Challenges in Modelling and Analyzing Quantitative Aspects of Bike-Sharing Systems*

Maurice H. ter Beek[1], Alessandro Fantechi[1,2], and Stefania Gnesi[1]

[1] ISTI–CNR, Via G. Moruzzi 1, Pisa, Italy
{terbeek,gnesi}@isti.cnr.it
[2] DINFO, Università di Firenze, Via S. Marta 3, Firenze, Italy
alessandro.fantechi@unifi.it

Abstract. Bike-sharing systems are becoming popular not only as a sustainable means of transportation in the urban environment, but also as a challenging case study that presents interesting run-time optimization problems. As a side-study within a research project aimed at quantitative analysis that used such a case study, we have observed how the deployed systems enjoy a wide variety of different features. We have therefore applied variability analysis to define a family of bike-sharing systems, and we have sought support in available tools. We have so established a tool chain that includes (academic) tools that provide different functionalities regarding the analysis of software product lines, from feature modelling to product derivation and from quantitative evaluation of the attributes of products to model checking value-passing modal specifications. The tool chain is currently experimented inside the mentioned project as a complement to more sophisticated product-based analysis techniques.

1 Introduction

Bike-sharing systems (BSS) are becoming popular not only as a sustainable means of smart transportation in the urban environment, but also as a challenging case study that presents interesting run-time optimization problems.

A case study of the EU project QUANTICOL (http://www.quanticol.eu) concerns the quantitative analysis of BSS seen as collective adaptive systems (CAS). The design of CAS must be supported by a powerful and well-founded framework for quantitative modelling and analysis. CAS consist of a large number of spatially distributed entities, which may be competing for shared resources even when collaborating to reach common goals. The nature of CAS, together with the importance of the societal goals they address, mean that it is imperative to carry out thorough analyses of their design and to investigate all aspects of their behaviour before they are put into operation. In this context it is important to realize that the design and behaviour of the individual entities from which a CAS is composed, may exhibit variability not only in the kind of features but also in the quantitative characteristics of features themselves.

* Research partly supported by the EU FP7-ICT FET-Proactive project QUANTICOL (600708) and by the Italian MIUR project CINA (PRIN 2010LHT4KM).

T. Margaria and B. Steffen (Eds.): ISoLA 2014, Part I, LNCS 8802, pp. 351–367, 2014.
© Springer-Verlag Berlin Heidelberg 2014

Starting from the BSS case study identified in QUANTICOL, we sought to apply variability analyses on a small bike-sharing product line. For this purpose, we modelled a family of BSS, covering the specification of a discrete feature model [21], the specification of several non-functional quantitative properties, and behavioural specifications. To specify and analyze this family of BSS, we chose to adopt existing feature modelling and analysis tools rather than to build yet another tool. Looking for academic, freely available tools, we realized that no single tool was ready to fully satisfy our expectations. This led to the conclusion that the best option was the synergic use of the tool chain in Fig. 1, including S.P.L.O.T. [23], FeatureIDE [28], Clafer [15] and ClaferMOO [25], and VMC [12].

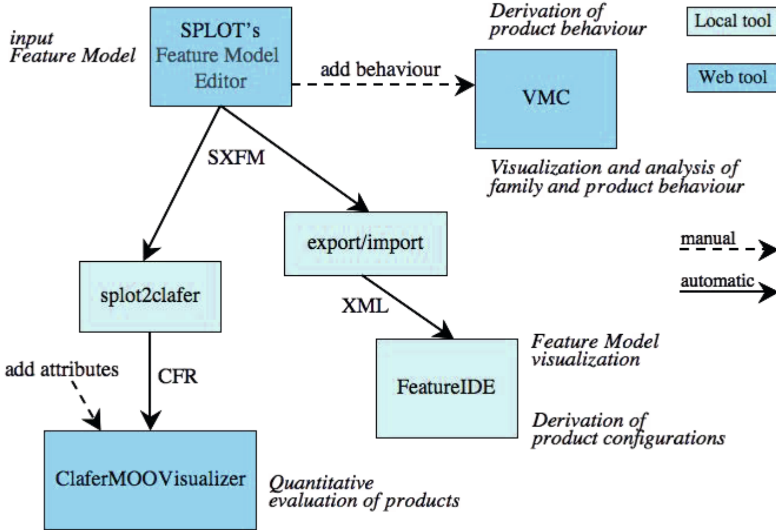


Fig. 1. The tool chain experimented in this paper

This tool chain includes academic tools that provide different functionalities regarding the analysis of software product lines, from feature modelling to product derivation and from quantitative evaluation of the attributes of products to model checking value-passing modal specifications.

We first specified a feature model of a BSS in S.P.L.O.T., which is a de facto standard for sharing feature models publicly that also allows to edit, debug, analyze, and configure feature models, after which this model was imported in FeatureIDE [28] for visualization in the FODA syntax [21,13] and automatic code generation in the future. The model was then imported in Clafer and, after the manual addition of feature attributes and global quantitative constraints over these attributes, ClaferMOOVisualizer was used for quantitative analyses and multi-objective optimization of the resulting attributed feature model. Finally, we manually specified several behavioural models of BSS in the recent extension of VMC that can handle data through value passing, allowing the automatic

generation of one, some, or all valid product behaviours and the simulation, visualization, and verification of either the full product line or a set of its valid products by properties expressed in a value-passing action-based branching-time modal temporal logic. The tool chain is currently further experimented to complement more sophisticated product-based analysis techniques in QUANTICOL.

