedBathroom noFreeCancellation, privateBathroom, freeCancellation];
P288 R:[card]; F:[cash, receipt, sharedRoom, singleRoom, sharedBathroom, noFreeCancellation, privateBathroom, freeCancellation];

Products P0 and P48 only differ with respect to the invoice feature (mandatory in P0, forbidden in P48). Product P288 can automatically be inferred by P0 and P48 by removing the feature invoice. Note that P288 is a super-product of both P0 and P48. FMCAT exploits this mechanism to generate the entire partial order of products.

2.4 The Behaviour

Each FMCA couples a feature model with its behaviour (an automaton). That of BusinessClient, EconomyClient and Hotel is depicted in Figs. 1, 2 and 3 (cf. Fig. 7, Appendix), respectively. In FMCA, offers are overlined (prefixed by ! in FMCAT), whilst requests are not (prefixed by ?). Permitted actions label dotted arcs (blue in FMCAT) and are suffixed by $\diamond$, whilst urgent, greedy and lazy actions label red, orange and green arcs, suffixed by $\square_u$, $\square_g$ and $\square_\ell$, respectively.

We describe the automata. All Hotel actions are permitted. From its initial state [0], it can offer a single or shared room. For a shared room, no free cancellation is the only available option, but for a single room it is also possible to choose free cancellation. The hotel contracts offer private or shared bathrooms, no matter what room was selected. In case of cash payments, only receipts are offered, whereas in case of credit card payments, only invoices are offered. Upon reaching its final state [3], it can interact with another client.

BusinessClient starts by requiring urgently a single room. This *urgent* necessary request must be matched in every possible context (i.e. the other contracts in the orchestration) to reach an agreement. This room is requested with no free cancellation and private bathroom options, after which the client offers payment by credit card, and requests a receipt or an invoice. The invoice request is *greedy*: it must be necessarily matched, but it can be delayed as long as the

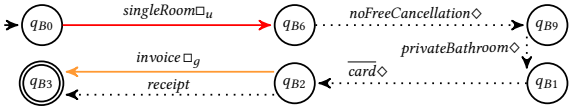

Figure 1: Behaviour of FMCA BusinessClient

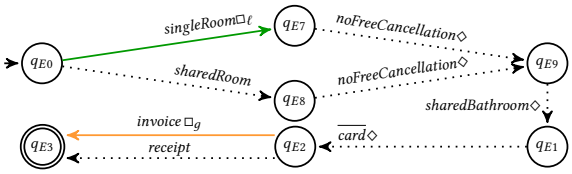

Figure 2: Behaviour of FMCA EconomyClient

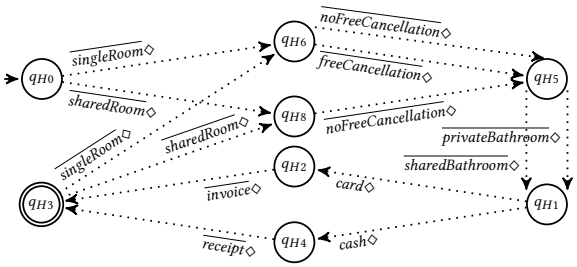

Figure 3: Behaviour of FMCA Hotel

context is not ready to match it. The receipt request is permitted, so it can be discarded in the orchestration if necessary; indeed, it is still possible to reach a final state by matching the invoice request.

EconomyClient is like BusinessClient, except that its single room request is *lazy* and it may request a shared room. A lazy request is the weakest of the necessary requests: indeed, this requirement can be deactivated in each possible context, as long as it will be matched in the orchestration at some point. The shared room request is permitted, so it can be discarded in the orchestration if necessary. Afterwards this non-deterministic choice among requesting a single or shared room, EconomyClient proceeds as BusinessClient, except for requesting a shared rather than a private bathroom.

