Modelling and Analysis with Featured Modal Contract Automata

Davide Basile[1,2] Maurice H. ter Beek[1] Stefania Gnesi[1]

[1] ISTI–CNR, Pisa, Italy
[2] University of Florence, Italy

14 September 2018
Gothenburg

Outline

- Motivations and Background
- FMCA: modelling and verifying DSPL
- Synthesis of orchestration of services
- Separation of concerns
- Conclusions and Future work

Motivations and Background

- *A Service Computing Manifesto: The Next 10 Years* [CACM 60 (2017)]

  "Service systems have so far been built without an adequate rigorous foundation that would enable reasoning about them. [...] The design of service systems should build upon a formal model of services."

- Web applications typically reuse services in different configurations over time to adapt to changing environments or resources

⇒ *Dynamic Service Product Lines*: organise service applications into product lines, with different configurations

- Services typically provided by mutually distrusted organisations, and compete to reach their goals

⇒ *Service contracts*: formal method to specify/verify service behaviour
  - agreement among services: fulfilment of all requirements/obligations
  - orchestration of services (supervisory control), dynamic adaptation

Featured Modal Contract Automata (FMCA) [DSPL@SPLC (2017)]

- *Automata for Specifying and Orchestrating Service Contracts* [LMCS 12 (2016)]
  - Contract Automaton: single service or (orchestrated) service composition
  - Goal: reach accepting (final) state by matching requests with offers by CA
  - Interactions implicitly controlled by orchestrator synthesised out of CA, directing them such that only finite executions in agreement can occur

  ⇒ Model behaviour of service ensembles, and a (verifiable) agreement notion characterises safe execution of services: all requests matched by offers
- FMCA extend Modal Service Contract Automata (MSCA) [VaMoS (2017)]
  1. *product line* modelling → *feature model* of service (offer/request) actions
  2. *behavioural variability* → distinguish between *permitted* service actions and *urgent/greedy/lazy required* service actions of decreasing criticality
- FMCA equipped with a prototype tool FMCAT [tool@SPLC (2017)]
  - import feature models and valid products from FeatureIDE [Springer (2017)]
  - scalability: non-trivial Hotel service product line modelled and analysed

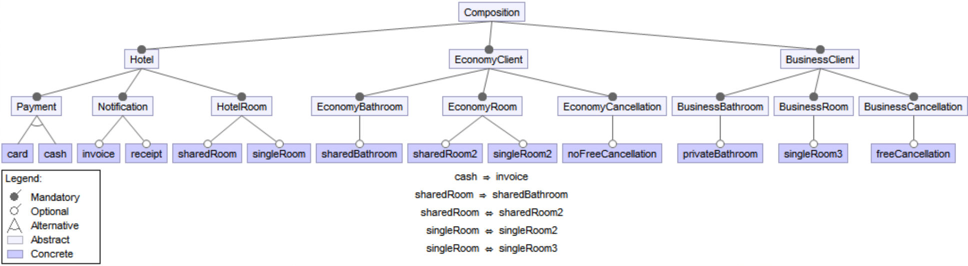
---

Available at `https://github.com/davidebasile/FMCAT`

Hotel service product line

A simple franchise of hotel reservation systems

- systems of three service contracts: Hotel, BusinessClient, EconomyClient
- each contract described by an FMCA: a behavioural description with a feature model, characterising a family of possible service contracts



$$(\text{card} \oplus \text{cash}) \wedge \bigvee_{\substack{f \in \mathcal{F}, \\ f \notin \{\text{card},\text{cash}\}}} f \wedge (\text{cash} \rightarrow \text{invoice}) \wedge (\text{sharedRoom} \rightarrow \text{sharedBathroom})$$

Valid products

- FeatureIDE generates 288 valid products from feature model φ
- FMCAT: single out those products admitting behaviour in *agreement* (i.e. all requests satisfied by corresponding offers)

$\Rightarrow$ An orchestration admits all and only behaviour in agreement

- If FMCA behaviour exposes *all* required and *no* forbidden features of a product *p*, then it *respects p*

- FMCA considers also *partial* interpretations of φ as valid products, i.e. in which not all variability is resolved (akin subfamilies)

$\Rightarrow$ This is a key aspect for efficiently synthesising the orchestration of the entire service product line from a partial order of valid products