As far as we know, there was no study available concerning the possible different realizations of a BSS starting from its description as a product line and then using methods and tools developed in the field of software product lines to

1. analyze the set of admissible products by inspecting the possible variability;
2. take into account the different attributes that may be used to measure, e.g., the development cost of the derivable products or the customer satisfaction;
3. verify satefy and liveness (e.g., availabiliy) properties of a behavioural model.

The paper is organized as follows. In §2 we introduce the bike-sharing case study. In §3 we show how to model the BSS with SPLOT and how to import it into FeatureIDE. In §4 we instead show how to import this model in Clafer, after which we extend it with attributes and evaluate it with ClaferMOOVisualizer. Finally, in §5, we consider a concrete BSS and model and analyze its behaviour with VMC. Our conclusions from this experience are presented in §6, followed by a list of future work in §7. The complete specifications of all models used in the case study presented in this paper are available in [9] and online at URL: `http://milner.inf.ed.ac.uk/wiki/files/y0R2Q6q/TRQC072014pdf.html`.

2 BSS: Bike-Sharing Systems

An increasing number of cities of varying size are adopting fully automated public bike-sharing systems (BSS) as a green urban mode of transportation [24]. The concept is simple (a user arrives at a station, pays for a bike, uses it for a while and returns it to a station) and their benefits multiple, including the reduction of vehicular traffic (congestion), pollution, and energy consumption.

The current third generation technology-based BSS have almost nothing (but the bikes) in common with the first generation free BSS introduced in Amsterdam nearly half a century ago. *Vélib′*, the well-known and highly successful BSS of the city of Paris, currently consists of over 20,000 bikes and some 1,800 stations. There are now similar BSS in more than 500 cities worldwide. The largest BSS can be found in China with upto 90,000 bikes and over 2,000 stations, one every 100 meters. Fourth generation BSS are already being developed. These include movable and solar-powered stations, electric bikes and smartphone real-time availability applications [24].

In the context of QUANTICOL we are collaborating with “PisaMo S.p.A. azienda per la mobilità pisana”, an in-house public mobility company of the Municipality of Pisa. They recently introduced the public BSS *CicloPi* in the city of Pisa, which currently consists of roughly 140 bikes and 14 stations.

More in detail, a BSS consists of parking stations distributed over a city, typically in close proximity to other public transportation hubs such as subway

and tram stations. (Subscribed) users may rent an available bike and drop it off at any station in the city. To improve the efficiency and the user satisfaction of BSS, the load between the different stations may be balanced, e.g., by using incentive schemes that may change the behaviour of users but also by efficient (dynamic) redistribution of bikes between stations.

3 Modelling a BSS: From S.P.L.O.T. to FeatureIDE

To develop an initial feature model we performed a requirements elicitation in the form of text mining a set of documents from the literature describing current BSS (mainly [24]) and the specific BSS of Pisa [20,26]. This allowed us to extract the main features of BSS and to identify their commonalities and variabilities. This led to bikes equipped with an optional localization feature (RFID or GPS) and an optional antithieves feature (which requires GPS though), parking stations with a capacity that is either fixed permanent or fixed portable or flexible, optional maintenance and redistribution of bikes, and – finally – an optional incentive scheme based on rewards. Obviously we could have taken many more features of BSS into account, but we believe that the chosen ones represent a sufficient starting point for this exploratory study.

The feature model representation in Fig. 2 was created with S.P.L.O.T.'s feature model editor, which is an online application developed by Marcílio Mendonça and others at the University of Waterloo [23]. Software Product Lines Online Tools is actually a web portal which integrates a number of research tools. S.P.L.O.T. allows to edit, debug, analyze, configure, share and download feature models. In particular, it allows to save models online to consult them later or to export them in the SXFM format.[1] It does not allow code generation, nor does it provide a way to render feature models in the graphical FODA syntax [21,13]. The main reason for which we nevertheless opted for S.P.L.O.T. as the first tool in the chain is that it is a de facto standard for sharing feature models publicly (its feature model repository currently has nearly 400 entries).

A tool that does allow to directly generate code (Java or C++) as well as a graphical representation in the FODA syntax, starting from a feature model, is FeatureIDE [28], which is an Eclipse plug-in developed mainly at the University of Magdeburg. FeatureIDE actually supports the full lifecycle of a software product line, from domain engineering to feature-oriented software development. However, it operates on feature models in an XML format. To nevertheless be able to use FeatureIDE to work with feature models created with S.P.L.O.T. (or directly with one of the feature models in its repository), it thus becomes necessary to translate the SXFM format. To this aim, FeatureIDE has a feature that automatically converts SXFM files into the desired XML format.

Using FeatureIDE's visualization functionalities we subsequently obtained the graphical representation of this feature model depicted in Fig. 3 (with an implicit conjunction among the 5 constraints). From this model, FeatureIDE allows the user to generate one of the 60,840 valid configurations (i.e., products).

[1] Simple XML Feature Model, a concise textual format to denote feature models.

