ORIGINAL ARTICLES
SPECIAL ISSUE

EVOKEDB: A KANSEI DATABASE SYSTEM FOR AUTOMATED SURVEYS

Andreas NORDGREN and Hideki AOYAMA

Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, 223-8522, Japan

Abstract: *Kansei* Engineering methods are difficult to implement into the design process due to a lack of specialized tools in the public domain. Surveys are costly to conduct, and the use and analysis of data through simplified models may not explore the deeper meaning of how *Kansei* relates to complex artifacts. EvokeDB, a *Kansei* database system for automated surveys, was created to address these issues. It implements a new model of *Kansei*, where realms and senses are considered important entities, into a flexible and usable web-application based on the Ruby on Rails framework. A database structure is presented, along with a description of how tables are mapped to programming objects in order to create a rich set of relationships that can be used to explore *Kansei*. The systems' open-source components, and the data flow between them, are explained. A simulated project exemplifies the steps needed to set-up and launch an un-supervised, on-line survey to multiple, remote users.

Keywords: *Kansei* Database, *Kansei* Measuring, Survey, Ruby on Rails, AJAX

1. INTRODUCTION

Kansei Engineering (henceforth referred to as *KE*) is gaining momentum as a force to be reckoned with in product development and many other areas involving human emotion. Innovation has always been a key to success when it comes to creating products and services that stand out in the marketplace, but in many cases nowadays, it is no longer enough to be the best performer -customers are often quantitatively satisfied and are not making purchasing decisions based purely on logic, but rather on how a product feels; it must appeal to them on an emotional level. Advances in technology, manufacturing and marketing are still essential, of course, but companies must also understand how to use the channel of *Kansei*-communication in order to survive and excel in a competitive environment.

However, there is a lot of confusion on how to implement *KE*-methods in the design process, and rightfully so; *Kansei* is an internal process that is inherently difficult to come to terms with and analyze with any degree of confidence. There are some specialized tools available [16], but most methods used today stem from generic algorithms in statistical analysis which require extensive training that most designers and product developers lack, or are unwilling to learn. As a set of methods to understand a persons' emotional response to an artifact, *KE* suffers from a lack of clarity on how to efficiently and accurately collect and interpret a set of data. This is a bold statement to make, but in my opinion we need to present better guidelines on how to describe artifacts and implement methods, in order for users without specialized training to carry out successful

Kansei-studies. Consider complex artifacts, such as most gadgets that cater for our new digital lifestyle – they have a number of stimuli in different realms that may all lead to associations which feed into our *Kansei*. For example, a portable media-player has stimuli such as shape, material, color, features and services attached to it, and yet there seems to be little consideration taken to how these stimuli are related, and by which senses they are explored, in many *KE*-studies. I argue that there lies a great danger in overly simplified models for complex artifacts – even a properly performed analysis of a simplified model may yield misleading results, or simply fail to recognize the most important stimulus for a given *Kansei*-word.

Thus, I think we can all agree that *KE* is powerful in skilled hands, but it does presents the novice with a rather difficult set of methodologies that must be mastered in order to implement it successfully. Even a trivial task, such as data collection, may challenge many potential users with a hurdle they cannot easily overcome. Data is a cornerstone of every *Kansei*-study, and surveys to collect it are essential but often tedious to create and perform. Consequently, there is definitely a need for a tool to automate surveys and guide novices through a study.

The second point I want to make in this introduction is the problem of the reliability of the data itself; is it still valid, can we trust our results? Currently, it takes resources of time and expertise to collect *Kansei* data; data which is then regarded as a beautiful box of sushi very tasty, but easily spoiled unless you consume it at once. We must find a more robust way to gather, analyze, and store *Kansei* data to make it more resilient to changes in trends. A red dress may not be “fashionable” every year, but it still a red dress, that is explored by sight or touch, and which stimuli

may lead to the same associations in a subject no matter what the current trends dictate. I believe associations are very important to *Kansei* – we understand and feel through connections to past experiences, our whole being depends on the rich set of links that our nervous system is made up by – neurons fire and we *relate*. There is meaning in an object, a color, or a scent, and how they are combined and fused together by our sensory organs to be processed internally. To find this meaning is the holy grail of *KE*, but the path to *Kansei* enlightenment is indeed one of many pitfalls – we need a companion to aid in our endeavor. But what type of system could offer this type of aid, in terms of accessibility, clarity and guidance? A system for *Kansei*-explorations must, at some level, possess the ability to relate like us, in order to shed some light on the meaning within the data. Therefore, in this paper I shall present to you a database-system that maps the relationships between *Kansei*-words, stimuli, senses, realms, artifacts and subjects. I believe this can present users with a more detailed and clarified understanding of *Kansei*, where costly data is never wasted, and new data is easily collected. The system is named EvokeDB (Evaluation and Verification of Objects in *Kansei* Engineering Database), and it is offered as a tool to make collection, analysis and storage of *Kansei* data accessible to non-experts in any field, via automated surveys. My contribution is twofold:

- First, a new model of *Kansei* for complex artifacts is suggested, where realms and senses are considered important entities for exploring *Kansei*.
- Second, I explain how to implement and use that model in a system of open-source components, with functions to create and launch on-line surveys. Specifically, I discuss the merits of the Ruby on Rails framework with AJAX for this type of application, and how they can be used to create a system that is responsive, flexible, easy to implement, and usable throughout a project as a source of *Kansei* interpretation.

