Clafer Tools for Product Line Engineering

http://clafer.org

Michał Antkiewicz, Kacper Bąk, Alexandr Murashkin,
Jimmy Liang, Rafael Olaechea, Krzysztof Czarnecki
Generative Software Development Lab
University of Waterloo, Waterloo, Canada
{mantkiew, kbak, amurashk, jliang, rolaechea, kczarnec}@gsd.uwaterloo.ca

ABSTRACT

Clafer is a lightweight yet expressive language for structural modeling: feature modeling and configuration, class and object modeling, and metamodeling. Clafer Tools is an integrated set of tools based on Clafer. In this paper, we describe different product-line variability modeling scenarios of Clafer Tools from the viewpoints of product-line owner, product-line engineer, and product engineer.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**]: Languages; D.2.13 [**Reusable Software**]: Domain engineering; F.4.1 [**Mathematical Logic**]: Logic and constraint programming; F.4.3 [**Formal Languages**]: Decision problems; G.1.6 [**Optimization**]: Constrained optimization; H.5.2 [**User Interfaces**]: Interaction styles; I.6.4 [**Model Validation and Analysis**]

General Terms

modeling, features, optimal variant, feature modeling, product line engineering, Pareto front visualization, exploration

Keywords

Clafer, ClaferWiki, ClaferIG, ClaferMOO, ClaferMOO Visualizer

1. INTRODUCTION

Clafer Tools is a a set of tools supporting many different tasks in Product Line Engineering (PLE) related to variability modeling, configuration, verification, and validation. In this paper, we demonstrate how the tools support different PLE scenarios from the point of view of three personas: *product-line owner*, *product-line engineer*, and *product engineer*. In general, product-line owners are concerned with getting the benefits of applying the product-line approach, deriving value from their product line, increasing market share and competitiveness, and product-line planning and scoping. Product-line engineers are concerned with creating and evolving a shared product-line architecture, managing a feature set and implementation assets, and supporting product teams. Product engineers are concerned with deriving correct variants from the product line that satisfy customer's requirements.

SPLC '13 Tokyo, Japan

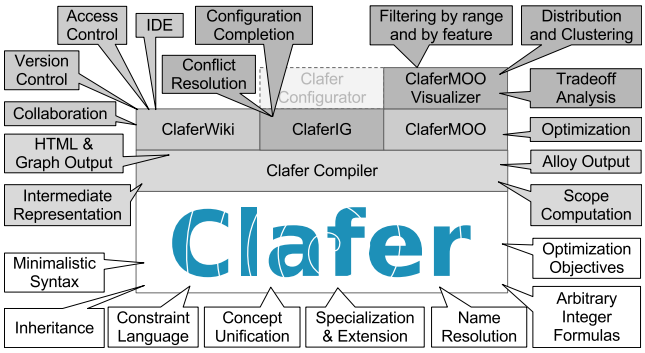

Figure 1: Architecture and Capabilities of Clafer and Tools

Clafer Tools are based on Clafer [4, 10], a lightweight modeling language. Clafer (Class feature reference) is a structural modeling language with minimalistic syntax and rich semantics equivalent to first-order relational logic. Clafer can be used for feature modeling, specialization, and configuration, metamodeling, and domain modeling. Distinguishing features of Clafer are: unification of many modeling concepts (class, instance, attribute, reference, feature, and feature group) into a single concept *clafer*, powerful constraint language, name resolution, and optimization objectives.

Figure 1 presents a layered architecture of Clafer Tools and the major capabilities provided by the language Clafer and the tools. Clafer Compiler [3] is a reference implementation of the language and it is the basis of all other tools. The compiler outputs different formats, such as Alloy, HTML, and the *intermediate representation* (IR) of the model. The IR is the result of compilation and contains fully resolved information about the model. The IR is primarily used by the tools. The compiler also computes approximate *scopes*, that is, smallest numbers of objects of each type needed to instantiate the model. The scopes enable automatic model instantiation by finite scope reasoners such as Alloy.