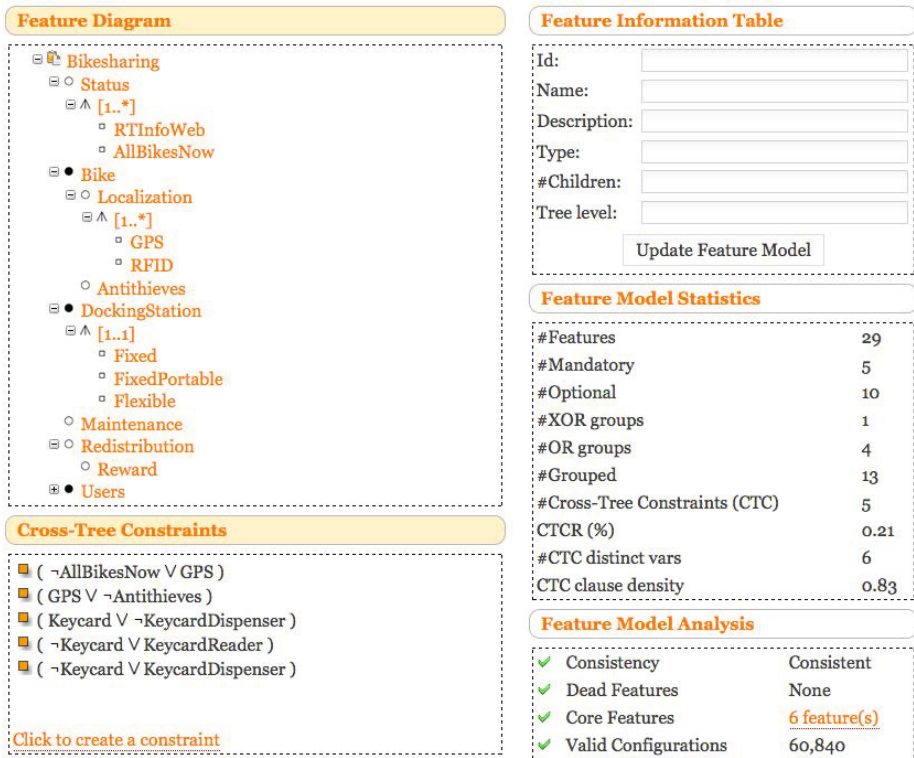


Fig. 2. The BSS feature model in S.P.L.O.T., not showing the Users' subfeatures

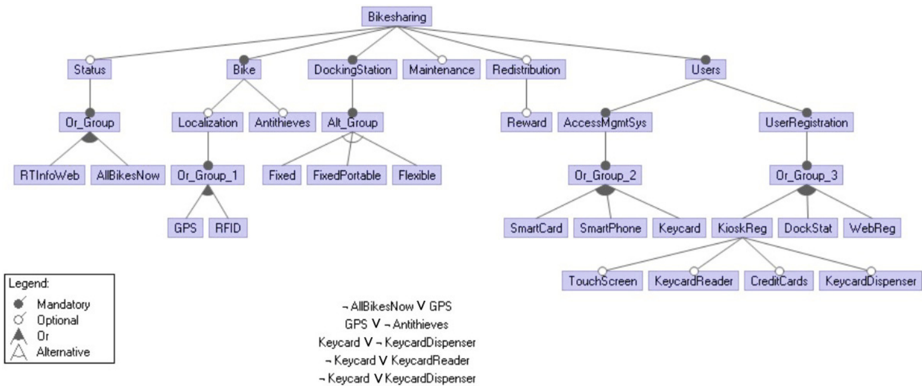


Fig. 3. The complete BSS Feature Model in FeatureIDE

4 Adding Attributes to the BSS: From S.P.L.O.T. to ClaferMOOVisualizer

Until now we considered ordinary feature models, i.e., feature diagrams modelling the hierarchical parent-child relationships between a set of features as a rooted tree and possibly some additional cross-tree constraints. As outlined in the introduction, in the context of QUANTICOL we are specifically interested in quantitative analyses of BSS, meaning that we consider the behaviour of the components of a BSS to exhibit variability not only in the kind of features they possess, but also in the quantitative (non-functional) characteristics of their features. To achieve this, we manually add attributes and quantitative constraints among attributes and features to the BSS specification and perform quantitative analyses, i.e., we perform modelling and analysis of attributed feature models [13]. Neither S.P.L.O.T. nor FeatureIDE currently cater for attributed feature models, but FeatureIDE is being extended to support quality attributes [28].

Clafer, a lightweight textual modelling language for software (product lines) developed jointly at the University of Waterloo and the IT University of Copenhagen, does allow attributed feature modelling [15]. Moreover, splot2clafer, a small tool written in Java, automatically translates files from S.P.L.O.T.'s SXFM format into the CFR format of Clafer.

In Clafer, each feature can have an associated attribute and quality constraints can be specified either globally or within the context of a feature. Think, e.g., of associating a cost to each feature and a global constraint that only allows products (feature configurations) whose total costs remain within a predefined threshold value. This is an example of a single optimization objective, but usually there can be more than one attribute associated to a feature, leading to multiple optimization objectives. It suffices to imagine that each feature also has a value for user satisfaction associated to it and while the objective might be to minimize the cost of a product it might at the same time be desirable to maximize user satisfaction.

The ClaferMOO extension of Clafer was specifically introduced to support attributed feature models as well as the resulting complex multi-objective optimization goals [25]. A multi-objective optimization problem has a set of solutions, known as the Pareto front, that represents the trade-offs between two or more conflicting objectives. Intuitively, a Pareto-optimal solution is thus such that no objective can be improved without worsening another objective. A set of Pareto-optimal variants generated by ClaferMOO can be visualized (as a multi-dimensional space of optimal variants) and explored in the interactive tool ClaferMOOVisualizer, which was specifically designed to support SPL scenarios. The tool can help understand differences among variants, establish their positioning with respect to various quality dimensions, select the most desirable variants, possibly by resolving trade-offs, and understand the impact that changes made during a product line's evolution have on a variant's quality dimensions.