2. AIM

The introduction pointed out a few areas of *KE* where I wish to make a contribution. The primary aim of this research was to create a tool that would be able to support non-experts in any field with means to collect, evaluate, and store *Kansei* data for artifacts of any complexity. This aim presented me with a number of requirements. First of all, “non-experts in any field” is a very broad audience with various requirements, so the system needed to be flexible enough to allow a user to make alterations to make it fit the needs of a specific project, whether it is about a new car, a theme-park ride, or cup-noodles. This flexibil-

ity should, however, not come at the expense of increased complexity and less usability, as this would discourage users without extensive knowledge of programming computers. This lead me to consider a web-application which could be developed with open-source components, distributed freely and deployed on different platforms. A web-application would also, once properly set up, allow remote users to interact with the system without installing specialized software since most modern computers are already equipped with web-browsing capabilities. This would, for instance, enable teams of designers to collaborate, share data and use the system even though they are geographically separated. Surveys could also be launched on-line, a feature which makes it easier to approach a large number of subjects for a low cost. However, even though web-content can be created quite easily by many applications, it does in general take skills and time to create a working survey backed by a database. This system should therefore have functions to automate many of the tedious parts of creating and conducting surveys, thereby making them more accessible as a tool in *KE*.

Furthermore, I wanted to implement a robust structure where data from many different projects could be stored and shared. This structure should be based on a model of *Kansei* that considers the complexity of products and services, how relationships between fundamental characteristics affect *Kansei*, so that designers can interpret data from many different viewpoints throughout the design process and gain a deeper understanding of the meaning of *Kansei*. It should be easy to run surveys, or consult the system on a word, stimulus or sense, and receive some feedback and guidance based on the data collected. My focus was therefore initially only on automated surveys, but I also wanted to be able to augment the functionality of the system in the future to include features such as automated generation of *Kansei* languages and more advanced analytical methods. The model in Section 3 was defined with these requirements in mind, although this paper will mainly describe EvokeDB from the perspective of data-collection.

3. FOUNDATION

3.1 A new model for *Kansei*

In the previous sections I have mentioned a few entities that deserve a more in-depth description. A fundamental problem that I wanted to approach with this system is how to explore the meaning of artifacts, that is, to understand how *Kansei* is evoked, and to find out what stimuli or events lead to the formation of an emotional response in a person. What exactly are the main entities we are dealing with? An artifact (product, service or event) is explored by

a person through senses (sight, hearing, touch, smell and taste) and processed internally to form a logical understanding, as well as an emotional response, to the artifact. However, the emotional response depends on the environment and context the artifact is presented within, and to a higher or lesser degree, the parts it constitutes of. Most physical objects have a number of different properties or characteristics which may lead to associations which in turn feed into a persons' *Kansei*, and the same is probably true, although less obvious, for services as well. I propose an approach where an artifact can be viewed as a whole, but also as an entity made up of parts. To enable this, I introduce the concept of *realms*.

A realm is a grouping or set of similar characteristics of an artifact. For example, a car could have realms of color, shape, sound, material, or even less obvious realms such as brand-image and showroom, freely defined by the designer. Each realm contains stimuli, such as { red, green, blue, .. } for color, or { titanium, aluminum, leather, .. } for material. Inclusion of these entities into the model will allow for a more detailed understanding of how *Kansei* relates to a certain sense, realm or stimulus, not just the artifact itself. This is an extended use of *Kansei* data that will allow for sharing between projects in various domains, as the data relates to fundamental characteristics (stimuli) rather than only referring to an artifact in a narrow sense. Nagamachi [1] explained how the expression "domain" could be used to structure products based on their type, but this should not be confused with realms. Realms are used to structure properties of an artifact, not to describe its type. In my opinion the domain is mainly involved in the logical understanding of artifacts, that is, how they are grouped in a context of other artifacts. That is not to say it has no bearings for *Kansei*; in fact, it is very important, but it should be supplemented by realms to provide means to explore how fundamental stimuli, like color or sounds, can affect *Kansei*. The model in Figure 1 depicts entities,

in a context provided by a domain, connected with arrows to symbolize relationships. The entities were modeled as objects, along with all their attributes and relationships, to give a strong, yet flexible, foundation to the EvokeDB *Kansei* database-system.

This model allows a user to query the database in a more focused manner, or even to determine how a certain sense contributes to *Kansei*. In many cases a designer wants to "tweak" the parameters of the artifact to elicit just the right *Kansei*, but to do so requires a high sensibility to the problem at hand – for instance, changes in one parameter may produce the desired response for a *Kansei*-word "sporty", but also bring with it an undesired change in "robust". The relationships in this new model can guide a designer to the best realm or sense to work within, based on a specific *Kansei*-word.

Another problem that will also benefit from this model is automated generation of *Kansei* languages. In most studies it is important to choose a set of words (a vocabulary, or what I will refer to as *Kansei* language), which can adequately describe an artifact and span the space of emotional responses of a group of subjects.

