Virtualizing Information Spaces for the Expansion and Integration of Heterogeneous Data Collections and Systems

Kostas Saidis*
saiko@di.uoa.gr

National and Kapodistrian University of Athens
Department of Informatics and Telecommunications

Abstract. The amount of information produced as well as consumed in the world is constantly expanding. Users anticipate their applications to cope with such an expansion, demanding the support of novel and heterogeneous data sources and collections. At the same time, due to the advanced connectivity offered by the expansion of the Internet, users need to use the same information in different contexts, reusing and refining data across applications. In this thesis, we deal with the common challenge underlying the above data expansion, integration and interoperability needs, which is how to automate the inclusion of new data in existing service provisions. To avoid invasive, time-consuming and expensive source-code extensions that frequently break applications, we propose the virtualization of information spaces, introducing the notion of virtual objects. Our proposal seeks to shift the barriers raised by heterogeneous data representations, formats and protocols by ascertaining the following assumption. Should we manage to isolate the "substance" of data from any application-specific "materialized forms", we can automate the process of detaching the data from one context and attaching the data to another, in a multitude of operational environments. Virtual objects are logically placed between the business-logic and the data-sources of applications to offer a common, reusable and composable runtime interpretation of data that transcends the application-context boundaries. On one hand, proposed virtual objects allow applications to adapt to new information, without changing their business-logic implementation. On the other, virtual objects can inter-connect data spaces to facilitate the integration and interoperability of information. We expect that data-centric applications will benefit from our approach, while our experimentation shows that virtual objects impose minimal overheads even when used atop heterogeneous sources.

Keywords: Virtual Objects, Information Expansion, Data Integration, Interoperability, Middleware, Domain-specific Languages

* Dissertation Advisor: Alex Delis, Professor

1 Introduction

The concept of *virtualization* originates from operating systems [1], where a software abstraction layer is introduced—the virtual machine—to behave as a computer system's hardware—a real machine. With the emergence of cloud computing and the "software as a service" paradigm, virtualization has offered flexibility, automation and ease of use, reducing OS installation, deployment and maintenance costs [2]. In addition, programming platforms such as Java or .NET have built upon virtual machines to offer features like architecture neutrality, garbage collection and network code loading, ultimately advancing developer productivity.

In this thesis, we use virtualization as a means to *enable the use of information spaces to transcend the application-context boundaries*, allowing users to create, share, reuse and refine information across applications. To this end, we work to automate the process of detaching an information space from one context and re-attaching the space to another in various operational environments. Should we view such data-detach and data-attach as logical operations, at a high level of abstraction, these operations play a significant role in the information space life-cycle, since information spaces:

a. *constantly expand*: automating the ability to attach new information spaces to existing application contexts will allow for the cost-effective introduction of novel types of content in applications and, thus, foster the expansion of information.
b. *outlive applications*: automating the ability to detach information spaces from their applications will simplify data migration and also reduce the data lock-in by legacy technology.
c. *should interoperate*: automating the ability to detach/attach information spaces from/to application contexts will clearly simplify interoperability.

In order to separate the information space from the application context, and offer reusable and composable data detach/attach operations, we propose *the virtualization of the information space*. The unique characteristic of our approach is that we use proposed *virtual objects* as archetypes of data items, offering a platonic view of data [3]; a virtual object aims to identify the "substance" of a data item, independently of any application-specific "materialized forms" such as storage-specific data layouts, protocol-based data representations or application-oriented data views. Our objective is to offer a software abstraction layer—a virtual machine for data objects—which is embedded in applications in order to enable the use of information to transcend the application-context boundaries.

1.1 Problem Definition

In working with virtual objects, our contribution is that we simplify and automate a wide variety of real-world data-expansion, migration and interoperability issues. Specifically, we manage to answer the common challenge underlying these

issues, which is *how to automate the inclusion of new data in existing service provisions*. Data-expansion calls for augmenting the information space with new data sources and collections, coping with the pressure to rapidly adapt to new information. In data-migration, the difference is that the new data source has to be migrated before participating in the existing service provision. In data-interoperability, the difference is that applications use a data-exchange protocol to act as the data sources of each other. Yet, in all cases, similar, if not identical, steps need to be performed, including:

Step 1: revisit data access actors to support newly encountered data sources or collections. For example, to support a new data source, an application may need to introduce a new machinery to access the new data, dealing with network connections, database queries, XML parsing, web services and other protocols. Similar revisions may need to be performed in order to support a novel collection type.