The outcome of evaluating the various options (costs, benefits, etc.) in a systematic way can help finding the right BSS for a particular city, i.e., providing concrete answers to questions like:

- How many and what kind of bikes to buy?
- How many and what kind of stations to buy, and where to place them?
- Which features (antitheft, maintenance, smart services, etc.) to include?
- Include or exclude the (dynamic) redistribution of bikes?
- Set incentives for users to return bikes to less popular stations (e.g., uphill)?
- How much should the users pay and according to which charging policy?

We thus decided to annotate the features of the BSS with attributes and to define global quantitative constraints over these attributes. For now we limited ourselves to the cost and customer satisfaction and, in specific cases, capacity and security. Consequently, these constraints aim to minimize the total cost of a configuration and at the same time maximize customer satisfaction, capacity, and security of a BSS. We obtained realistic numbers for cost and security from documents from companies selling BSS (obtained from PisaMo) and kept their ratio in the model. The values for customer satisfaction instead stem from discussions with PisaMo. The specification (in Clafer's CFR format) of the resulting attributed feature model, excluding the Users (sub)features because ClaferMOOVisualizer currently runs out of memory in case of too large models, is:

```
// BBS+.cfr
abstract Feature
  customersat : integer
  cost : integer

abstract SecurityFeature : Feature
  security : integer

abstract CapacityFeature : Feature
  capacity : integer

abstract Bikesharing
  or Status : Feature ?
    [ customersat = 0 ]
    [ cost = 0 ]
    RTInfoWeb : Feature
      [ customersat = 10 ]
      [ cost = 5 ]
    AllBikesNow : Feature
      [ customersat = 20 ]
      [ cost = 10 ]
  Bike : SecurityFeature
    [ customersat = 0 ]
    [ cost = 0 ]
    [ security = 0 ]
    or Localization : SecurityFeature ?
      [ customersat = 0 ]
      [ cost = 0 ]
      [ security = 0 ]
      RFID : SecurityFeature
        [ customersat = 10 ]
        [ cost = 10 ]
        [ security = 1 ]
      GPS : SecurityFeature
        [ customersat = 15 ]
        [ cost = 15 ]
        [ security = 2 ]
    Antithieves : SecurityFeature ?
        [ customersat = 5 ]
        [ cost = 7 ]
        [ security = 4 ]
  xor DockingStation : CapacityFeature
    [ customersat = 0 ]
    [ cost = 0 ]
    Fixed : CapacityFeature
      [ customersat = 17 ]
      [ cost = 30 ]
      [ capacity = 10]
    FixedPortable: CapacityFeature
      [ customersat = 20 ]
      [ cost = 35 ]
      [ capacity = 10]
    Flexible: CapacityFeature
      [ customersat = 23 ]
      [ cost = 40 ]
      [ capacity = 20]
  Maintenance : Feature ?
      [ customersat = 15 ]
      [ cost = 10 ]
  Redistribution : Feature ?
      [ customersat = 15 ]
      [ cost = 10 ]
      Reward : Feature ?
        [ customersat = 5 ]
        [ cost = 10 ]
  [ Antithieves => GPS ]
  [ AllBikesNow => GPS ]

  total_customersat : integer =
    sum Feature.customersat
  total_cost : integer =
    sum Feature.cost
  total_security : integer =
    sum SecurityFeature.security
  total_capacity : integer =
    sum CapacityFeature.capacity

Mybike : Bikesharing
<< max Mybike.total_customersat >>
<< min Mybike.total_cost >>
<< max Mybike.total_security >>
<< max Mybike.total_capacity >>
```

By default, features are mandatory, but appended by '?' they become optional. Their hierarchy is represented by indentation. Parent features of an (exclusive) or group of subfeatures are preceeded by `or` (`xor`). Cross-tree constraints can be written as first-order logic formulae. Attributes are listed under the features, after which optimization objectives can be set to `maximize` or `minimize` their sums (or possibly other arithmetic operations on integers).

Figure 4 depicts the result of optimizing this specification with ClaferMOO-Visualizer. After running for over an hour, it generated a feature and quality matrix over four quality dimensions with 106 optimal variants (53 are visible) out of the 360 valid configurations of the feature model without the Users subtree. The feature model is in the first column, followed by sums of attributes. The numbered columns represent variants, indicating presence (green 'tick') or absence (red crossed circle) of optional features, followed by the variants' numeric quality values (summing attributes). This allows to spot common or rare features. It is also possible to filter variants by selecting features that should be present in all or none of the variants without recalculating the Pareto front. We see, e.g., that variant 51 offers maximal security and capacity at a high cost and with a high customer satisfaction. If this is too expensive, then variant 52 offers the same capacity, near-optimal security, and still a reasonable customer satisfaction at a more affordable cost. The variants are also depicted in a graph as bubbles in at most four quality dimensions (x-axis, y-axis, size, colour). This view can be narrowed down by setting specific quality ranges. The tool can also sort or compare variants (fixing features in advance), list commonalities and differences, do trade-off analysis (e.g., with preconfigured variants), etc.

