The Enlightened, Post-Modern Product Model

Maxim Noah Khailo

February 9, 2016

1 Purpose

There is both a universal objective reality and our subjective interpretations of it. The post-modern world is one of flux and uncertainty which can only collapse into the shadow of objectivity when illuminated.

Below is a description of a product model, nay, a content model that both provides variant in the face of contextualization, but has its objective reality and definition.

Please read the companion paper "Everything is Search" after this one to realize the full potential of the approach described below.

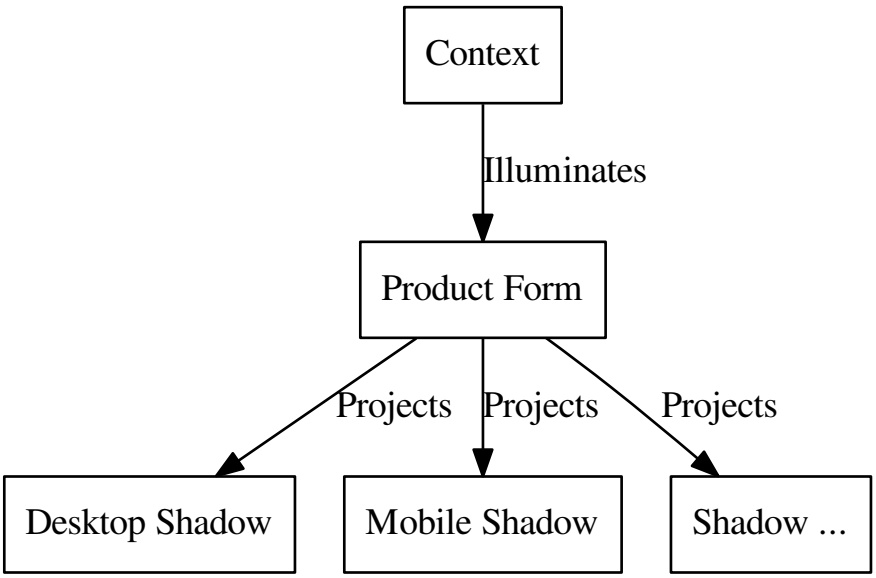
2 Forces

We have to balance several problems.

- Provide the most appropriate information at the right time.
- Semantically structure information into a computable form.
- Provide a manageable way to handle variant.

3 The Form and Shadow

The model is split into two concepts, the *Form* and the *Shadow*. A product takes on one form but can have many shadows. Shadows are created when a *Context* illuminates the *Form*.



3.1 The Form

The form is a combination of schema and data. The model is composed of dynamic attributes which have a type. Each attribute is a map of values that have names. The Form may look like this, represented as JSON.

```
//Product Form
{
    id: 3
    name: {
        type: "string":
        x: "Red Shoe",
        y: "Big Red Shoe",
    },
    description: {
        type: "string"
        x: "Buy Me",
        y: "Purchase Me",
    },
    image: {
        type: "uri"
        x: "http://a",
        y: "http://b",
        z: "http://c",
    }
}
```

3.2 The Shadow

The Product Shadow is what is used when displaying a product to a customer. At any point in time, the customer has a Context. The Product Shadow is the shape you get when you illuminate the Form from a particular vantage point called the Context. Here is an example of a Shadow represented as JSON.

```
//Product Shadow for English Desktop
{
    context: { modality: "desktop", language: "english" }
    product: 3,
    name: "x",
    description: "x",
    image: "x",
}
```

And here is another, but for mobile...

```
//Product Shadow for English Mobile
{
    context: { modality: "mobile", language: "english" }
    product: 3,
    name: "y",
    description: "y",
    image: "z",
}
```

Each attribute in the Shadow corresponds to an attribute in the Form. It selects the value from the Form via it's name. For example, the 'x' value from the 'name' attribute in the Form..

4 The Context

Each *Shadow* is a representation of the product *Form* given a *Context*. Each user will have a Context when they interact with our system. The Context is used to select which shadows to display for a particular product.