Step 2: introduce new business-logic actors to stage the new data. For example, a Java application has to introduce new Java classes and objects to stage the newly encountered data items at runtime.

Step 3: extend the current service provision to deal with the new data. For example, the application has to include the new business-logic actors in its existing implementation of services.

By virtualizing the information space, we work towards automating these steps in any operational environments. Viewed from different perspectives, the above steps raise various multi-disciplinary data integration, data quality and software evolution/adaptation issues [4–7]. Clearly, developers can handle these cases by code re-engineering. However, data expansion or integration requirements cannot always be predicted in detail during the initial application design and development phases. Consequently, the support of a new type of content may break the application, imposing drastic, invasive and expensive source code changes to all service provision actors. Developers may follow an ad-hoc approach to revisit service provision implementation and they can ultimately succeed in including this new content. However, yet another new requirement for supporting additional types of content may render this approach problematic, breaking service provision actors yet again.

The crucial need here is not to predict the future, but rather to achieve the return of investment on existing services; indeed, the challenge is to enable the existing service provision to operate atop constantly expanding, inter-connected and heterogeneous information spaces. Thus, a better approach is to base the application on a flexible framework that can isolate the application logic from the type of context and add indirection between the business-logic and the information space. Although adding indirection is a simple idea, designing a general and flexible framework for data expansion and integration is anything but simple. The framework needs to: a) isolate the structure of data, i.e., how the logical organization of data (e.g., the tuples of a database, or the elements of XML documents) map to the application's expectations; b) adapt the physical access to

data (e.g., provide network or database connections to objects in a way transparent to the application); c) abstract the object presentation, i.e., smoothly integrate the display of new kinds of objects in the application user interface; d) abstract the object manipulation, i.e., allow new object modification in a uniform way; and e) perform these tasks conveniently and efficiently, in particular without imposing significant runtime overhead over an inflexible, hard-coded implementation of the same features.

1.2 Contribution

Our proposal meets the above challenging requirements by virtualizing the information space. We use the term *information space* to refer to the dataset being managed by an application. Given that different applications develop different data manifestations and issue different mechanisms for their management, such application-specific characteristics designate the *application-specific information context*, or simply the *application-context*. The proposed virtual object environment decouples the information space from the application-context, achieving the multi-dimensional separation [8] of the following concerns:

1. *data-access/storage*: the storage representation(s) of data,
2. *data-synthesis*: the usage and composition of data,
3. *data-conceptualization*: the logical structure and modeling of data,
4. *data-discovery*: the data searching/indexing functionality.

These four concerns reflect the essential dimensions that couple the data to the application. By loosening these couplings, we manage to treat the four dimensions that compose the data/application interactions in in an orthogonal manner. Such a separation virtualizes the information space, automating the three data expansion/integration steps, by offering a *common interpretation of heterogeneous and diverse data* that transcends the application-context boundaries. Our proposal is well-aligned with the long-term objective of data independence [9, 10] and contributes to the open challenge of interpreting and synthesizing heterogeneous data in an automated manner [11, 12]. To the author's knowledge, the proposed virtual object approach is the first to deal with data expansion, interoperability and migration in a unified and integrated way. The majority of our work with virtual objects appears in [13–17] and [18].