2.5 Orchestrations

FMCAT can also compute a composition of the three automata from Figs. 1–3 to check whether there is an orchestration in agreement for one of the products of *FM*. The orchestration of a valid product p basically is the largest sub-portion of behaviour in agreement such that it respects p: all mandatory features of p are implemented and none of its forbidden features. Instead of computing the union of the orchestrations of all 288 valid products generated by FeatureIDE, it is possible to compute the orchestration of the entire service product line by only considering the *canonical products*.

Canonical products are essentially maximal products with non-empty orchestrations. The Hotel service product line has only two (Fig. 8, Appendix). The orchestration of composition BusinessClient ⊗ Hotel ⊗ EconomyClient for canonical product P4858 is depicted in Fig. 4. The one for canonical product P4859 is the same, but without state $\langle q_{H3}, q_{B8}, q_{E8} \rangle$ and its incident arcs. In these orchestrations, all contract requests are matched by corresponding offers. The orchestration $\mathcal{A}$ of the entire service product line is the union of the orchestrations of the two canonical products. Indeed,

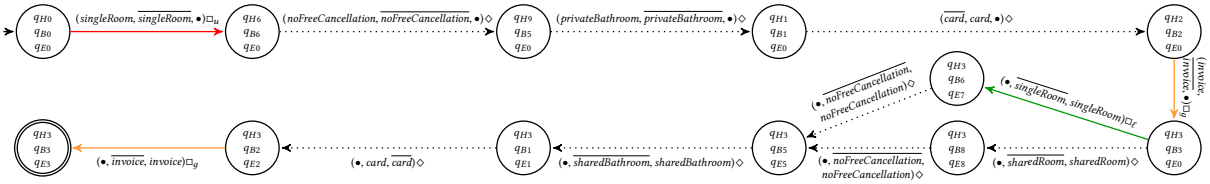

Figure 4: Orchestration of composition BusinessClient ⊗ Hotel ⊗ EconomyClient of automata for the canonical product P4858

these two products suffice to characterise the orchestrations of all products that are either empty or refinements (sub-automata) of $\mathcal{A}$.

Given the product line's orchestration $\mathcal{A}$, FMCAT can check all valid products respected by $\mathcal{A}$. In particular, of the 288 valid products FeatureIDE generated, only these two are respected by $\mathcal{A}$:

P86 R:[card, invoice, singleRoom, sharedBathroom, privateBathroom, noFreeCancellation]; F:[cash, receipt, sharedRoom, freeCancellation];
P94 R:[card, invoice, sharedRoom, singleRoom, sharedBathroom, noFreeCancellation, privateBathroom]; F:[cash, receipt, freeCancellation];

Both are orchestrations already included in $\mathcal{A}$. In particular, the orchestrations of products P86 and P94 are equal to the orchestrations of canonical products P4859 and P4858, respectively.

2.6 Modelling and Analysis with FMCA

Modelling with FMCA allows to decouple tasks of software engineers from tasks of experts in formal methods. Indeed, the syntactic description of a service product line by a feature model and its semantic description by an automaton are two separate concerns.

As a result, a software engineer can focus on designing a feature model and its valid products, leaving the task of specifying the semantics to an expert in formal modelling. Afterwards, these two aspects can be seamlessly integrated into one FMCA, making it possible to detect inconsistencies between the syntactic and semantic levels of the same formalism. In general, a software designer would like to minimise the number of valid configurations that do not admit safe behaviour (i.e. empty orchestrations) since valid products with empty orchestrations are such that the syntactic constraints provided by the feature model are not fulfilled by their behavioural description. Checking the orchestration of a service product line allows to detect products for which no safe behaviour is admitted. If such undesired products exist, one can amend the feature model or the behavioural description to obtain non-empty orchestrations.