If we assign statistics to each of the Shadow's attributes, as described in the companion paper "Everything is Search", this model will provide a solid foundation for algorithms that could modify shadows or create new ones based on data and statistics.

5 Mapping Variants to SKUs

The big question here is how do we handle variants. How do we map a product variant to a SKU? The Product Form can have variants which represent trees where the leaf nodes are the skus.

```
{
    variants1 : {
        color : {
            red : "SKU-RED1",
            green : "SKU-GREEN2",
        }
    }
    variants2 : {
        color : {
            purple : "SKU-BLUE3",
            orange : "SKU-ORGAN3",
        }
    }
    ...
}
```

The benefit of this model is we can represent arbitrarily complex variants by simply assigning the leaves of the tree with SKU values. If the leaf nodes are empty, we can have an algorithm that can walk the tree and assign meaningful SKU numbers.

Also, because of our Shadow model, we can handle cases such as "Only purple and orange are available in Germany." if the user is in a German context.

Image a product has the following variants

```
//Product Form With Variants
{
    Germany : {
        color : {
            red : "SKU-RED1",
            green : "SKU-GREEN2",
        }
    }
    USA : {
        color : {
            purple : "SKU-BLUE3",
```

```
            orange : "SKU-ORGAN3",
        }
    }
    ...
}
```

We can map the variant in the product form and shadow like so.

```
//Product Form with Variant Attributes
{
    id: 3
    name: {
        type: "string":
        USA : {
            color : {
                purple: "Purple Shoe",
                orange: "Sun Orange Shoe",
            }
        }
        German : { ... }
    }
    ...
}

//Product Shadow Using "USA" variant
{
    context: { language: "english", region: "USA" }
    product: 3,
    name: "usa",
    description: "english"
}
```

The benefit of this approach is that not all attributes have to change based on variants But it allows some to change. For example, the description attribute above doesn't change based on variant, but the name attribute does.

Also we can reuse different variants and attributes in different contexts.

6 Models

6.1 Context

A Context has an identifier, name, and attributes.

```
case class Context(
    id: Id,
    name: String,
    attributes: Json)
```

Where the attributes can be arbitrarily specific, but should be of a flat key/value nature.

```
{
    modality: "x",
    language: "y"
}
```

6.2 ProductForm

The ProductForm represents a product as described above.

```
case class ProductForm(
    id: Id,
    attributes: Json,
    variants: Json)
```

The attributes should be of the form

```
{
    attribute1: {
        type: "abc",
        val1 : "x",
        val2 : "y",
        ...
    }
    attribute2: {
        ...
    }
}
```

6.3 ProductShadow

The ProductShadow represents a Shadow of a product as described above.

```
case class ProductShadow(
    id: Id,
    context: Id,
    productForm: Id,
    attributes: Json)
```

Where the attributes correspond to attributes in the product form, selected by name.

```
{
    attribute1: "x",
    attribute2: "...",
    ...
}
```

6.4 Tiny MVP

For Tiny MVP we will seed only 1 default context. The product form will have one "default" variant and minimal set of attributes.

```
//MVP Product Form
{
    title: {
        type: "string",
        ...
    }
    description: {
        type: "string",
        ...
    }
    images: {
        type: "images"
        ...
    }
    price: {
        type: "price"
        ...
    }
}
```

We provide the following Admin endpoints

- GET /api/v1/contexts
- GET /api/v1/products/x
- GET /api/v1/product-shadows/x
- POST /api/v1/product-shadows/
- PUT /api/v1/product-shadows/x
- GET /api/v1/product-forms/x
- POST /api/v1/product-forms/
- PUT /api/v1/product-forms/x

The "/api/v1/products/" route will implement what eventually the storefront JSON will look like.

We can index everything in ES using context like this.

- /api/search/products-[context]/
- /api/search/product-shadows-[context]/
- /api/search/product-forms-[context]/