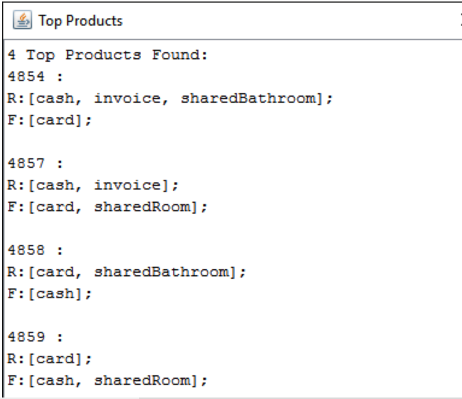
Partial order of valid products

$$p_1 \preceq p_2$$

$$\Leftrightarrow$$

required (R) and forbidden (F) features of p_2 are included in those of p_1

$\Rightarrow$ 4860 interpretations satisfying φ, but only 4 maximal products



Partial order generation

FMCAT exploits the valid products imported from FeatureIDE to automatically generate the partial order of valid 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, noFree-Cancellation, privateBathroom, freeCancellation];

(Super-)product P288 can automatically be inferred from P0 and P48: remove feature invoice

Scalability via partial order generation

Results from the theory of FMCA

1. If the orchestration of a product p is empty (i.e. no agreement possible for p), then so are all its subproducts' orchestrations
2. If it is non-empty, then all its subproducts' orchestrations are refinements (i.e. subautomata) of the orchestration of p

⇒ To synthesise the orchestration of the entire product line it suffices to consider only the maximal products

e.g. only consider the 4 maximal products, instead of all 4860 products

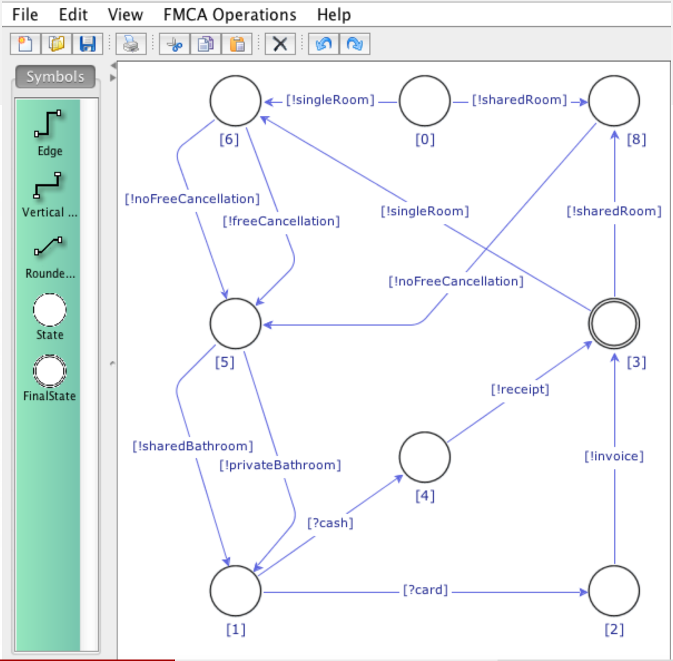
Behaviour

FMCA are an extension of MSCA; basically enhanced FSA

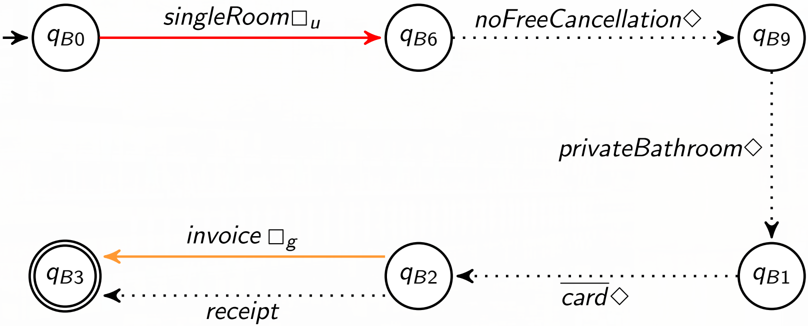
- Partitioned alphabet
  - A^r: *request* actions (prefixed by ? in FMCAT), further partitioned into
    - *permitted* actions (suffixed by $\Diamond$)
    - *urgent*/*greedy*/*lazy required* actions (suffixed by $\Box_u/\Box_g/\Box_\ell$)
  - A^o: *offer* actions (overlined, prefixed by ! in FMCAT)
- Labelled transitions
  - *may offer*: $(\bullet, \ldots, \bullet, \overline{a}, \bullet, \ldots, \bullet)\Diamond$
  - *may request*: $(\bullet, \ldots, \bullet, a, \bullet, \ldots, \bullet)\Diamond$
  - *must request*: $(\bullet, \ldots, \bullet, a, \bullet, \ldots, \bullet)\Box$
  - *may match*: $(\bullet, \ldots, \bullet, a, \bullet, \ldots, \bullet, \overline{a}, \bullet, \ldots, \bullet)\Diamond$
  - *must match*: $(\bullet, \ldots, \bullet, a, \bullet, \ldots, \bullet, \overline{a}, \bullet, \ldots, \bullet)\Box$
- Dynamically compositional, describing single and composite services
  - *rank*: the number of services in the contract
  - *single* service: $rank = 1$ and $A^r \cap co(A^o) = \varnothing$