Our implementation of virtual objects, called *DOLAR (Data Object Language And Runtime)*, is realized in Java and consists of a virtual object domain-specific embeddable language (DSEL) [19–21] and the respective runtime environment. We have used DOLAR in several real-world applications and operational environments, allowing us to effectively answer various data expansion, integration and interoperability needs. For example, DOLAR has allowed us to deal with data-expansion in Pergamos, the Univ. of Athens digital library (http://pergamos.lib.uoa.gr (http://pergamos.lib.uoa.gr)). Pergamos is the largest academic digital library in Greece, hosting about 300,000 items and exceeding 1 TB of space, while it has been in production use for more than five years. In

the thesis, we present how the use of DOLAR has helped us answer the need to gradually (a) use Pergamos to develop a variety of collections originating from independent digitization projects at the University, (b) add existing University collections in Pergamos, including digitized books, Domino-based dissertations, etc. The use of virtual objects has achieved the return of investment on Pergamos services, allowing the business-logic to adapt to the gradual inclusion of new data sources and collections without modifications. In general, embedding virtual objects in applications dissociates the application-logic from the data-inherent idiosyncrasies, enabling applications to extend their "low-level" information space options without modifying their "high-level" business-logic services. In terms of cross-context DOLAR usage, we have used virtual objects in a real-world interoperability scenario originating from John S. Latsis Public Benefit Foundation (http://www.latsis-foundation.org). We show how the virtual space has allowed the Foundation's digital archive and its collections to transcend the application boundaries, presenting the effectiveness, automation and reuse offered when dealing with data migration and interoperability issues. Finally, our experimental evaluation, carried out in both synthetic and production environments, shows that DOLAR-imposed operational overheads are minimal; DOLAR-enabled applications scale as well as the underlying datastore(s), even when used atop a variety of heterogeneous SQL, XML and Web data sources.

2 Related Work

Interoperability is a strong and diachronic requirement of data-intensive systems. Syntactic interoperability refers to the ability of systems to communicate and exchange data, while semantic interoperability refers to the ability of systems to accurately and meaningfully interpret and use the data in an automated manner [11, 12]. XML is a flexible markup language, offering an extensible representation to store and exchange data. To this effect, XML has been used to deliver a plethora of protocols for achieving syntactic interoperability. The Semantic Web [22] uses RDF and ontologies [23] to build upon XML to offer semantic interoperability. However, as put in [24], it may be that there are many semantic webs. Indeed, various middleware [25, 26] and service-oriented architectures [27] are used to consolidate heterogeneous APIs and services to make them interoperate in various operational environments. Although interoperability involves various cultural, social and legal issues [28], the key challenge is to achieve automation, enabling information that originates in one context to be used in another in ways that are as highly automated as possible [29].

This is the exact goal of our approach, decoupling the information space from the application-context to permit the use of data to transcend the application-context boundaries. Our virtual space proposal builds upon the separation of concerns to loosen the data/application couplings and automate the cross-context usage of data. The key here is that the proposed virtual space offers a uniform means to deal with the three data expansion/integration steps, which are relevant to all kinds of applications and all types of data. The virtual object space

can be used in conjunction with any application architecture and in the thesis we demonstrate the use of DOLAR in two production cases, where virtual objects are used in an MVC [30] and a REST[31] architecture.

Our approach to realize the virtual space as a DSEL aims to achieve an additional level of separation, that is, to decouple the data from the application-context in terms of the programming-language used. Our long-term goal with virtual objects is to offer an embeddable, language-independent, data-centric runtime environment—*a virtual machine for data objects*. Although our current DOLAR implementation is realized in Java, the key elements of the virtual space—including virtual objects and prototypes, datastore drivers and connectors, composition schemes and virtual object inheritance—can be clearly implemented in any general-purpose programming-language. Moreover, the prototype format feature of our DSEL builds upon the introspection of DOLAR prototypes to bridge virtual objects with multiple syntactic notations, including JSON, XML or our custom DOLAR syntax. Different applications can use different notations for storing and exchanging their prototypes, yet, all notations are parsed to generate identical prototypes in terms of the DOLAR language. This way, we manage to treat the data exchange and definition syntax as a pluggable component of the virtual space and not as a hard-coded option. In general, the DOLAR DSEL combines the flexibility and versatility of XML with the programmability and ease of use of scripting languages, in an effort to offer a foundation for issuing multiple data-definition utilities; for example, we also present *DOPs Creator*, a GUI tool for defining prototypes.

Finally, XML ontologies are widely used to model and conceptualize information in various contexts [32]. The main difference between ontologies and our approach—which stands for any comparison between ontologies and OO systems—is that ontologies use inference as the primary compositional mechanism, while in DOLAR we use virtual object instantiation and inheritance.