For example, *FM* forces an invoice to be emitted in case of cash payments, and identifies cash and card payments as alternative features. Thus, in case feature cash is implemented, it requires also feature invoice to be implemented but not feature card. However, in the Hotel automaton, an invoice is emitted only for credit card payments, whereas a mere receipt is provided in case of cash payments. Thus, maximal products P4854 and P4857 are not canonical (i.e. they have an empty orchestration) because they require feature cash to be implemented. Conversely, maximal products P4858 and P4859 are canonical because cash is not implemented, and accordingly their orchestration depicted in Fig. 4 blocks (permitted) cash payments. It suffices to remove "cash → invoice" from φ or to swap actions [!receipt] and [!invoice] in the Hotel automaton to remove such inconsistency. Note that inconsistencies can easily be detected by inspecting only the 4 maximal products computed by FMCAT, instead of all 288 valid products generated by FeatureIDE.

3 CONCLUSION

We have described an experience in modelling and analysis with FMCA by showing how to model and analyse a Hotel service product line with FMCAT. A service product line's feature model was integrated with behavioural models of its services via the FMCA formalism, and safe orchestrations were synthesised in seconds, both for valid products and for the entire service product line.

FMCAT thus presents a unified SCT approach to syntactic and semantic variability of service product lines, in the sense that the resulting synthesised orchestrations respect both a feature model (and its generated valid products, imported from FeatureIDE) and the contract-based service product line behaviour (modelled as automata). Furthermore, the approach is such that it requires only a small fraction of the total number of products satisfying the feature model to be able to synthesise the safe orchestration of the service product line. The only related such integrated approaches that we are aware of are the CIF 3 toolset [12] and Behavioural Clafer [23].

CIF 3 provides first-class support for modelling a (highly modular) system's requirements and behaviour, with a formal and compositional underlying semantics based on (hybrid) transition systems, including timed behaviour, and interfaces to both UPPAAL [13], which is a well-known tool for the modelling, simulation and verification of real-time systems, and mCRL2 [18], which is an industrial strength model checker for verifying properties expressed in the modal μ-calculus with data or its feature-oriented variant of [8, 9].

In [11], it was shown how CIF 3 can synthesise a family model representing an automaton for each of the valid products of a product line from an (attributed) feature model, behavioural component models associated with the features and additional behavioural requirements like state invariants, event orderings and guards on events (reminiscent of Featured Transition Systems (FTS) [16]). The resulting family model satisfies all feature-related constraints and all behavioural requirements. As far as we know, this was the first application of SCT to product lines. We recently equipped also FMCA with real-time constraints, to be fulfilled in successful orchestrations, and we are planning to extend FMCAT, like CIF 3, to deal with such timed contract automata, by reusing libraries from timed games providing operations on zones [19, 25], to which the orchestration synthesis for timed contract automata can be reduced.

Behavioural Clafer is an extension of the general-purpose modelling language Clafer [1], providing feature modelling via a special-purpose constraint language, a light-weight class modelling language with an analyser for product generation, semantic variability by means of UML state diagrams and FTS-like automata and, finally, behavioural constraints allowing for LTL model checking. Compared to CIF 3 and to FMCAT, Behavioural Clafer provides first-class support for architectural modelling but limited support for behavioural modelling and no support for controller synthesis.

Even though FMCAT was equipped with a user-friendly GUI, it is a research prototype mainly used by the authors to perform experiments and further develop the FMCA formalism. We plan to improve the tool's documentation, which currently mainly consists of research papers, to promote its usage. Moreover, we would like to implement the tool as a plugin for Integrated Development Environments, like Eclipse for FeatureIDE, and to relieve software developers from the task of specifying the abstract behaviour (i.e. the automata). Indeed, automata could be automatically inferred given the corresponding service implementations [24] (e.g. as WS-BPEL processes [29]) to be coupled with the corresponding feature model to analyse the products' behaviour via the API of FMCAT.