5 Adding Behaviour and Value-Passing to the BSS: VMC

In recent years, we have laid the basis for the use of modal specifications and temporal logics to specify and analyze behavioural variability in SPL, by developing the modelling and verification environment presented in [2,3,4], which has been implemented in the variability model checker VMC [10,12] that is freely usable online (`http://fmt.isti.cnr.it/vmc/v6.0`). VMC is a model checker for product lines modelled as modal transition systems (MTS) [1] with additional variability constraints, but with no specific reference to feature models. This is one of the differences[2] with the successful approaches based on featured transition systems (FTS) [17,16], in which transitions are labelled with actions *and* features and an encoding of the feature model *is* included (basically, the set of features and the set of valid products in terms of their features).

VMC offers the automatic generation of one, some, or all valid product behaviours of a product line and the simulation, visualization, and verification of either the full product line or a set of its valid products. VMC's explicit-state on-the-fly model-checking algorithm allows the verification of properties expressed in so-called variability-ACTL interpreted over MTS. It moreover offers the possibility to inspect the (interactive) explanations of a verification result.

[2] The commonalities and differences between these two approaches are discussed in [3].

Fig. 4. The BSS Feature Model in ClaferMOOVisualizer, corresponding to the BSS+.cfr specification, i.e., excluding the Users features and their subfeatures

On the basis of the algorithms presented in [8], on-the-fly model checking of v-ACTL formulas over MTS can be achieved in a complexity that is linear with respect to the size of the state space. It is beyond the scope of this paper to present detailed descriptions of the model-checking algorithms and architecture underlying this family of model checkers, but we refer the interested reader to [8].

Until very recently, a critical point in this modelling and verification environment was the lack of a possibility to model an adequate representation of the data that may need to be described when considering realistic systems. We now present a case study that makes the need for data handling clear.

Inspired by [19], we consider a BSS with N stations and a fleet of M bikes. Each station i has a capacity K_i. The dynamic behaviour of the system is then:

1. Users arrive at station i.
2. If a user arrives at a station and there is no available bike, then the user leaves the system.
3. Otherwise, the user takes a bike, rides it for a while, and then chooses station j to return it to.
4. When the user arrives at station j, if there are less than K_j bikes in this station, then the user returns the bike and leaves the system.
5. If the station is full, then the user chooses another station, say k, and goes there.
6. A bike redistribution activity *may* be requested and *may* possibly be fulfilled.
7. The user can repeat these steps, riding a bike again for a while until the user returns the bike.

This list contains a mix of a kind of static constraints stemming from the differences in configuration (features) between products, such as the optional possibility to have a redistribution mechanism, as well as more operational constraints defining the behaviour of products through admitted sequences (temporal orderings) of actions or operations implementing features according to certain values.

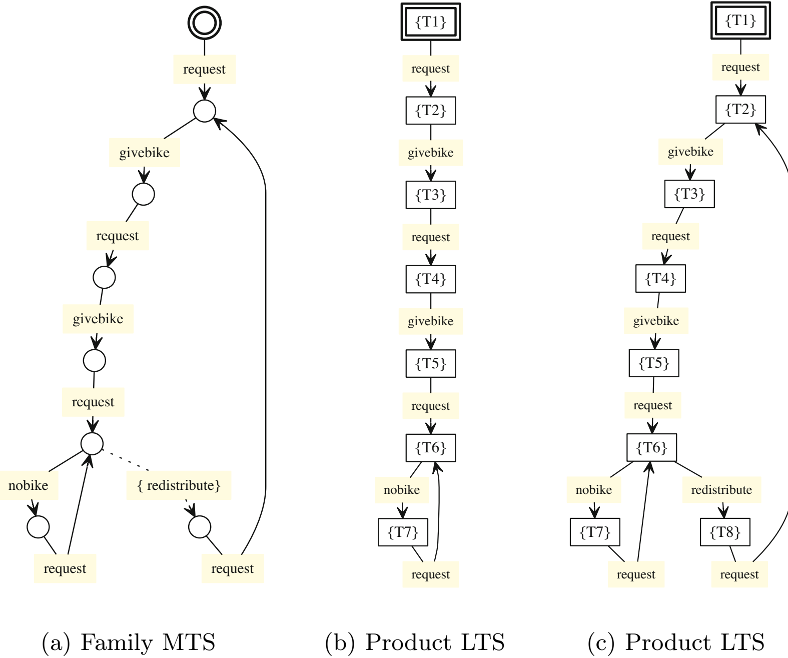
As a first step towards more complex data handling, the latest version of VMC (v6.0) accepts models specified in a value-passing modal process algebra and allows model checking of properties expressed in a value-passing action-based branching-time modal temporal logic. The formal definitions of the syntax and semantics of VMC's input language and of its v-ACTL logic can be found in [11]. We illustrate these new features of VMC by means of two simple yet intuitive examples from [11] inspired by the case study.

We first specify the behaviour of a family of bike-sharing stations in the value-passing modal process algebra. Processes can pass and receive integer parameter values (and store them in a variable preceded by a '`?`'), actions can be optional in which case they are typed `may`, and nondeterministic choice can be guarded by a comparison of values. A system definition must be complemented with a top term of the form `net SYSTEM = P`, where `P` is the initial process (or composition of processes). The below specification accounts for the possibility of having a dynamic redistribution scheme as an optional feature of the BSS. Without loss of generality, we assume a bike-sharing station with 2 as its maximum capacity.

```
Station(X) = request.StationBikeRequested(X)
StationBikeRequested(Y) =
  [Y<1] ( nobike.Station(Y) + redistribute(may).Station(Y+2) ) +
  [Y>0] givebike.Station(Y-1)

net BSS = Station(2)
```

From this specification of a family of bike-sharing stations, VMC generates the MTS in Fig. 5(a) and derives its possible product behaviours in Figs. 5(b)-5(c).