3 Overview of the Virtual Object Space

A virtual object offers a runtime manifestation of data which separates (a) how the data are being accessed and stored, (b) how the data are logically arranged at runtime and (c) how the data are synthesized and composed. Figure 1 depicts a virtual object, comprising (from left to right):

a. an application-specific physical/storage representation of data. The virtual object uses our *datastore driver (DOStore)* mechanism to capture the physical data access/storage options of applications.
b. an application-neutral logical structure of data. The virtual object conforms to a conceptualization expressed in terms of a *structural prototype* definition, offering a uniform, application-neutral representation of the logical arrangements of data at runtime.
c. an application-specific behavior of data. The virtual object adheres to a *behavioral prototype*, yielding application-specific sub-objects which are compatible with the application's service provision expectations.

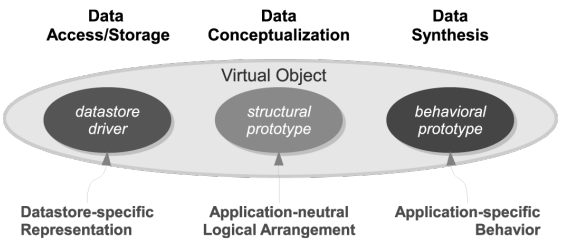

Fig. 1. A virtual object comprising a datastore driver, a structural and a behavioral prototype

By separating the left, data storage concern from the middle, data conceptualization concern, we manage to virtualize the datastores and treat heterogeneous storage artifacts as serializations of virtual objects. A virtual object adheres to the logical structure defined by its structural prototype consisting of *fields*, used for holding attributes, metadata or similar name/value pairs; *relation contexts*, used for holding the relationships among objects; and *stream handles*, used as pointers to files and similar document-based content. In OO terms, these elements designate the internal state of virtual objects; the structural prototypes provide the logical arrangements of data, offering the "data object classes", and virtual objects provide the runtime manifestation of data, offering the "data object instances". During instantiation, virtual objects use the *datastore driver* mechanism to establish a two-way link to the underlying storage artifacts. The driver connects the virtual field, relation-context and stream-handle runtime structures to any respective storage-specific structures employed by the datastore(s) used beneath. The datastore driver mechanism designates the data-access API of the virtual space, where the datastore drivers play the role of the data-access actors of Step 1. The virtual space uses the driver mechanism in a transparent to the developer fashion, offering an effective virtual object load/store metaphor which virtualizes the datastores. For example, developers can both load and store heterogeneous virtual objects using literally a single line of code. The developer is provided with the abstraction that heterogeneous storage artifacts are serializations of virtual objects, dealing with runtime artifacts that can access and also update any datastores with identical effectiveness and ease of use. Behind the scenes, the virtual space automatically deals with common data fetching tasks including (a) staging the data in virtual object runtime structures, (b) synchronizing the access to these structures and, finally, (c) flushing such structures to underlying datastores as needed.

By separating the middle, data-conceptualization concern from the right, data-synthesis concern, we separate the data-inherent from application-inherent behavior as follows:

– *data-inherent behavior* depends on the logical arrangements of the data in question; for example, the thumbnail of a "book" item may originate from

a digitized image of the book's first page, while the thumbnail of a "photo" item may originate from the digitized image of the photo itself.
- *application-inherent behavior* depends on the service provision requirements of the application at hand, regardless of the data items supported; for example, an application will either provide a thumbnail display for all of its items or it will not support such a display at all.

As far as the business-logic is concerned, the acquisition of a thumbnail from a photo, a book, or any other item should be transparent. The interpretation of data-inherent behavior (i.e. how to acquire the thumbnail) depends on the structure of the data item at hand. The implementation of application-inherent behavior (i.e. what to do with the thumbnail) depends on the service provision at hand. We use the notion of *composition schemes* to achieve this separation, offering an amalgamation of (a) a method construct of OO languages and (b) a projection operation of query languages. There is a direct analogy between the methods of "code objects" and the composition schemes of virtual objects: the methods/schemes available on an object define the messages the object can respond to. Yet, in contrast to "code objects", a virtual object *does not contain executable code* but responds to messages by transforming its internal state. A behavioral prototype defines a set of composition schemes, designating the runtime interface between virtual objects and application artifacts such as components and modules. The analogy, in a general-purpose OO language, is that schemes correspond to methods and behavioral prototypes correspond to interfaces. Schemes are realized and behave at runtime as sub-objects, transforming the structure of virtual objects to automatically match the application's expectations. Composition schemes are are a unique feature of our approach, providing the key for enabling virtual objects to expose different behavior when used across applications.