REFERENCES

[1] K Bąk, Z. Diskin, M. Antkiewicz, K. Czarnecki, and A. Wąsowski. 2016. Clafer: unifying class and feature modeling. *Softw. Syst. Model.* 15, 3 (2016), 811–845. https://doi.org/10.1007/s10270-014-0441-1

[2] M. Bartoletti, T. Cimoli, and R. Zunino. 2015. Compliance in Behavioural Contracts: A Brief Survey. In *Programming Languages with Applications to Biology and Security (LNCS)*, C. Bodei, G.L. Ferrari, and C. Priami (Eds.), Vol. 9465. Springer, 103–121. https://doi.org/10.1007/978-3-319-25527-9_9

[3] D. Basile, P. Degano, and G.L. Ferrari. 2016. Automata for Specifying and Orchestrating Service Contracts. *Log. Meth. Comput. Sci.* 12, 4:6 (2016), 1–51. https://doi.org/10.2168/LMCS-12(4:6)2016

[4] D. Basile, F. Di Giandomenico, and S. Gnesi. 2017. FMCAT: Supporting Dynamic Service-based Product Lines. In *Proceedings 21st International Systems and Software Product Line Conference (SPLC'17)*, M. ter Beek, W. Cazzola, O. Díaz, M. La Rosa, R.E. López-Herrejón, T. Thüm, J. Troya, A. Ruiz-Cortés, and D. Benavides (Eds.), Vol. 2. ACM, 3–8. https://doi.org/10.1145/3109729.3109760

[5] D. Basile, F. Di Giandomenico, S. Gnesi, P. Degano, and G.L. Ferrari. 2017. Specifying Variability in Service Contracts. In *Proceedings 11th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'17)*, M.H. ter Beek, N. Siegmund, and I. Schaefer (Eds.). ACM, 20–27. https://doi.org/10.1145/3023956.3023965

[6] D. Basile, M.H. ter Beek, F. Di Giandomenico, and S. Gnesi. 2017. Orchestration of Dynamic Service Product Lines with Featured Modal Contract Automata. In *Proceedings 21st International Systems and Software Product Line Conference (SPLC'17)*, M. ter Beek, W. Cazzola, O. Díaz, M. La Rosa, R.E. López-Herrejón, T. Thüm, J. Troya, A. Ruiz-Cortés, and D. Benavides (Eds.), Vol. 2. ACM, 117–122. https://doi.org/10.1145/3109729.3109741

[7] D.S. Batory. 2005. Feature Models, Grammars, and Propositional Formulas. In *Proceedings 9th International Software Product Lines Conference (SPLC'05) (LNCS)*, J.H. Obbink and K. Pohl (Eds.), Vol. 3714. Springer, 7–20. https://doi.org/10.1007/11554844_3

[8] M.H. ter Beek, E.P. de Vink, and T.A.C. Willemse. 2016. Towards a Feature mu-Calculus Targeting SPL Verification. In *Proceedings 7th International Workshop on Formal Methods and Analysis for Software Product Line Engineering (FMSPLE'16) (EPTCS)*, J. Rubin and T. Thüm (Eds.), Vol. 206. Open Publishing Association, 61–75. https://doi.org/10.4204/EPTCS.206.6

[9] M.H. ter Beek, E.P. de Vink, and T.A.C. Willemse. 2017. Family-Based Model Checking with mCRL2. In *Proceedings 20th International Conference on Fundamental Approaches to Software Engineering (FASE'17) (LNCS)*, M. Huisman and J. Rubin (Eds.), Vol. 10202. Springer, 387–405. https://doi.org/10.1007/978-3-662-54494-5_23

[10] M.H. ter Beek, S. Gnesi, and M.N. Njima. 2011. Product Lines for Service Oriented Applications - PL for SOA. In *Proceedings 7th International Workshop on Automated Specification and Verification of Web Systems (WWV'11) (EPTCS)*, L. Kovács, R. Pugliese, and F. Tiezzi (Eds.), Vol. 61. Open Publishing Association, 34–48. https://doi.org/10.4204/EPTCS.61.3

[11] M.H. ter Beek, M.A. Reniers, and E.P. de Vink. 2016. Supervisory Controller Synthesis for Product Lines Using CIF 3. In *Proceedings 7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques (ISoLA'16) (LNCS)*, T. Margaria and B. Steffen (Eds.), Vol. 9952. Springer, 856–873. https://doi.org/10.1007/978-3-319-47166-2_59