(a) Family MTS (b) Product LTS (c) Product LTS

Fig. 5. (a)-(c) A family MTS and two product LTS generated by VMC

To consider also the behaviour of a user whose bike request can result in either a bike, no bike, or a redistribution, we can specify the following BSS family:

```
User = request.(givebike.User + nobike.User + redistribute.User)

net BSS = Station(2) /request,givebike,nobike,redistribute/ User
```

Due to the simplistic user behaviour and the synchronous parallel composition[3] of `Station(2)` and `User` on all possible actions, this specification actually results in the same family behaviour (MTS) and product behaviours (LTS) as in Fig. 5.

[3] The parallel composition operator is parametrized by the actions `/.../` to synchronize.

To illustrate what kind of variability analyses can be performed, we present a few properties expressed in v-ACTL, and the result of model checking them with VMC against the example BSS depicted in Fig. 5:[4]

Eventually it must occur that no more bike is available: $EF^{\Box}\,\{nobike\}\ true$. This formula is true (due to $F^{\Box}$ only must actions may occur before *nobike*).

It is always the case that eventually it must occur that no bike is available: $AG\,EF^{\Box}\,\{nobike\}\ true$. Also this formula is true.

It is possible for the user to request and receive a bike for three times in a row: $\langle request\rangle\,\langle givebike\rangle\,\langle request\rangle\,\langle givebike\rangle\,\langle request\rangle\,\langle givebike\rangle\ true$. This formula is of course false for a station of capacity 2.

As a final example, we model a possibly infinite number of users that take a bike from station I to station J. Initially, station I has N bikes, which it gives (when available) to a requesting user or accepts from a returning user. If the station receives more than M bikes, the exceeding $N - M$ bikes are distributed to station J. Station I must accept all bikes distributed by other stations or returned by a user (possibly for redistribution). It could easily be extended to N stations and K groups of users that take a bike from one station to another.

```
Station(I,N,J,M) =
  request(I).
    ( [N = 0] nobike(I).Station(I,N,J,M) +
      [N > 0] givebike(I).Station(I,N-1,J,M) ) +
  return(I).Station(I,N+1,J,M) +
  redistribute(may,?FROM,?TO,?K).
    ( [TO = I] Station(I,N+K,J,M) +
      [TO /= I] Station(I,N,J,M) ) +
  [N > M] redistribute(may,I,J,N-M).Station(I,M,J,M)

-- two stations:
net STATIONS =
  Station(s1,2,s2,2) /redistribute/ Station(s2,2,s1,2)

Users(I,J) =
  request(I).
    ( givebike(I).return(J).Users(I,J) +
      nobike(I).Users(I,J) )

--  one or two groups of users
net USERS = Users(s1,s2) -- // Users(s2,s1)

net BSS = STATIONS /request,givebike,nobike,return/ USERS
```

[4] In VMC, $[\,]^{\Box}$, μ, ν and $F^{\Box}$ need to be written as `[]#`, `min`, `max` and `F#`, respectively.

Note that the two stations only synchronize on redistribution, which is an optional may action with parameter values to distribute $N - M$ bikes from station I to J. From this specification of a family of bike-sharing stations, VMC generates the MTS with 18 states depicted in Fig. 6 in case there is only one user group (i.e., `net USERS = Users(s1,s2)`); in case of two user groups (i.e., `net USERS = Users(s1,s2) // Users(s2,s1)`) it has 224 states.[5]

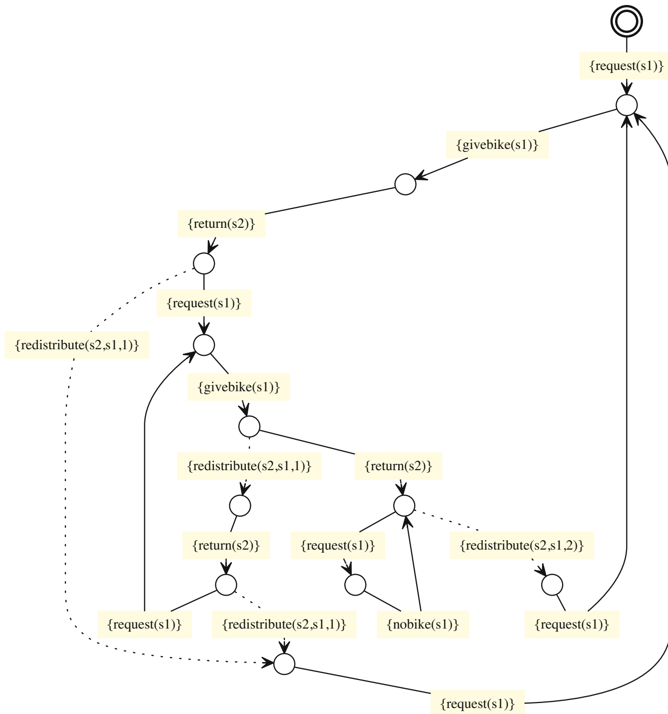


Fig. 6. A family MTS of a BSS with 2 stations and 1 group of users generated by VMC

We model checked the following properties expressed in v-ACTL with VMC, now against the example family of BSS with one user group depicted in Fig. 6:

Eventually it must occur that station 1 has no bikes: $EF^{\Box}\,\{nobike(s1)\}\ true$. This formula is true (cf. the path of only must actions leading to *nobike(s1)*).