Clafer Wiki [5] is a collaborative integrated development environment (IDE) for modeling in Clafer. The wiki integrates the Clafer compiler and renders the models as hyperlinked HTML and as graphs. The wiki supports *literate-*

style modeling: fragments of a model can be interwoven with rich text. All model fragments collected from a page are compiled together, allowing for hyperlinking. The wiki also provides parse and compile error highlighting, identifier use-to-definition navigation via hyperlinks, simplified graph rendering and CVL [9] variability abstraction rendering, version control based on Git, and access control. A public instance of the wiki, called Model Wiki, showcases different models in Clafer (link available on project web site).

Clafer Instance Generator (ClaferIG) [6] is a reasoner for Clafer: it finds instances of models or identifies a set of conflicting constraints. The instance generator can be used for model validation and verification, and configuration completion. The instance generator relies on the scopes computed by the compiler and uses Alloy [1] as a backend reasoner. We are also working on an alternative backend using Choco constraint solver [2]. Users can adjust the scopes in the rare cases whereby the computed scopes are insufficient.

Clafer Multi-Objective Optimizer (ClaferMOO) [7, 12] is a reasoner for the *attributed-feature models* subset of Clafer, which finds a set of Pareto-optimal model instances given a set of optimization objectives (called Pareto front). ClaferMOO can be used for discovering feature configurations of optimal variants within a product line.

Finally, ClaferMOO Visualizer [8] is an interactive, web-based, graphical user interface (GUI) for visualization and exploration of the set of Pareto-optimal instances generated by ClaferMOO. The visualizer provides a view into the multi-dimensional Pareto front showing up to four chosen dimensions at a time. The visualizer can be used for various analyses of the Pareto front: observing density and clustering of variants, observing correlations, top-down analysis using selection by feature and by target quality ranges, bottom-up analysis by comparing variants selected manually (tradeoff analysis), and analysis of pre-configured variants versus the optimal variants [11]. A public instance of the visualizer is available (link available on project web site).

Clafer Configurator is a new tool we are currently implementing, but it is not yet released. The configurator will be interactive and web-based and it will provide a graphical user interface for working with ClaferIG.

In the following section, we describe how Clafer Tools can be used in different product-line engineering scenarios by telling a story of a fictional company: ACME.

2. STORY

ACME is a supplier of a range of specialty Android phones designed to satisfy specific requirements of large corporate and government organizations. New variants were evolved from old ones by applying the *clone and own* approach.

Motivation. Alice, the leader of ACME, wants to 1) apply the product-line approach to reduce time-to-market, 2) scope the product line to gain competitive advantage in the market, 3) discover which variants are best in their target customer segments and estimate the cost and predict properties of new variants, and 4) sell customized products in small batches to reach customers who are not currently economical to engage. Alice asks Bob, a chief product-line engineer at ACME, to create a product-line architecture and a platform that would eventually cover 100% of ACME's products.

2.1 Product Line Engineering

Bob decides to apply the incremental approach and he begins with identifying two similar variants that can constitute the initial platform. Bob contacts product teams and they each submit models of Android phones (see Fig. 2a). The two models (`AndroidPhone1` and `AndroidPhone2`) are modeled using Clafer—each is described by a hierarchy of features expressed using indentation. The models represent concrete variants and therefore they contain no variability. We only focus on `Connectivity` and `Battery` features.

```
AndroidPhone1
    Connectivity
        Bluetooth40
        GSM
        WiFi
        USB
    Battery
        LiBattery1450

AndroidPhone2
    Connectivity
        Bluetooth21
        GSM
        USB
    Battery
        LiBattery1150
```