There are few guidelines to help a designer select these words -a standard approach is to gather all words that are even remotely related by browsing magazines or manuals, interviewing people, and so on. This leads to a large set of words which can be reduced by removing overlap of meaning by performing a factor analysis on data from a pilot study. However, this is again very costly and time-consuming, and leaves a lot to be desired in terms of clarity and guidance. Through this new model, it is possible to query the database on which words are most strongly related to an artifact, realm, stimuli, sense, or combination thereof, and propose a *Kansei* language based on the results. If 'red' and 'aluminum' will be characteristics of a new product, then by all means, let's use that to query the database on data from past surveys and find out which words correlate most strongly. As long as a designer can define which realms, stimuli and senses will be used in a new project, this feature of EvokeDB will at least give a good starting point, a base vocabulary, for the designer to consider. Do note that the performance of EvokeDB in this regard will depend on the amount of data it has -the more it is used, the more meaningful *Kansei*-languages it will be able to produce. Fortunately, EvokeDB makes storage of survey entries for numerous projects easy, and this data will also be actively used by the system as a source for *Kansei* interpretation, thereby providing use for old data as well.

Figure 1: *Kansei* model

3.2 Definition of objects

The objects in Figure 1 lays the foundation for EvokeDB, but the system also implements a number of additional

objects (described below) that was needed for a richer set of functions.

- **Artifact** – a product, service, event or other entity which will undergo a *Kansei*-evaluation. An artifact can be divided into many realms, like color, shape, sound and material.
- **Realm** – a set of unique characteristics of an artifact. The realms, and the stimuli which belong to the realms, will be used to describe an artifact in detail.
- **Sense** – a human sense associated with a sensory organ.
- **Subject** – a person taking part of a survey. There are many attributes that can be used to describe a subject in detail, such as age, gender, nationality, income level, education, hobbies and so forth.
- **Survey** – an evaluation on a set of artifacts, on a set of words (language), by a set of subjects. Each survey belongs to a project, and has one language. Different surveys may use different input methods and ranges, so to be able to compare and use data from many surveys, the data must be normalized and therefore include attributes of min_value and max_value.
- **Language** – a set of words used to span a *Kansei* space. A language can be used in many surveys.
- **Word** – a word, commonly an adjective, that describes a certain emotion. A word belongs to one or many languages.
- **Project** – helps a designer to organize and structure surveys. Each project can have many artifacts and many surveys.

- **Note** – a note is a short piece of text attached to a word, so that a designer, or subject in a survey, can write down an idea, question or other remark relating to the word, and save it in the system.
- **SurveyEntry** – this is where the bulk of the collected data is kept. Every survey will yield a large number of entries where subjects enter a value on how each artifact is related to a specific word.

Once these entities had been defined, they could be represented as tables in the database, as described in the next section.

3.3 Database structure

Figure 1 displayed a model of four objects whose relationships are very important for EvokeDB, and Section 2 added an additional six objects used to augment this model in order to provide functions to automate surveys, analyze data and propose *Kansei* languages.

However, EvokeDB is a web-application that relies on a database to store its objects, and therefore the database-structure in Table 1 was constructed with ten different tables along with attributes to represent the objects from Section 3.2. This schema is needed for the functionality of EvokeDB, but it is by no means set in stone -new attributes can easily be added, thus giving a designer more control over the system and how it is configured. This type of flexibility is usually not easy to implement without re-writing parts of the application, but we cannot expect designers to perform such operations. Changes to attributes should be recognized and handled by the application

Table 1: Database structure

artifacts	realms	senses	subjects	surveys
id	id	id	id	id
name	name	name	name	name
description	description	description	description	description
image_url	created_on	sensory_organ	date_of_birth	min_value
version	updated_on	created_on	gender	max_value
created_on		updated_on	nationality	language_id
updated_on			annual_income	project_id
			created_on	created_on
			updated_on	updated_on
languages	words	projects	notes	survey_entries
id	id	id	id	id
name	name	name	author	artifact_id
description	description	description	content	subject_id
created_on	created_on	created_on	word_id	survey_id
updated_on	updated_on	updated_on	created_on	word_id
			updated_on	value
				created_on

automatically. This problem presented me with a serious challenge. Up until this point I have described EvokeDB as a model with ten objects, implemented as tables with fields in a database. However, more advanced features, such as automated surveys are not possible unless I can use these objects in a programming environment, and work with them using an object-oriented language. This requires a translation between the object-oriented layer and the database layer, that is capable of preserving the properties of the objects, as well as their relationships. Simple relationships between objects can be set up through a join-table containing only the foreign keys of the objects being joined, but this is difficult to maintain when relationships increase or become more complex, and it also requires complex SQL-queries to retrieve objects. Thus, an object-oriented programming language coupled to an efficient object-relational mapping for the database-tables was needed to bring functionality to EvokeDB. The Ruby on Rails framework provided a solution which I will describe in section 4.1.

3.4 On data retrieval and semantic search-algorithms

I will explain how user requests are handled by the system in the coming sections, but I find it suitable to place EvokeDB within a context at this point. There has been a few other *Kansei* Database-systems presented prior to my research, where semantic associative search-algorithms are possible through careful definition of meta-data that describe contents (photos, recordings, videos) in a multimedia database, in terms of *Kansei* [6,7]. EvokeDB is not a multimedia database, although it contains some multimedia content used to present artifacts. Furthermore, it does *not* use meta-data to describe its contents; instead meaning is found through the rich set of relationships (Section 4.1.2), which can be explored easily through its interface. This is a simpler approach, but one that fits well to the design intent for this system.