Eventually it may occur that station 2 has no more bikes: $EF\,\{nobike(s2)\}\ true$. This formula however is false.

[5] In VMC, text or code can be commented out by prefixing it with `--`.

For all products, if redistribution is implemented, then it is always the case that eventually station 1 must give a bike:[6] $(\neg\, EF^{\square}\, \{\mathit{redistribute(*,s1,*)}\}\ \mathit{true}) \vee (AG\, EF^{\square}\, \{\mathit{givebike(s1)}\}\ \mathit{true})$. This formula is true for all products (LTS) of the family (MTS in Fig. 6).

6 Conclusions

Both S.P.L.O.T. and FeatureIDE provide the key functioning of what can be expected from a typical feature modelling tool: Creating, editing and analyzing a feature model, providing some statistics of the feature model, and deriving product configurations. A careful account, based on user experiences, of the commonalities and variability of the two tools is presented in [27], which confirms the fact that neither of the two is better than the other in all circumstances.

S.P.L.O.T. is a Web-based tool, including a repository of hundreds of feature models. It is quite user friendly and immediate to use also thanks to the availability of previously developed models. However, adding the model to the aforementioned repository is mandatory, which raises concerns over the privacy of the developed models that can thus be accessed and modified by anyone. Furthermore, S.P.L.O.T. allows only a single product configuration to be created, which however cannot be saved in the repository.

FeatureIDE, on the other hand, is a locally executable tool, integrated in Eclipse. Hence, it allows not only the generation of products as feature combinations, but it also allows to automatically generate code skeletons that reflect the feature structure of a product within Eclipse itself. Although we did not exploit this feature in this paper, we consider it important for intended future work on the BSS case study within QUANTICOL.

The complementarity of the two tools with respect to the above issues has been exploited in the experience described in this paper by first defining the feature model in S.P.L.O.T. and then importing it into FeatureIDE.

Subsequently, we adopted the online tool ClaferMOOVisualizer for quantitative analyses of an attributed feature model, since – as far as we know – it is the only tool that exhibits this functionality. Specific tooling exists to interface with S.P.L.O.T., and this was an important reason for maintaining a copy of the feature model in S.P.L.O.T. Unfortunately, we had to reduce the Clafer specification of an attributed feature model of the BSS by leaving out the entire Users feature and its subfeatures. The reason is that ClaferMOOVisualizer currently runs out of memory in case of too large models.

Finally, we moved from the analysis of structural aspects of BSS to that of behavioural aspects by considering possible behaviours of BSS by manually defining process-algebraic models of both the users and the docking stations and verifying some illustrative properties with the Web-based model checking tool VMC. More specifically, we used the most recent value-passing extension of the aforementioned modelling and verification environment.

---

[6] Note how ‘*’ can be used as a ‘don’t care’ symbol for parameter values.

Our overall experience with the tools was influenced by the fact that all of them are academic tools that at times present some minor problems: S.P.L.O.T., FeatureIDE, and VMC showed more maturity in this respect, while the online version of ClaferMOOVisualizer still manifests some instability.

Obviously, a single standard format for feature models and the use of locally running versions of the tools would increase the synergy between the tools. For now, however, the exploited tool chain was sufficient for our aim of a preliminary modelling and analysis of a bike-sharing product line.

7 Future Work

In the context of planned future work on the BSS case study within QUANTICOL, we are currently studying a further, more general, parametric extension of the above environment as well as the addition of a quantitative dimension to the behavioural model. For the latter aim, we are considering an extension to weighted MTS [5]. For the former aim, on the other hand, we plan to borrow ideas from parametric MTS [14] and from the way in which the parametrized processes and data handling features of the formal specification language mCRL2 [18] (`http://www.mcrl2.org`) have been exploited for SPL in [7,6,22].

Actually there exists specific 'quantitative' behaviour (e.g., finding another docking station if the initially chosen station is found full) that might lower or raise user satisfaction, based on the success rate, thus impacting the attributes in the feature model. It remains a challenge for the future to try to capture also such a scenario in our approach.

Acknowledgments. We thank Marco Bertini from PisaMo S.p.A. for generously sharing with us his knowledge on BSS in general and *CicloPi* in particular. We thank Franco Mazzanti, who is the developer of VMC, for helping us with §5. Finally, we thank the reviewers for their useful comments.

References

1. Antonik, A., Huth, M., Larsen, K.G., Nyman, U., Wąsowski, A.: 20 Years of Modal and Mixed Specifications. Bulletin of the EATCS 95, 94–129 (2008)
2. Asirelli, P., ter Beek, M.H., Fantechi, A., Gnesi, S.: A Logical Framework to Deal with Variability. In: Méry, D., Merz, S. (eds.) IFM 2010. LNCS, vol. 6396, pp. 43–58. Springer, Heidelberg (2010)
3. Asirelli, P., ter Beek, M.H., Fantechi, A., Gnesi, S.: Formal Description of Variability in Product Families. In: de Almeida, E.S., Kishi, T., Schwanninger, C., John, I., Schmid, K. (eds.) Proceedings of the 15th International Software Product Lines Conference (SPLC 2011), pp. 130–139. IEEE (2011)
4. Asirelli, P., ter Beek, M.H., Fantechi, A., Gnesi, S.: A Compositional Framework to Derive Product Line Behavioural Descriptions. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part I. LNCS, vol. 7609, pp. 146–161. Springer, Heidelberg (2012)