(a) Models of existing variants

```
MobilePhone
    Connectivity
        xor Bluetooth
            Bluetooth20 ?
            Bluetooth21 ?
            Bluetooth40 ?
        GSM
        LTE ?
            [LiBattery1400]
        WiFi ?
        USB ?
    xor Battery
        LiBattery1150 ?
        LiBattery1400 ?
```

(b) Feature Model

```
abstract xor Bluetooth
    Bluetooth20 ?
    Bluetooth21 ?
    Bluetooth40 ?

MobilePhone
    Connectivity
        'Bluetooth
        GSM
        LTE ?
            [LiBattery1400]
        WiFi ?
        USB ?
    xor Battery
        LiBattery1150 ?
        LiBattery1400 ?
```

(c) Structure via quotation

```
abstract MobilePhone
    Connectivity
        'Bluetooth
        GSM
        LTE ?
            [LiBattery1400]
        WiFi ?
        USB ?
    xor Battery
        LiBattery1150 ?
        LiBattery1400 ?

LowEndPhones : MobilePhone
    [no WiFi]

HighEndPhones : MobilePhone
    [USB && WiFi]
    QuadCPU ?
```

(d) Specialization

Figure 2: SPL modeling in Clafer

Modeling in Clafer. Based on the two variants, Bob identifies commonalities and variabilities and creates a feature model using Clafer as well (see Fig. 2b). The feature model has the same structure as the variant models but it is enriched with variability information. In Clafer, features are mandatory by default, whereas optional features are followed by question marks. Exclusive-or feature groups are preceeded by a keyword `xor`, the default group caridnality is $[0..*]$, which does not constrain the grouped features. Clafer is a minimalistic, yet expressive, language that unifies many concepts. For example, Clafer unifies features with feature groups, by allowing each feature to specify both feature cardinality and group cardinality; e.g., `Bluetooth` is a mandatory feature and an exclusive-or group. The unification of concepts greatly eases model evolution and eliminates permature committment since the modelers are not forced to choose modeling constructs and later switch them.

Constraint Language. Bob augments feature model with constraints originating from the problem domain. For example, the `LTE` module consumes more energy than the traditional `GSM` module, therefore it requires battery of higher capacity. The requirement is expressed by writing a con-

straint in the context of `LTE`. The constraint simply says that if `LTE` is selected, then also the battery `LiBattery1400` must be selected. Clafer offers an expressive constraint language that allows Bob to specify any first-order logic formulas, which reduces the amount of tacit knowledge. Similarly to features, constraints can be nested under features, which places them in the relevant context. Name resolution rules automatically locate features in the hierarchy when referring to features from constraints, thus eliminating the need for fully-qualified names. The two characteristics of Clafer have positive impact on model evolution.

Dealing with Scale. After several iterations of evolving the model, Bob decides to split it into smaller pieces. Standard feature models do not provide any mechanism for structuring models. As the models grow, they become difficult to comprehend and inconvenient to work with. Clafer provides model structuring via *quotation* (see Fig. 2c). First, Bob extracted the feature `Bluetooth` from the feature model and created an *abstract* feature of the same name. Next, in the original model he preceded `Bluetooth` by back-quote. The mechanism includes the abstract feature `Bluetooth` into the hierarchy via inheritance (we omit the details here). Abstract features define types, whereas non-abstract features specify instances. Abstract clafers are often used to enable reuse of both structure and constraints, thus supporting feature model reuse (illustrated later).

Configuration Completion and Conflict Resolution. When models become larger, it is very difficult to verify and validate them without automated tools. First, manual configuration of large feature models is time-consuming. Second, when feature models include constraints, finding configurations is error-prone. Bob uses ClaferIG for verification and validation. The tool offers reasoning on a wide range of structural models, in particular, over feature models. ClaferIG provides two main functionalities: 1) configuration completion, and 2) conflict detection. For a given (partially configured) feature model, it tries to find a complete configuration of the model. Bob inspects the generated configurations and checks whether the existing variants are covered by the model Fig. 2a. Bob elicits missing constraints when he sees incorrect configurations and restricts the model to prohibit them. He confirms that the model from Fig. 2b is correct. If the model was incorrect (either inconsistent, or inconsistent with the existing variants), ClaferIG presents the set of conflicting constraints, which Bob can remove.