4. APPLICATION COMPONENTS

4.1 Ruby on Rails framework (RoR)

Ruby is a modern object-oriented programming language created by Yukihiro Matsumoto in the mid 1990s [2]. It has a very intuitive and clean syntax with similarities to both Perl and Python, with a design philosophy that focuses on ease-of-use for the programmer.

Rails is younger yet – it was first developed in 2004 by David Heinemeier Hansson as a tool to help develop an on-line project-management application, and it has since then been extracted and released to the public [3]. Active development has turned Rails into a very popular framework for building web-applications, due to the speed and

ease of building complex application.

There are many different programming languages one could use to make a *Kansei* database-system, but I advocate the Ruby on Rails approach as a very suitable candidate for this type of system, by reasons presented in the following sections.

4.1.1 Rails MVC-architecture

A Model-View-Controller architecture gives an application structure by separating its data model, user interface, and control logic into three distinct components that can easily be manipulated and maintained by the programmer.

In Rails, the *Model* consists of classes representing tables in a database. These classes inherit from the *ActiveRecord*-class, which provides an object-relational mapping and thereby methods to perform CRUD-operations (Create, Retrieve, Update and Delete). Through this mapping it becomes very easy to manipulate objects as Rails will handle communication with the database and relieve the programmer, or user of the system, of making complex SQL-queries. The Model will also contain code to describe the relationships between objects, whether they are one-to-one, one-to-many, or many-to-many. Furthermore, this inherited object-relational mapping works without explicit configuration, as long as the tables in the database are named according to a specified convention of pluralism (for instance, object Artifact will map to table artifacts).

Convention over configuration is a Rails mantra, and in most cases the constraints of following conventions are well worth the ease-of-use this philosophy brings.

The *View* renders the Model into a form more suitable for interaction, and is used to display logic and communicate with the users. Each method in the Controller is connected to a View, which usually consists of a code-fragment in HTML (Hyper-Text Markup Language) with styling provided by CSS (Cascading Style-Sheet).

Finally, the *Controller* classes consist of code to handle user interaction and call the appropriate logic to respond to requests. Code for many common tasks is automatically generated by Rails scaffolding-techniques, which in essence creates a skeleton of an application by one simple command. This scaffold is not perfect by any means – it should be improved, replaced or augmented bit by bit – but it will still give a lot of functionality to an application in a very short amount of time and relieve the programmer of many tedious tasks.

4.1.2 Relationships in Rails

Section 3.3 described the problem of handling communication between objects and tables in the database in an efficient manner that would not require construction

of complex SQL-queries. Luckily, Rails provides an object-relational mapping through the *ActiveRecord*-class, which can be used to set up relationships between objects with a few lines of code that will automate queries between the application and database. As an example, let's look at the class-definition in the model for a Survey-object:

```
class Survey < ActiveRecord::Base
  belongs_to :language
  belongs_to :project
  has_and_belongs_to_many :senses
  has_many :survey_entries, :dependant => true
  has_many :artifacts, :through => :survey_entries
  has_many :words, :through => :survey_entries
  has_many :subjects, :through => :survey_entries
end
```

This declaration states that the Survey will inherit from the *ActiveRecord*-class and thereby obtain a number of methods to communicate with the underlying database. Furthermore, there are a number of relationships that will automatically link to the appropriate tables in the database and add functionality to the model. For example, a survey belongs to a project, and a project can have many different surveys. A survey will be conducted with a specific language (set of words), but a language can also be used in many different surveys. These two relationships are called one-to-many, and are used frequently in EvokeDB. There are a number of many-to-many relationships as well, easily identified by the *has_and_belongs_to_many* and *has_many :through* -declarations. *ActiveRecord* has three basic types of relationships between tables, as described in the sections below.

4.1.2.1 One-to-one relationship

A one-to-one relationship is created by using a foreign key in one row in one table to reference another row in another table, thus creating a link between the two. The corresponding models will have declarations of *belongs_to* and *has_one* inserted to make Rails aware of this relationship.

4.1.2.2 One-to-many relationship

This relationship is declared by a pair of *belongs_to* and *has_many* -declarations entered into the respective models of the child-and parent-object. Rails will be able to link to the right table through the foreign key (for Surveys this is *language_id* and *project_id*, see Table 1).

4.1.2.3 Many-to-many relationship

For this type of relationship there are two approaches, each with its own merits and demerits.

A simple relationship can be set up through a join table (*senses_surveys*) of foreign keys (*sense_id*, *survey_id*), as explained earlier in section 3.3. Each model will have a declaration of *has_and_belongs_to_many* (*habtm*) to tell Rails about this relationship. With this type of relationship it is easy to link a survey to a set of senses, and likewise, a sense to a set of surveys, but it is not possible to include more objects or give this link any attributes; it is and will never be more than a link between two objects.

Relationships between multiple objects (N-way joins) and richer relationships where the link can carry attributes require the *has_many :through* -construct, where the join-table will actually be implemented as an object with its own attributes. The *survey_entries* is such a join-table. It is implemented as the *SurveyEntry*-object by Rails to link words, artifacts, subjects and surveys to a value (see Table 1). This table will contain the bulk of the data in EvokeDB, and the *SurveyEntry*-object will often be called upon to deliver data for an analysis.