[12] D.A. van Beek, W.J. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J.M. van de Mortel-Fronczak, and M.A. Reniers. 2014. CIF 3: Model-Based Engineering of Supervisory Controllers. In *Proceedings 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14) (LNCS)*, E. Ábrahám and K. Havelund (Eds.), Vol. 8413. Springer, 575–580. https://doi.org/10.1007/978-3-642-54862-8_48

[13] G. Behrmann, A. David, K.G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks. 2006. UPPAAL 4.0. In *Proceedings 3rd International Conference on the Quantitative Evaluation of SysTems (QEST'06)*. IEEE, 125–126. https://doi.org/10.1109/QEST.2006.59

[14] A. Bouguettaya, M. Singh, M. Huhns, Q.Z. Sheng, H. Dong, Q. Yu, A.G. Neiat, S. Mistry, B. Benatallah, B. Medjahed, M. Ouzzani, F. Casati, X. Liu, H. Wang, D. Georgakopoulos, L. Chen, S. Nepal, Z. Malik, A. Erradi, Y. Wang, B. Blake, S. Dustdar, F. Leymann, and M. Papazoglou. 2017. A Service Computing Manifesto: The Next 10 Years. *Commun. ACM* 60, 4 (April 2017), 64–72. https://doi.org/10.1145/2983528

[15] C.G. Cassandras and S. Lafortune. 2006. *Introduction to Discrete Event Systems*. Springer, New York, NY, USA. https://doi.org/10.1007/978-0-387-68612-7

[16] A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, and J.-F. Raskin. 2013. Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. *IEEE Trans. Softw. Eng.* 39, 8 (2013), 1069–1089. https://doi.org/10.1109/TSE.2012.86

[17] G. Cledou and L. Barbosa. 2017. Modeling Families of Public Licensing Services: A Case Study. In *Proceedings 5th International FME Workshop on Formal Methods in Software Engineering (FormaliSE'17)*, S. Gnesi, N. Plat, and H. Melgratti (Eds.). ACM, 37–43. https://doi.org/10.1109/FormaliSE.2017.8

[18] S. Cranen, J.F. Groote, J.J.A. Keiren, F.P.M. Stappers, E.P. de Vink, W. Wesselink, and T.A.C. Willemse. 2013. An Overview of the mCRL2 Toolset and Its Recent Advances. In *Proceedings 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13) (LNCS)*, N. Piterman and S.A. Smolka (Eds.), Vol. 7795. Springer, 199–213. https://doi.org/10.1007/978-3-642-36742-7_15

[19] A. David *et al.* 2017. UPPAAL DBM Library. Retrieved Nov 1, 2017 from http://people.cs.aau.dk/~adavid/UDBM/

[20] D. Georgakopoulos and M.P. Papazoglou (Eds.). 2008. *Service-oriented Computing*. MIT Press, Cambridge, MA, USA. https://mitpress.mit.edu/books/service-oriented-computing

[21] S. Günther and T. Berger. 2008. Service-Oriented Product Lines: Towards A Development Process and Feature Management Model for Web Services. In *Proceedings 2nd Workshop on Service-Oriented Architectures and Software Product Lines: Putting Both Together (SOAPL'08)*, R.W. Krut and S.G. Cohen (Eds.). Lero Centre, University of Limerick, Ireland, 131–136. http://www.cse.chalmers.se/~bergert/paper/2008-soapl-webservices.pdf.

[22] A. Halin, A. Nuttinck, M. Acher, X. Devroey, G. Perrouin, and P. Heymans. 2017. Yo Variability! JHipster: A Playground for Web-apps Analyses. In *Proceedings 11th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'17)*, M.H. ter Beek, N. Siegmund, and I. Schaefer (Eds.). ACM, 44–51. https://doi.org/10.1145/3023956.3023963

[23] P. Juodisius, A. Sarkar, R.R. Mukkamala, M. Antkiewicz, K. Czarnecki, and A. Wąsowski. 2016. Clafer: Lightweight Modeling of Structure and Behavior with Variability. (2016). Unpublished manuscript.