The virtual object environment is logically placed between the business-logic and the data sources to offer:

1. *a common representation of heterogeneous data.* For example, heterogeneous storage artifacts, such as SQL "book" tuples and XML "book" documents, will adhere to a uniform "book" virtual object representation at runtime. This is achieved with the help of the virtual object load/store metaphor, as realized by virtual object prototypes and datastore drivers described before. This metaphor enables us to treat heterogeneous storage artifacts as native objects of the virtual space, simplifying Steps 1 and 2.
2. *a uniform composition of semantically different data.* For example, diversely structured data, such as "book" and "photo" items, regardless of their storage representations, will conform to uniform manipulation at runtime. Here, we build upon the common representation offered by the virtual space to manage diversely structured virtual objects through a uniform programming interface. This is critical, as we seek objects that can be defined by their responses to messages and not by their internal representation [33]. With the use of our composition schemes and behavioral prototypes, the service provision logic

can catch up with the gradual addition of new content without source code modifications, automating Step 3.

3. *a set of reusable and composable data attach/detach mechanisms.* For example, applications can build upon the virtual objects of each other, regardless of their network location, logical structure or storage representation, to share, reuse and refine their data spaces. The data attach/detach mechanisms of DOLAR include *virtual object inheritance* and the *virtual space connector* and *prototype format* mechanisms. On one hand, prototype formats and virtual space connectors offer a pluggable and disciplined means to (a) exchange prototypes and (b) inter-connect virtual spaces, allowing DOLAR applications to instantiate the virtual objects of each other. On the other hand, virtual object inheritance allows for the reuse and refinement of virtual object definitions, automating the inclusion of new data in existing service provisions.

Building upon such virtual object structural and behavioral prototypes, we offer a mixin-based [34–36] multiple inheritance mechanism. With virtual object inheritance, we support the subclassing, subtyping and specialization features offered by inheritance in general, yet, we use these features to automate the cross-context usages of data. For example, virtual objects can be polymorphic in the OO sense, yet, the virtualization offered by the proposed environment allows virtual objects to effectively support different polymorphisms in different contexts. Virtual object inheritance provides the most powerful data attach/detach mechanism of our proposal, as it enables developers to reuse and extend virtual objects for attaching/detaching information among different contexts.

4 Conclusions

Our proposal offers a novel virtual object language and runtime that transcends the data, knowledge and software engineering boundaries in order for information spaces to transcend the storage, programming-language and application boundaries. We view our virtual object approach as an effort to offer a virtual machine for data objects, an embeddable environment that permits applications to develop a common, reusable and composable interpretation of data that transcends the application-context boundaries. The virtual space can be embedded in applications; it can reused across contexts; it can be refined to effectively match different service provision expectations. The virtual view of data employed in our approach allows us to deal with the common challenge underlying data expansion, integration and migration, that is, the adaption of existing service provisions to new information. As the amount of information undergoes constant growth globally, the above adaptation requirement becomes more and more dominant, as it involves all types of applications and all kinds of data. The proposed virtual objects meet this requirement by allowing developers to expand applications to support novel information without breaking their existing implementations.