In the class-definition in section 4.1.2 we can see that the *survey_entries* has an option declared as *dependant => true*. This ensures that if a survey, subject or artifact is deleted, then the corresponding *survey_entry* will also be deleted. Rails makes this type of database-maintenance very easy to implement.

4.2 AJAX and on-line surveys

AJAX (Asynchronous JavaScript and XML) is yet another acronym that has received a lot of attention in the web-developing community during the last year. It is not a technology in itself, but rather an umbrella term representing a group of technologies and how they are used together.

XMLHttpRequest is one of the most important components of AJAX as it allows an object to exchange data asynchronously with a web-server through an independent connection channel between a web-applications' client-side and server-side [4]. This solves a serious drawback with web-applications, which up until now have left subjects in on-line, unsupervised surveys less supported in terms of how the system communicates with them. A traditional desktop application is very responsive – if a user clicks a menu or makes an input to the application, the result is usually displayed at once. However, the Web has been, and still is, a far more constrained medium in terms of how we interact with it.

Basically, we are presented with content in a hypertext-system which we navigate through by clicking on links that tells the server to serve us new information; it is an interaction model of constant call-and-response that keeps users waiting and thus lacks the responsiveness of a desktop application.

By implementing AJAX it is possible to create a user-interface which communicates with the user in “real-time” without the constant need to submit data and reload the whole page to get a response back from the server. For example, a common task in a survey to collect *Kansei*-data is to present a subject with an artifact, which is then evaluated by the semantic differential method [5] on a set of words (or bipolar pairs of words) representing *Kansei*. This evaluation is usually performed by making a correlation of how strongly the artifact relates to the word on a range from low to high, either by entering a number, clicking on a radio-button, or adjusting the value on a slider. In a supervised survey a subject can always ask the interviewer about the input method or the meaning of the range, but this is not possible for large-scale, on-line surveys. It is therefore important to give clear instructions and explain the meaning of a range before the survey, as well as provide feedback *during* it. With AJAX, the system can respond to a users’ input directly through a number of predefined events, and thereby provide support and make sure that the meaning of an input is properly understood. A traditional approach on the other hand, is bound by the limitations of the hypertext interaction model and can not cater for this type of feedback during an evaluation.

4.3 Apache web-server

Apache [13] is an open-source HTTP-server which *EvokeDB* will run on. It is available for a number of different platforms at apache.org. A number of other HTTP-servers, both commercial and open-source, are also available (Rails even comes with its own), so this choice would probably depend on the current IT-infrastructure of a company wishing to implement *EvokeDB*.

4.4 Mysql database

There are many databases that would fit the bill for this application and I am certainly not biased regarding which is the best one. I chose *MySQL* [14] since I am familiar with it and because it is offered under a GNU General Public License (GPL). Download available at dev.mysql.com.

4.5 PhpMyAdmin database administration tool

PhpMyAdmin [15] is a tool to handle administration of *MySQL* via an intuitive Web-interface. Again, it is not the only tool available for this task, and others may be just as suitable. For the more experienced user it is of course also possible to access the database directly through a command-line interface. However, a graphical user-interface will certainly be a helpful tool for the novice. Direct access to the database is mainly for management, addition of attributes to models, or retrieval of data through SQL-

queries, but is in general not required for most projects once the database structure has been set up.

PhpMyAdmin is available in 50 languages and offered under a GNU General Public License (GPL) at phpmyadmin.net.

4.6 Putting it all together

The number of components required might seem daunting at first, but they are in general easy to install and configure, and gives a user a high level of control over the system. The first step is to install a web-server capable of running RoR. After this a database with administration-interface can be installed. These are the principal components needed for the *EvokDB*-system. Once they are working, the database-structure is created by running a given SQL-query. Finally, the code for the system can be uploaded to the web-server. As explained in Section 4.1.1 the system uses an MVC-architecture, so the code is organized in a tree of folders and files according to RoR’s separation of models, views and controllers. Configuration of the database-location completes the system and from this point on all the objects are available, the user can start creating and running surveys immediately. Figure 2 gives an overview of the components and the flow of requests between them. One positive aspect of this system is that it can run locally, or within the IT-infrastructure of a company, or exposed to the whole world through the Internet. This gives many options on how to conduct a survey.

The top of the figure displays two different interfaces – *phpMyAdmin* and *EvokeDB*. This is the client-side of the application. First of all there is a flow for direct access to the database through *phpMyAdmin* → *Apache* →

Figure 2: System components and flow

MySQL. This gives a user access to administrate the database and configure EvokeDB; entering new attributes of objects is one example of possible configurations. Secondly, there is the flow starting through the EvokeDB-interface. This flow will also communicate with the database but via the objects defined by Ruby on Rails. Any requests will go through the appropriate objects' *model* and *controller*, which in turn will query the database and present the user with an output (formatted by HTML, CSS, and AJAX) through a *view*. The interactions of a user (designer) and those of subjects taking a survey are of course not the same even though they use the same objects, but this is easily handled by defining different controllers for them. This is yet another example of how separation of data model and control logic makes the system flexible and easy to update.

5. EVOKEDB IN ACTION

In this section I will make a simulated run through the steps of the application to exemplify the flow and use of EvokeDB. Let us pretend we are an automaker in the premium segment, and that we want to explore *Kansei* of luxury cars in order to ensure the success of our upcoming model. Even as a top-tier company, we are on a limited budget and therefore we need to invest our resources wisely, where they count the most. However, we have an overwhelming amount of parameters to consider – should we put our effort into careful selection of the exterior shape, interior materials, colors, perform extensive sound engineering, or just simply put on a very nice set of rims? We need some guidance on how different stimuli will affect a customers feelings towards our product, so we decide to make a survey to collect data.