Specialization and Extension. Bob knows that ACME divides the market into two segments: low-end phones and high-end phones. He needs to do the same with the feature model, i.e., create two specializations of the product line. Normally Bob would have to clone the feature model and modify the new clones. In Clafer, he can do it in another way: via inheritance. First, he marks the feature `MobilePhone` as abstract. Next, he creates two non-abstract features `LowEndPhones` and `HighEndPhones` that extend `MobilePhone`. He is able to attach constraints to the new models to partially configure them. In case of `LowEndPhones`, the phones have no `WiFi` feature. `HighEndPhones`, on the other hand, have both `USB` and `WiFi` modules, but also may have a quad-core CPU (represented by the feature `QuadCPU`). In that way, Bob can arrange models into multiple extension and specialization layers.

Collaboration. Bob is a part of small group of platform engineers who edit the feature model. They share the model with all other engineers at ACME in read-only mode. Such a setup is necessary for collaborative consensus building typically performed doing workshops. Bob uses ClaferWiki to control the model centrally. ClaferWiki provides an interface for editing and viewing the models as well as it supports *literate modeling*, the ability to interleave model fragments with rich text natural language descriptions. Thanks to the Git version control underlying the wiki, they can create branches of the wiki to support scoping future versions of the product line.

2.2 Product Engineering

Carol is a product engineer at ACME. Her job is to derive variants from the product line for customers. She defines variants by creating feature configurations.

Deriving correct variants. A correct variant is one whose defining feature configuration satisfies all problem and solution space constraints. Even if all constraints are known, manually verifying whether a feature configuration satisfies the constraints is tedious and error prone. Since Bob was able to express all constraints in the product-line feature model, ClaferIG can be used to verify the correctness of the configuration and help Carol fix the possible violations.

Deriving valid variants (1). A valid variant is one which satisfies the customer's requirements. For the requirements that can be directly mapped to product-line features, the features can simply be selected in the feature configuration. For the remaining features, ClaferIG can be used to generate possible completions of the feature configurations, which then Carol can review with the customer.

Reusing configurations. For large feature models, it is infeasible to always begin the configuration process from scratch. Typically, product engineers know which existing variants are closest to satisfying the customer's requirements and therefore they can take the feature configurations of these variants and modify them as needed. Unfortunately, modifying existing configurations may result in constraint violations—these can be detected using ClaferIG.

Carol can also prepare a set of *partial configurations* typical for different market segments (similarly to Fig. 2d) and use them as a starting point. ClaferIG can be used to generate possible configuration completions.

Partial configurations are essential for supporting *staged configuration*, whereby many product engineers configure only the parts of the feature model they are knowledgeable about. Consequently, selections made by one engineer can limit selections available to the others. Clafer naturally supports staged configuration.

Deriving valid variants (2). The customer may have requirements that do not directly map to product-line features but that are related to certain non-functional or emerging qualities of variants. In such cases, simply selecting features by trial and error may potentially be very time consuming and result in suboptimal variants from the point of view of the customer. To be able to satisfy these quality requirements, Carol works with Bob to add the necessary quality attributes to the product line. They extend the feature model with quality attributes, such as energy consumption or expected productivity gains as shown in Fig. 3. Each feature can individually contribute to the overall variant's quality. For instance, `Bluetooth40` adds 4 units to the overall variant's productivity, while it consumes 3 units of energy. Battery contributes negatively, since battery produces

```
abstract Feature                        WiFi : Feature ?
  productivity : integer                  [productivity = 20]
  consumption : integer                   [consumption = 10]
                                        USB : Feature ?
abstract MobilePhone                      [productivity = 25]
  Connectivity                            [consumption = −10]
    xor Bluetooth                     xor Battery
      Bluetooth21 : Feature ?           LiBattery1150 : Feature ?
        [productivity = 4]                [consumption = −75]
        [consumption = 2]               LiBattery1400 : Feature ?
      Bluetooth40 : Feature ?             [consumption = −90]
        [productivity = 8]            total_productivity = sum Feature.productivity
        [consumption = 3]             total_consumption = sum Feature.consumption
    GSM : Feature
      [productivity = 2]          optimalPhone : MobilePhone
      [consumption = 2]             [Wifi && no LTE]
    LTE : Feature ?
      [productivity = 16]         << max optimalPhone.total_productivity >>
      [consumption = 25]          << min optimalPhone.total_consumption >>
```