[24] C. Laneve and L. Padovani. 2015. An algebraic theory for web service contracts. *Form. Asp. Comp.* 27, 4 (2015), 613–640. https://doi.org/10.1007/s00165-015-0334-2

[25] A. Legay and L.-M. Traonouez. 2013. PyEcdar: Towards Open Source Implementation for Timed Systems. In *Proceedings 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13) (LNCS)*, D. Van Hung and M. Ogawa (Eds.), Vol. 8172. Springer, 460–463. https://doi.org/10.1007/978-3-319-02444-8_35 See also https://project.inria.fr/pyecdar/.

[26] M. Mannion. 2002. Using First-Order Logic for Product Line Model Validation. In *Proceedings 2nd International Software Product Lines Conference (SPLC'02) (LNCS)*, G.J. Chastek (Ed.), Vol. 2379. Springer, 176–187. https://doi.org/10.1007/3-540-45652-X_11

[27] F.M. Medeiros, E.S. de Almeida, and S.R. de Lemos Meira. 2009. Towards an Approach for Service-Oriented Product Line Architectures. In *Proceedings 3rd Workshop on Service-Oriented Architectures and Software Product Lines: Enhancing Variation (SOAPL'09)*, R.W. Krut and S.G. Cohen (Eds.). Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 151–157.

[28] J. Meinicke, T. Thüm, R Schröter, F. Benduhn, T. Leich, and G. Saake. 2017. *Mastering Software Variability with FeatureIDE*. Springer, Cham, Switzerland. https://doi.org/10.1007/978-3-319-61443-4

[29] OASIS. 2007. Web Services Business Process Execution Language Version 2.0. Retrieved Nov 2, 2017 from http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[30] M. Raatikainen, V. Myllärniemi, and T. Männistö. 2007. Comparison of Service and Software Product Family Modeling. In *Proceedings 1st Workshop on Service-Oriented Architectures and Software Product Lines: What is the Connection? (SOAPL'07)*, S. Cohen and R. Krut (Eds.). Special Report CMU/SEI-2008-SR-006, Carnegie Mellon University, Pittsburgh, PA, USA, C:1–C:10. https://resources.sei.cmu.edu/asset_files/SpecialReport/2008_003_001_14906.pdf.

[31] P.J. Ramadge and W.M. Wonham. 1987. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.* 25, 1 (1987), 206–230. https://doi.org/10.1137/0325013

[32] Q. Yi, X. Liu, A. Bouguettaya, and B. Medjahed. 2008. Deploying and managing Web services: issues, solutions, and directions. *VLDB J.* 17, 3 (2008), 735–572. https://doi.org/10.1007/s00778-006-0020-3