The seven steps in next section exemplifies the set-up of EvokeDB, and some considerations taken for this survey.

5.1 Perform initial configuration

For a new project there are a number of objects that we must create instances of, as described below.

1. Create a project

The first task is to create a new project with a name, description and other attributes (owner, deliveries, deadlines?) we associate with our project. This helps to structure storage of data to allow us to reference it easily, even years down the line.

2. Enter artifacts

Then we enter all the artifacts that belongs to the project, and describe them properly. We select a number of cars in the luxury segment, and also

include our new concepts. We also decide to use pictures of the cars as a means of presentation that will allow us to deploy this survey via our company IT-infrastructure. This is a limitation of the affective bandwidth, but we will consider it and accept it as a necessary sacrifice to allow a low-cost survey of large scale. The pictures are uploaded to the system so that EvokeDB can create a presentation of each artifact through a template that we can easily manipulate to fit our requirements on styling.

3. Define realms

Now, here is our chance to structure the complexity of the car. We define four realms as 'shape', 'color', 'brand_image' and 'wheels'. The EvokeDB-interface also allows us to enter the stimuli of each realm, like {BMW, Mercedes Benz, Bentley, ... } for brand_image and so on, and make all the links to the respective artifacts, simply by clicking on buttons.

4. **Select words** -For this survey we are interested in *Kansei* in terms of luxury cars. As this is our first survey we do not have enough data from the past to allow EvokeDB to present us with a *Kansei*-language based on the realms we have entered. For now we just collect words from magazines, manuals and on-line resources relating to luxury cars and enter them into the database.

5. Group words into a Language

We are likely to run a lot of surveys if this one turns out successful, so it would be good to create some structure by allowing words to be grouped in languages. We decide to create a language called "Luxury" and link it to a set of words { luxurious, elegant, status, modern, expensive, comfortable, spacious, classic, ... }.

6. Create survey

We create a new survey and link it to the language and project we just created. A project can have many different surveys where each survey is about a specific language. Now we must also decide what type of range to use in the survey., and enter this into the min_value and max_value fields. We go with an AJAX-enabled slider between values 0 and 10 for this survey.

7. Link to senses

As we only have pictures to represent the cars, our subjects are restricted to use their eyes to explore them. Therefore, we make a single link between our

new survey and the sense of vision. This was our last task in the set-up of the survey.

All of these actions are accessible through a web-interface presented by each objects view. It is therefore very easy to list, create, edit, show or delete objects without any knowledge in programming, the corresponding tables in the database will automatically be updated by the mapping provided by RoR. This typical set-up can be done in less than 10 minutes by the average person, even less than creating a traditional paper-based questionnaire, and therefore EvokeDB offers a substantial saving in time compared to setting up a non-automated system. Furthermore, the user can now easily manage every study he or she undertakes, reference them for various projects and continue to enter more words and artifacts into the system.

5.2 Run survey

The survey is now ready for deployment on our sever. We decide to test it on our intranet first, and invite a few colleagues by email. These people simply click on a link in the email to be transferred to our server where they are greeted and prompted to fill in their personal data. The Subject-object in EvokeDB will handle that form-entry and save the information in the database. Successful entry of a subject will call upon the Survey-object and request that our survey is launched. Without going into the specifics of any code, I will try to explain this in simple terms. Basically, what happens next is that the Survey asks Project how many Artifacts are included. In a similar fashion the Survey will also extract all the words in the Language it belongs to. Then, for each artifact the survey will create a web-page that gives an instruction (defined by us) and presents the artifact with pictures according to the design we defined in our template. A list of all the words are also included with a slider to adjust the value of how strongly an artifact is associated with a particular word. The order of artifacts and words are randomized to avoid any statistical bias (it is also my experience that this helps combat fatigue in the subjects to some extent – a new order of words for each artifact forces them to focus instead of entering values by a pattern they easily adopt to when the list of words is static).

An example of a survey-interface for a project such as the one I have outlined here is (partially) depicted in Figure 3 below.

Note how a *Kansei* is evaluated – a slider is used to set a value in a range. This will allow subjects to enter their *Kansei* evaluation very precisely. The presentation of the word (i.e., “How *elegant* is this car?”) and the text accompanying the slider can be defined by the designer of the survey. Furthermore, it is possible to change the range and

even looks (size, color) of the slider should the need arise, as it is simply defined in CSS. Initially the slider is set at zero and displays a message prompting the subject to enter his or her *Kansei*. As soon as the slider is adjusted it will give feedback to the subject by stating the current rating. This is one of the strengths with AJAX, it makes the application more responsive in terms of how information is displayed and communicated. Another feature provided by AJAX can be seen in the dictionary. If a subject is uncertain of the meaning of a word, it is possible to simply click on it and a dictionary entry (or explanation provided by the designer) will emerge. This is a very subtle way to provide additional information without cluttering up the interface or confusing the subject by launching a new window like a traditional approach without AJAX would require. In case a subject wants to comment on a word, a similar feature of EvokeDB will allow this by providing a text-box. Such comments will create a new instance of a Note (see section 3.2) and attach itself to the word.