Figure 3: Quality attributes and optimization objectives.

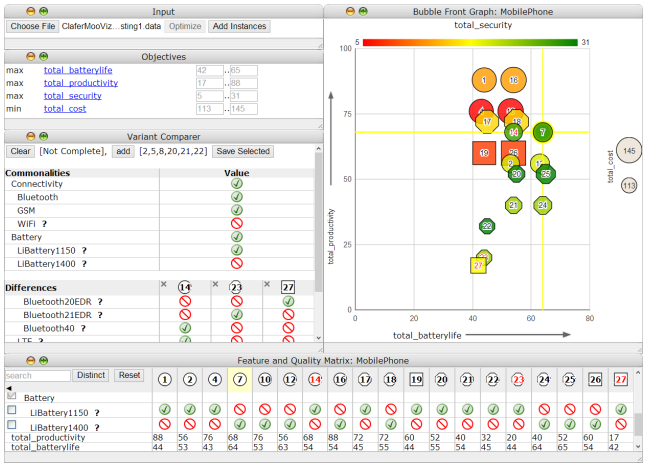

Figure 4: ClaferMoo Visualizer

rather than consumes energy. Finally, they add variant quality attributes `total_productivity` and `total_consumption`. While each definition uses a simple sum, more complex formulas can be used to model feature interactions. Despite knowing contributions of individual features, Carol cannot easily predict the overal quality of the resulting variant; this is due to complex dependencies among the features. Carol can, however, specify feature configurations and see the resulting variant quality values.

Multi-Objective Optimization. Carol is interested in optimal variants where the productivity is maximized while the energy consumption is minimized. ClaferMOO [7] can be used to solve this optimization problem: it computes a set of *Pareto-optimal* (i.e., non-dominated) variants given the above optimization criteria. Carol first prepares a partial configuration `optimalPhone`, which includes the feature `WiFi` and excludes `LTE` and specifies the two optimization goals (Fig. 3). Next she runs ClaferMOO, which computes all optimal completions of the partial configuration. The set of optimal variants can potentially be significantly smaller than the set of all variants. Carol can inspect the optimal variants with the customer, but choosing the most suitable ones requires a global view into the Pareto front.

Trade-off and feature impact analysis. Satisfying customer's requirements requires balancing the desired feature set and the acceptable levels of quality. Since the optimal variants are non-dominated, gains in one quality usually require sacrifices in other quality. In order to better help the customer, Carol wants to understand the effects of features on variant quality. For instance, `Bluetooth40`, compared to `Bluetooth21`, gives two times more productivity while consuming one unit of energy, and this trade-off is hard to see. Carol can use ClaferMoo Visualizer [8] [11] to visualize and explore the Pareto front (Fig. 4). Finally, Carol and her customer choose the best available variant.

3. CONCLUSION

We presented how Clafer Tools support a number of product-line engineering tasks. Clafer is an expressive modeling language that supports a rich set of concepts in a unified way, which eases model evolution. ClaferWiki is a web-based, collaborative IDE for literate modeling using Clafer. Clafer is supported by reasoners, which help with model verification, validation, and configuration (ClaferIG), and optimization (ClaferMOO). Finally, ClaferMOO Visualizer can be used to visualize and explore the Pareto front allowing product engineers to discover most suitable variants given customer's requirements.

