Korz: Envisioning a Paradigm for Dynamic Multidimensional Contextual Variation (Extended Abstract)

David Ungar[1], Harold Ossher[2], and Doug Kimelman[2]

1 **IBM Research, San Jose, CA, USA**
davidungar@us.ibm.com
2 **IBM Research, Yorktown Heights, NY, USA**
{ossher,dnk}@us.ibm.com

Object-oriented inheritance graphs work well as long as each entity has only one dimension of variation. Sadly, as soon as a second dimension of variation is required, the object-oriented programmer is forced to resort to the visitor pattern, strategy pattern, or an aspect-oriented methodology, each requiring a cumbersome refactoring. In addition to the time spent splitting and refactoring, aspect-oriented, feature-oriented and related approaches increase effort by complicating the object model with additional modularity constructs to encapsulate concerns that do not align with the dominant dimension, and the means to compose or weave them.

Moreover, modern services and applications are becoming increasingly context-aware: they must adapt their behavior to the context in which they are running, and that context can change dynamically. For example, many applications on mobile devices are location-sensitive, and change the information they show, and/or their behavior, as location changes. Programming software that can adapt to context is challenging, especially when that context can change during execution. Each user has a unique context, and many different dimensions of context come into play simultaneously, such as who the user is, what his or her access rights and preferences are, what kind of device is being used for interaction, etc. There is a growing need for a programming paradigm that provides for multidimensional, dynamic, contextual variation.

Korz is a new computational model that combines implicit arguments and multiple dispatch in a slot-based model. This synthesis enables the writing of software that supports contextual variation along multiple dimensions, and graceful evolution of that software to support new, unexpected dimensions of variability, without the need for additional mechanism such as layers or aspects. Rather than bog down the object-oriented model with additional concepts such as layers, Korz provides a more fine-grained model out of which subjective objects can emerge.

A Korz system consists of a sea of method and data slots in a multidimensional space. There is no fixed organization of slots into objects – a slot pertains to a number of objects instead of being contained in a single object – and slots can come together according to the implicit context in any given situation, yielding subjective objects. There is no dominant decomposition, and no dimension holds sway over any other. IDE support is essential for managing complexity when working with the slot space and with subjectivity, allowing the task at hand to dictate what subspaces to isolate and what dominance of dimensions to use when presenting nested views to the user.

Computation occurs in a context, which is also multidimensional, binding specific values to some or all of the dimensions in the slot space. At each computation step, a slot is selected from the space, using multiple dispatch that is based on the context, a selector, and explicit arguments, and then that slot is evaluated. The context is implicitly passed along to this evaluation, and hence serves as a set of implicit arguments.

Korz reduces to procedural programming in the zero-dimensional case, and object-oriented

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–2
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

programming in the one-dimensional case (the single, implicit context element being the "self" or "this" object). We believe Korz to be simpler, more flexible, more dynamic, and more expressive than previous approaches, particularly for evolving a program when additional kinds of variation arise.

We have built and exercised a prototype Korz implementation using the Self language, virtual machine and environment. Our Korz prototype includes an interpreter, debugger, and a partial interactive development environment (IDE). The syntax used in the prototype is based on Self syntax. This early experience has revealed much that needs to be done, but has also shown considerable promise.

Korz's contribution lies in combining a relatively small number of pre-existing concepts: multiple dispatch, implicit, symmetric, named arguments, and slots with unified state and behavior as the fundamental particle. This combination yields more than the sum of the parts. Multiple dispatch supports multiple dimensions of variation, implicit arguments support evolution and contextual programming, and the slot-based metaphor allows for subjective gathering of slots into different "objects" for different situations. Together, they allow a program to be easily extended to accommodate new kinds of variation and new perspectives. In addition to accommodating ordinary programming tasks, and contextual variation for services and applications, this extensibility may also enable better architectures for reflection and more flexible and unified type systems.

This abstract is based on: Korz: Simple, Symmetric, Subjective, Context-Oriented Programming [1].

References

1 David Ungar, Harold Ossher, and Doug Kimelman. Korz: Simple, symmetric, subjective, context-oriented programming. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*, Onward! 2014, pages 113–131, New York, NY, USA, 2014. ACM.