Once all the words have been evaluated the subject will click on a submit-button at the end of the form – this will send the data to SurveyEntry-object, which in turn will make sure it is stored in the survey_entries table in the database and request Survey to launch the next artifact. When there are no artifacts left to evaluate, the subject will be sent to a page where we thank him or her for the contribution to our survey. Again, this page can be chosen from a library of pages or created by templates.

5.3 Analyze results

Now that all the data is stored in our database, we can begin to analyze it to find some meaning within the *Kansei* it portrays. First of all, there are no real mysteries about this database, it can be accessed and queried like any other database, as depicted in Figure 2 - it is possible to export

How elegant do you feel this car is?

Not at all. [0] [10] Extremely so.

My rating: 7.6

Dictionary entry for elegant:

Main Entry: el-e-gant Pronunciation: 'e-li-gənt Function: adjective Etymology: Middle French or Latin; Middle French, from Latin elegant-, elegans; akin to Latin eligere to select — more at ELECT 1 : marked by elegance 2 : of a high grade or quality : SPLENDID <elegant gems priced at hundreds of thousands of dollars> synonym see CHOICE - el-e-gant-ly adverb

[close]

How modern do you feel this car is?

Not at all. [0] [10] Extremely so.

Not rated yet - please enter your Kansei.

Figure 3: Example of survey interface

its data or write SQL-queries from a command line or other interface to retrieve data for output. But, the new model and the relationships that come with it gives us a number of possibilities to do low-level analysis of *Kansei* with predefined or automated SQL-queries, accessible through an easy-to-use interface of buttons and drop-down menus. EvokeDB currently has automation for this type of basic queries. Based on the artifacts in the project, their realms and stimuli, and the senses used in the survey, the system can support a novice user with simple predefined queries, although highly specialized queries must still be entered manually. Still, this allows EvokeDB to function as a source of *Kansei* interpretation throughout a project – it can be referenced quickly to look up relationships without resorting to a full-fledged, time-consuming analysis. For example, with sufficient data it is possible to query the database not only like “What is the most elegant car?” but also use more subtle queries like “How does black compare to red in terms of elegance?” or “How is ‘BMW’ different from ‘Mercedes Benz’?” or “Are chromed rims more valued among men than women?” The relationships with different realms and senses bring meaning to words, meaning that has been obscured by simplified models. We still need a lot of data, but the relationships in EvokeDB are very easy to set up and therefore allows us to use all the stimuli from the presentations of artifacts.

6. DISCUSSION AND FUTURE PLAN

The system presented here was spawned from what always gripped me about *KE* – the lack of tools and guidelines, the “fuzziness” of how data is interpreted, the time and effort of making surveys, and the limited use of data. It was simply too much of an effort to implement. Data is the bread and butter of *KE*, and there is a need to simplify how we collect and store it. I envisioned a system that would automate surveys, and work as a sort of reference to *Kansei* – in short, it should be able to find the essence of various stimuli. I quickly realized the power of associations, but also the difficulties of implementing them. Even in a face-to-face interview-situation it is difficult to make a subject “think-out loud” and capture a free chain of associations, and I was focusing on unsupervised surveys of large scale. However, this new model of mine, which links entities in *KE* by rich relationships, is a first step towards a deeper understanding of *Kansei*, and I hope to make further progress into this challenging area.

Currently, the system works very well for my initial design intention, and I am actively using it for my ongoing projects. It is easy to set up and launch surveys, and a user can store data and reference it through the interface without any knowledge in SQL. This gives more control, there

is structure to all the entities involved in a project, and the system can offer some guidance in *Kansei* explorations. The analytical features, however, are still rather basic, and the predefined queries do not offer the best solution to efficiently retrieve and analyze data. I want to improve this before EvokeDB is released to a wider audience.

Many users are accustomed to information retrieval by keyword search, similar to those offered by search-engines on the Web, and I think this could be a feature that would bring a lot of usability to EvokeDB. However, a relational database is far more challenging to query than one consisting solely of text, due to its higher complexity in structure. In the last few years this problem has been addressed through active research, and there are today a number of systems available that can perform a keyword search on a relational database and rank the output efficiently [8-11]. These results are promising for future development of EvokeDB, as an implementation of keyword search would present a user with the easiest possible interface for information retrieval.

My next area of concern is the definition of realms. As far as basic realms go, there is no real challenge to define them and share data between domains, but it soon becomes difficult to extract and categorize subtle stimuli from an artifact. Yet these small design cues, such as character lines in a car, can hold a great deal of the image portrayed by an artifact, and without them a *Kansei* evaluation may not be valid. Schütte [12] addressed these issues by collecting product properties from different sources, such as existing products, new concepts and company image. These properties were subjected to a selection process where three different groups (customers, experts, company) rated the importance to guide the final compilation step. This method requires additional work, but should provide a more robust set of chosen properties.

There are many areas in *KE* that are still unexplored, but this gives us, as researchers, a great challenge and an opportunity for innovation. For my part, I believe that associations hold the key to successful definitions of accurate, yet compact *Kansei*-languages for complex artifacts, and I will continue to work on this in the future.

I welcome discussions about the EvokeDB-system and the model, and I would very much like to hear suggestions for improvements. I plan to release EvokeDB in the public domain, but until then I encourage interested parties to contact me by email (andnord@gmail.com) for beta source-code and documentation.