Future Work. We are currently working on a web-based Clafer Configurator to support manual configuration and staged configuration, which can be used together with optimization and the visualizer to seamlessly support "deriving valid variants (1) and (2)" (see Sect. 2.2). We are also planning experimental evaluation of the tools with professional product-line stakeholders.

Acknowledgments. We thank Andrzej Wąsowski and Derek Rayside for their feedback. We thank Chris Walker for implementing Clafer Wiki and Neil Vincent Redman for implementing many extensions to the ClaferMooVisualizer.

4. REFERENCES

[1] Alloy. http://alloy.mit.edu.
[2] Choco. http://www.emn.fr/z-info/choco-solver.
[3] Clafer Compiler. https://github.com/gsdlab/clafer.
[4] Clafer Homepage. http://clafer.org.
[5] Clafer Wiki. https://github.com/gsdlab/claferWiki.
[6] ClaferIG. https://github.com/gsdlab/claferIG.
[7] ClaferMOO. https://github.com/gsdlab/claferMoo.
[8] ClaferMoo Visualizer. https://github.com/gsdlab/claferMooVisualizer.
[9] Common Variability Language revised submission, 2012. http://www.omgwiki.org/variability/doku.php#cvl_revised_submission.
[10] K. Bąk, K. Czarnecki, and A. Wąsowski. Feature and meta-models in clafer: Mixed, specialized and coupled. In *International Conference on Software Language Engineering*, pages 291–301, 2010.
[11] A. Murashkin, M. Antkiewicz, D. Rayside, and K. Czarnecki. Visualization and exploration of optimal variants in product line engineering. Submitted to SPLC'13.
[12] R. Olaechea, S. Stewart, K. Czarnecki, and D. Rayside. Modeling and multi-objective optimization of quality attributes in variability-rich software. In *NFPinDSML*, 2012.

5. APPENDIX: DEMO SCRIPT

In the demonstration we will follow the story (see Sect. 2) presented in this paper and show all tools using a larger version of the presented example. We will show actual working tools, not just slides. In particular, we will do the following:

1. Briefly show the website http://clafer.org, which is an entry point into Clafer and contains pointers to all related materials. Point to tool distribution and download and installation instructions.
2. Introduce the context: ACME company, moving to product lines from clone & own.
3. Introduce the three personas: product-line owner and engineer, and product engineer.
4. Open the ClaferWiki (example screenshot of a wiki page shown on the right) and create two pages describing AndroidPhone1 and 2. Explain how the wiki uses the compiler, literate modeling, conflict resolution support, etc.
5. Derive the feature model from the two configurations, illustrate constraints and quotation mechanism.
6. Create partial configurations and use ClaferIG to check whether the configurations are correct wrt. the feature model. Introduce a conflict on purpose to illustrate conflict resolution.
7. Create two specializations, LowEndPhones and HighEndPhones.
8. Simulate collaboration on the model using wiki, show version control and conflict resolution (time permitting).
9. Create variant configurations, show constraint propagation and configuration completion using ClaferIG or ClaferConfigurator.
10. Show partial configurations and staged configuration.
11. Add quality to the model and run multi-objective optimization by hand (execute ClaferMOO). Briefly inspect the results to show how hard it is to work with Pareto front data.
12. Finally, open ClaferMOO Visualizer (see Fig. 5 below) and show analysis using Bubble Front Graph view (clustering, distribution), and filtering by feature, by target quality range using Feature and Quality Matrix, tradeoff analysis using Variant Comparer view.
13. Conclude the demonstration.
14. Q & A.