5. Bauer, S.S., Fahrenberg, U., Juhl, L., Larsen, K.G., Legay, A., Thrane, C.R.: Weighted modal transition systems. Formal Methods in System Design 42(2), 193–220 (2013)
6. ter Beek, M.H., de Vink, E.P.: Towards Modular Verification of Software Product Lines with mCRL2. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part I. LNCS, vol. 8802, pp. 368–385. Springer, Heidelberg (2014)
7. ter Beek, M.H., de Vink, E.P.: Using mCRL2 for the analysis of software product lines. In: Proceedings of the 2nd FME Workshop on Formal Methods in Software Engineering (FormaliSE 2014). IEEE (2014)
8. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: A state/event-based model-checking approach for the analysis of abstract system properties. Science of Computer Programming 76(2), 119–135 (2011)
9. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: A collection of models of a bike-sharing case study. Technical Report TR-QC-07-2014, QUANTICOL (May 2014), http://milner.inf.ed.ac.uk/wiki/files/y0R2Q6q/TRQC072014pdf.html
10. ter Beek, M.H., Gnesi, S., Mazzanti, F.: Demonstration of a model checker for the analysis of product variability. In: Proceedings of the 16th International Software Product Line Conference (SPLC 2012), vol. 2, pp. 242–245. ACM (2012)
11. ter Beek, M.H., Gnesi, S., Mazzanti, F.: Model Checking Value-Passing Modal Specifications. In: Perspectives of System Informatics - Revised selected papers of the 9th International Andrei Ershov Memorial Conference (PSI 2014). LNCS. Springer (2014)
12. ter Beek, M.H., Mazzanti, F., Sulova, A.: VMC: A Tool for Product Variability Analysis. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012. LNCS, vol. 7436, pp. 450–454. Springer, Heidelberg (2012)
13. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated Analysis of Feature Models 20 Years Later: A Literature Review. Information Systems 35(6) (2010)
14. Beneš, N., Křetínský, J., Larsen, K.G., Møller, M.H., Srba, J.: Parametric Modal Transition Systems. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 275–289. Springer, Heidelberg (2011)
15. Bąk, K., Czarnecki, K., Wąsowski, A.: Feature and Meta-Models in Clafer: Mixed, Specialized, and Coupled. In: Malloy, B., Staab, S., van den Brand, M. (eds.) SLE 2010. LNCS, vol. 6563, pp. 102–122. Springer, Heidelberg (2011)
16. Classen, A., Cordy, M., Heymans, P., Legay, A., Schobbens, P.-Y.: Formal semantics, modular specification, and symbolic verification of product-line behaviour. Science of Computer Programming 80(B), 416–439 (2014)
17. Classen, A., Cordy, M., Schobbens, P.-Y., Heymans, P., Legay, A., Raskin, J.-F.: Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. IEEE Transactions on Software Engineering 39(8), 1069–1089 (2013)
18. Cranen, S., Groote, J.F., Keiren, J.J.A., Stappers, F.P.M., de Vink, E.P., Wesselink, W., Willemse, T.A.C.: An Overview of the mCRL2 Toolset and Its Recent Advances. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013 (ETAPS 2013). LNCS, vol. 7795, pp. 199–213. Springer, Heidelberg (2013)
19. Fricker, C., Gast, N.: Incentives and Redistribution in Bike-Sharing Systems with Stations of Finite Capacity. arXiv:1201.1178v3 [nlin.AO] (September 2013)
20. Gianfrotta, L., Topazzini, S.: Progettare Servizi Pubblici: Elaborazione di un Modello per lo Sviluppo di Nuovi Servizi e sua Applicazione al caso Bike Sharing di Pisa. Master's thesis, Università di Pisa (2013) (in Italian)

21. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Carnegie Mellon University (November 1990)
22. Lochau, M., Mennicke, S., Baller, H., Ribbeck, L.: DeltaCCS: A Core Calculus for Behavioral Change. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part I. LNCS, vol. 8802, pp. 320–335. Springer, Heidelberg (2014)
23. Mendonça, M., Branco, M., Cowan, D.D.: S.P.L.O.T.: software product lines online tools. In: Arora, S., Leavens, G.T. (eds.) Companion Proceedings of the 24th Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2009), pp. 761–762. ACM (2009)
24. Midgley, P.: Bicycle-Sharing Schemes: Enhancing Sustainable Mobility in Urban Areas. Background Paper CSD19/2011/BP8, Commission on Sustainable Development, United Nations Department of Economic and Social Affairs (May 2011)
25. Murashkin, A., Antkiewicz, M., Rayside, D., Czarnecki, K.: Visualization and exploration of optimal variants in product line engineering. In: Kishi, T., Jarzabek, S., Gnesi, S. (eds.) Proceedings of the 17th International Software Product Line Conference (SPLC 2013), pp. 111–115. ACM (2013)
26. Niccolai, C., Zanzi, E.: Progettare i servizi: Creazione di un modello di validità generale e applicazione al servizio di Bike Sharing a Pisa. Master's thesis, Università di Pisa (2013) (in Italian)
27. Pereira, J.A., Souza, C., Figueiredo, E., Abílio, R., Vale, G., Costa, H.A.X.: Software Variability Management: An Exploratory Study with Two Feature Modeling Tools. In: Proceedings of the 7th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS 2013), pp. 20–29. IEEE (2013)
28. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: FeatureIDE: An extensible framework for feature-oriented software development. Science of Computer Programming 79, 70–85 (2014)