7. CONCLUSION

Kansei Engineering (*KE*) is becoming more popular as a set of methods to create products and services that meet

the emotional needs and wants from customers; an artifact should just *feel* right, and *KE* has the methods to succeed in this aspect. However, it is very difficult to implement *KE* in a design process without specialized training. In particular, surveys to collect data take time and skill to create and launch, with few specialized tools available in the public domain. Furthermore, there is a lack of guidelines on how to extract design parameters from an artifact – complex artifacts can lead to many different associations in a person, and this can in turn influence how the artifact is perceived emotionally. Because of this, it is hard to determine what property or characteristic of an artifact is most strongly correlated to a certain *Kansei*. Simplified models do not consider how an artifact is explored, or the context it is presented within. A new model of *Kansei*, which uses relationships between words, artifacts, realms, senses and subjects can address this problem and provide a more in-depth exploration of *Kansei*. I defined such a model and showed how to implement it by open-source components. A database-structure was defined, and the Ruby on Rails approach provided an elegant solution to the challenge of mapping objects and their relationships to tables in this database. AJAX was presented as a set of methods to bring usability and responsiveness to a web-application, and its merits for collecting *Kansei*-data was explained. Finally, the set-up of a simulated project was explained, along with a description of survey-automation and -interface. These results show that online surveys can be automated and thereby become accessible even for users with limited knowledge in computer programming. EvokeDB (Evaluation and Verification of Objects in *Kansei* Engineering Database) can offer designers and product developers a tool to collect, store and analyze *Kansei* data in a short time with less need for specialized training. A tool like this can easily be implemented and used throughout a design process.

REFERENCES

1. M. Nagamachi; *Kansei* Engineering: The Framework and Methods. In; M.Nagamachi (Ed.) *Kansei* Engineering 1; Kaibundo Publishing Co., Ltd., Kure, 1-9 (1997).
2. D.Thomas; Programming Ruby; The Pragmatic Bookshelf; Dallas, Texas, (2006).
3. D. Thomas, D. Heinemeier Hansson; Agile Web Development with Rails; The Pragmatic Bookshelf; Dallas, Texas, (2006).
4. N.C. Zakas, J. McPeak, J. Fawcett; Professional Ajax; Wiley Publishing, Inc., Indianapolis, Indiana, (2006).
5. C.E. Osgood, G.J. Suci, and P.H. Tannenbaum; The Measurement of Meaning; University of Illinois Press, Illinois, (1957).
6. T. Kitagawa, and Y.Kiyoko; A mathematical model of meaning and its application to multidatabase systems; Proceedings 3rd IEEE International Workshop on Research Issues on Data Engineering; Interoperability in Multidatabase Systems, Vienna, 130-135 (1993).
7. Y. Kiyoki, T.Kitagawa, and T.Hayama; A metadata-base system for semantic image search by a mathematical model of meaning; SIGMOD Record; 23 (4), 34-41 (1994).
8. S. Agrawal, S. Chaudhuri, G. Das, DBXplorer: a system for keyword-based search over relational databases, Proceedings of the 18th International Conference on Data Engineering, San Jose, California, (2002).
9. G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, S. Sudarshan, Keyword searching and browsing in databases using BANKS, Proceedings of the 18th International Conference on Data Engineering, San Jose, California, (2002).
10. V. Hristidis, L. Gravano, Y. Papakonstantinou, Efficient IR-style keyword search over relational databases, Proceedings of the 29th VLDB Conference, Berlin, (2003).
11. F. Liu, C. Yu, W. Meng, A. Chowdhury, Effective keyword search in relational databases, SIGMOD, Chicago, Illinois, 563-574 (2006).
12. S. Schütte; Developing the space of product properties supporting *Kansei* engineering procedure, International Journal of *Kansei* Engineering; 5 (4), 11-19 (2006).
13. The Apache Software Foundation, www.apache.org, (2006).
14. MySQL AB, dev.mysql.com, (2006).
15. The phpMyAdmin Project, www.phpmyadmin.net, (2006).
16. Design & Emotion Society, vvv.designandemotion.org (2006).

Andreas NORDGREN

I am a Ph.D-candidate of Keio University since 2004. My research focuses on *Kansei* Engineering methodologies for design support systems in the automotive field. In particular, I work with shape design and how it relates to customers' impressions of cars; this data is analyzed and used to explore the

emotional content in design spaces via a neural network solution. Prior to my doctoral studies, I received a Master's degree in Engineering Physics from Lulea Technical University in Sweden, and I have also spent time as a visiting researcher in the USA at Virginia Tech, and in Hong Kong at the Hong Kong University of Science and Technology. My future plan is to continue working with product development and *Kansei* Engineering – the beauty of machines is my passion, and I wish to explore every aspect of it.

Hideki AOYAMA

I am a Professor of Keio University. My research areas are Digital Design System (CAD), Digital Manufacturing System (CAM), Style Design System, Manufacturing System, and Manufacturing Process. I am a member of American Society of Mechanical Engineers, North American Manufac-

turing Research Institute of SME, Society of Manufacturing Engineers, Japan Society for Precision Engineering, Japan Society of Mechanical Engineers, Japan Society for Design Engineering, Japan Society for Abrasive Technology, Japan Society for Science of Design, and Japan Society of Die and Mould Technology.