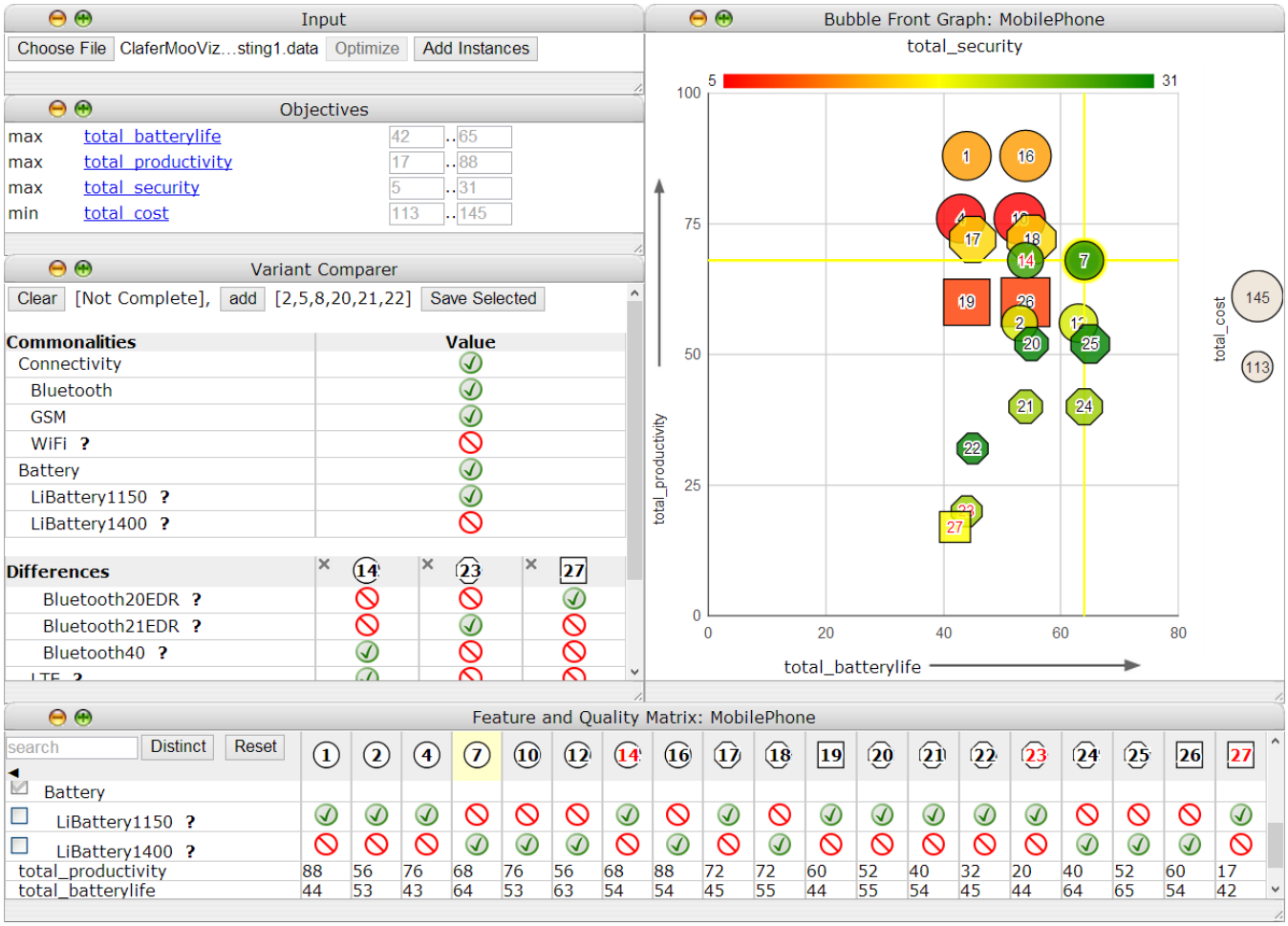

Figure 5: ClaferMoo Visualizer—Large View