⇒ Goal is to compute the *optimal* composition in *agreement*: maximal number of permitted requests matched by offers

Hotel service



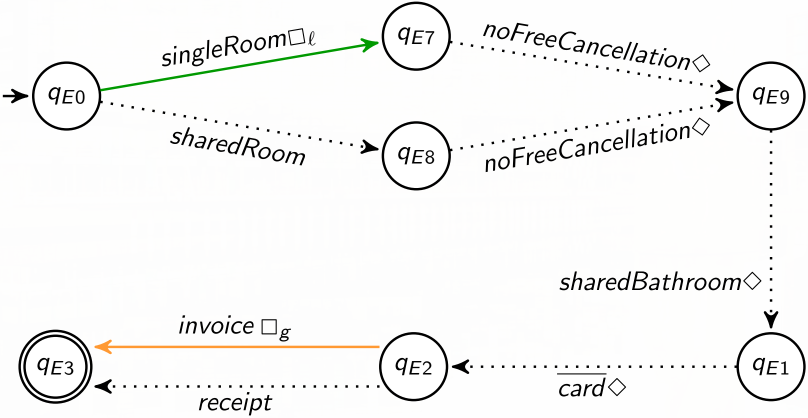
Business client service



urgent necessary request must be matched in every possible context (i.e. the other contracts in the orchestration) to reach an agreement

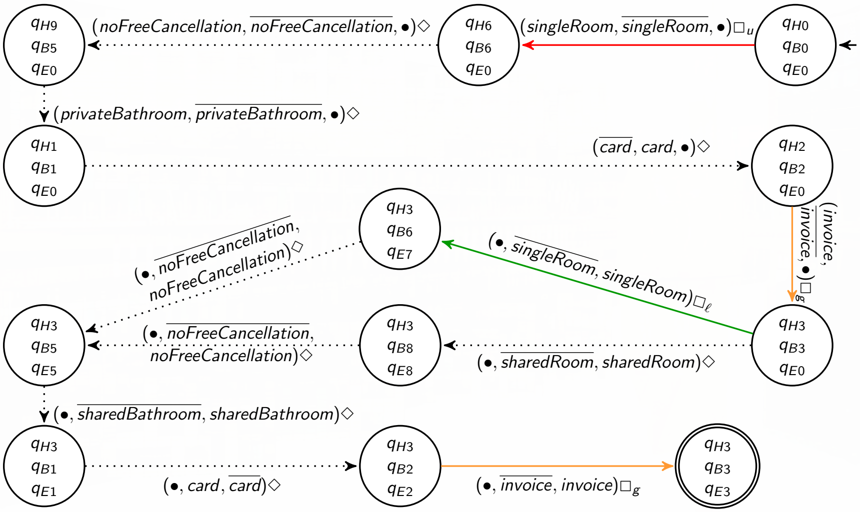
greedy necessary request must eventually be matched, but it can be delayed as long as the context is not ready to match it

Economy client service



lazy necessary request can be deactivated in every possible context, as long as it will be matched in the orchestration at some point

Orchestration $\mathcal{A}$ of service product line



FMCAT: only 2 of 288 valid products of FeatureIDE respected by $\mathcal{A}$

Separation of concerns

Decouple tasks of software engineers from tasks of experts in formal methods

- syntactic description of a service product line by a feature model
- semantic description of service contracts by contract automata

⇒ These two aspects can then be seamlessly integrated into one FMCA, allowing to detect inconsistencies between two levels of one formalism

- e.g. a software designer generally wants to minimise the number of valid configurations not admitting safe behaviour, since valid products with empty orchestrations mean that 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 all 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

Conclusions and Future work

- FMCA: unified approach to syntactic and semantic variability of DSPL
- FMCAT: open-source prototypical tool for FMCA
  - resulting synthesised orchestrations respect both a feature model (and its valid products, imported from FeatureIDE) and the contract-based service product line behaviour (modelled as contract automata)
  - only a small fraction of the total number of products that satisfy the feature model needs to be considered to be able to synthesise the safe orchestration of the service product line in seconds
- Related integrated approaches: CIF 3 [TACAS (2014)] and Clafer extended with Behaviour [Programming (2019)]
- Future work
  - quantities, time constraints [VECoS (2018)]
  - extended feature constraints: modelling QoS agreement
  - high-level modelling language on top of contract automata

---

Available at `https://github.com/davidebasile/FMCAT`

thanks for your attention!

SPLC 2018

Architecture FMCAT