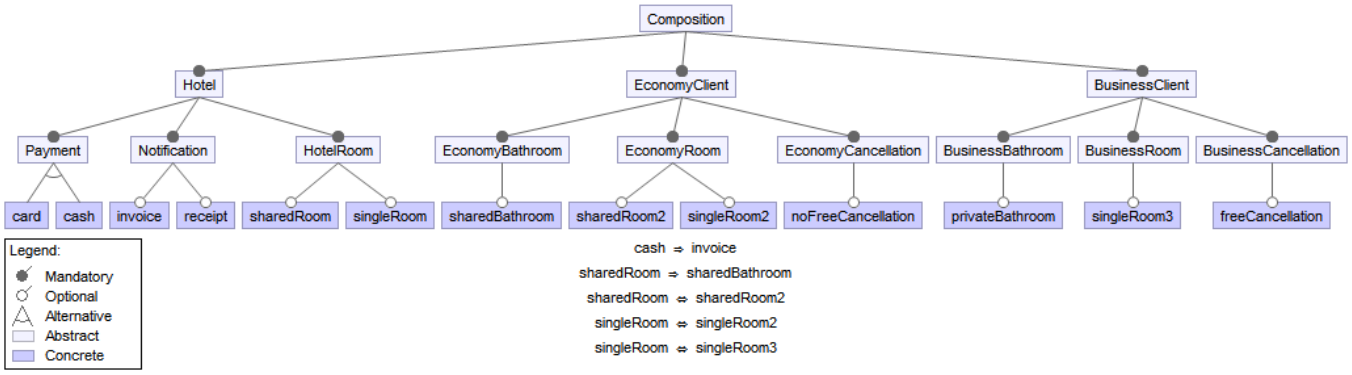

Figure 5: Feature model *FM* of Hotel service product line (designed in FeatureIDE)

A APPENDIX: TOOL SCREENSHOTS

We now provide some screenshots from tools used for this paper, i.e. FeatureIDE (not developed by us) and FMCAT (available at https://github.com/davidebasile/FMCAT).

Figure 5 depicts the composite feature model *FM* of the Hotel service product line, designed with FeatureIDE, which corresponds to the formula φ.

Figure 6 depicts the four maximal (i.e. top) products of *FM*, as computed by FMCAT, where R and F denote the set of mandatory (i.e. required) and forbidden features, respectively, of each product.

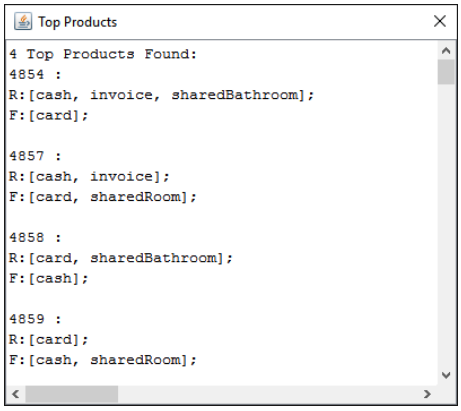

Figure 6: Maximal products of *FM* (computed by FMCAT)

Figure 7 depicts the behaviour of the FMCA Hotel, as drawn in FMCAT, which corresponds to Fig. 3.

Figure 8, finally, depicts the two canonical products of *FM*, as computed by FMCAT.

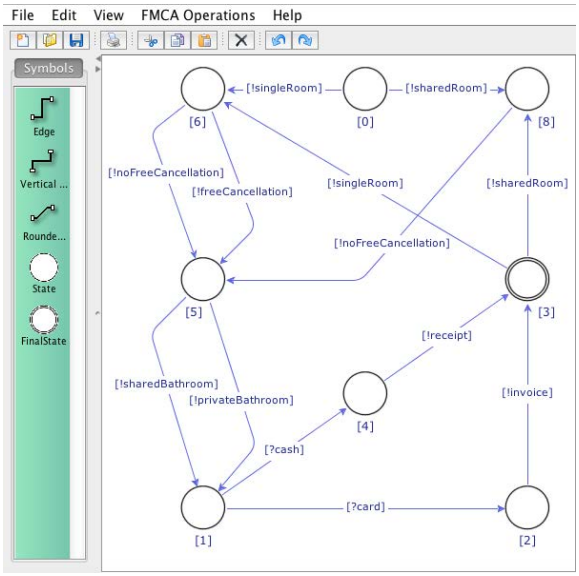

Figure 7: Behaviour of FMCA Hotel (drawn in FMCAT)

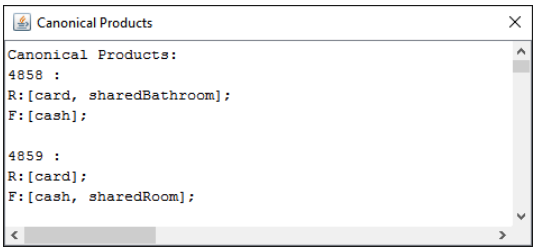

Figure 8: Canonical products of *FM* (computed by FMCAT)