References

1. R.P. Goldberg. Survey of virtual machine research. *IEEE Computer Magazine*, pages 34–45, June 1974.
2. M. Rosenblum. The reincarnation of virtual machines. *Queue*, 2(5):34–40, 2004.
3. D. Ross. *Plato's Theory of Ideas*. Oxford University Press, 1951.
4. M. Lenzerini. Data integration: a theoretical perspective. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, New York, NY, USA, 2002. ACM.
5. K. Chen, H. Chen, N. Conway, H. Dolan, and J. M. Hellerstein T. S. Parikh. Improving data quality with dynamic forms. In *ICTD'09: Proceedings of the 3rd international conference on Information and communication technologies and development*, pages 487–487, Piscataway, NJ, USA, 2009. IEEE Press.
6. I. Lukovic, P. Mogin, J. Pavicevic, and S. Ristic. An approach to developing complex database schemas using form types. *Softw., Pract. Exper.*, 37(15):1621–1656, 2007.
7. D. Parsons, A. Rashid, A. Telea, and A. Speck. An architectural pattern for designing component-based application frameworks. *Softw. Pract. Exper.*, 36(2):157–190, 2006.
8. P. Tarr, H. Ossher, W. Harrison, and S.M. Sutton. N degrees of separation: Multi-dimensional separation of concerns. In *Proc. of the 21st Int. Conf. on Software Engineering (ICSE)*, pages 107–119, 1999.
9. C. J. Date and P. Hopewell. File definition and logical data independence. In *Proceedings of the 1971 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '71, pages 117–138, San Diego, California, 1971. ACM.
10. J.M Hellerstein. Toward network data independence. *SIGMOD Rec.*, 32(3):34–40, September 2003.
11. S. Heiler. Semantic interoperability. *ACM Comput. Surv.*, 27(2):271–273, 1995.
12. A.M. Ouksel and A. Sheth. Semantic interoperability in global information systems. *SIGMOD Rec.*, 28(1):5–12, 1999.
13. K. Saidis, G. Pyrounakis, and M. Nikolaidou. On the effective manipulation of digital objects: A prototype-based instantiation approach. In *Proceedings of the 9th European Conference on Digital Libraries*, pages 26–37, Vienna, Austria, 2005.
14. K. Saidis, G. Pyrounakis, M. Nikolaidou, and A. Delis. Digital object prototypes: An effective realization of digital object types. In *Proceedings of the 10th European Conference on Digital Libraries*, Alicante, Spain, September 2006.
15. K. Saidis and A. Delis. Towards a Unified Runtime Model for Managing Networked Classes of Digital Objects. In *2nd DELOS Workshop on Foundations of Digital Libraries, In conjunction with the 11th European Conference on Digital Libraries*, Budapest, Hungary, 2007.
16. K. Saidis and A. Delis. Type-consistent Digital Objects. *D-Lib Magazine*, 13(5/6), May/June 2007. [doi:10.1045/may2007-saidis].
17. K. Saidis and A. Delis. Integrating multi-dimensional information spaces. In *2nd Workshop on Very Large Digital Libraries, In conjunction with the 13th European Conference on Digital Libraries*, Corfu, Greece, 2009.
18. K. Saidis, Y. Smaragdakis, and A. Delis. Dolar: virtualizing heterogeneous information spaces to support their expansion. *Software: Practice and Experience*, Accepted for publication, 2010, available online at http://dx.doi.org/10.1002/spe.1050.

19. P. Hudak. Building domain-specific embedded languages. *ACM Computing Surveys*, 28(4es), 1996.
20. D. S. Wile. Supporting the DSL Spectrum. *Journal of Computing and Information Technology*, CIT 9(4):263–287, 2001.
21. M. Mernik, J. Heering, and A.M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, 2005.
22. N. Shadbolt, T. Berners-Lee, and W. Hall. The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.
23. S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. C. A. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The semantic web: The roles of xml and rdf. *IEEE Internet Computing*, 4:63–74, 2000.
24. C. Marshall and F. Shipman. Which semantic web? In *HYPERTEXT '03: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, pages 57–66, New York, NY, USA, 2003. ACM.
25. P. A. Bernstein. Middleware: a model for distributed system services. *Commun. ACM*, 39:86–98, February 1996.
26. F. Kon, F. Costa, G. Blair, and R. H. Campbell. The case for reflective middleware. *Commun. ACM*, 45:33–38, June 2002.
27. T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 2005.
28. P. Miller. Interoperability. What is it and Why should I want it? *Ariadne, Issue 24*, June 2000. http://www.ariadne.ac.uk/issue24/interoperability/intro.html.
29. IDF. The DOI Handbook, The International DOI Foundation, 2006. Edition 4.4.1, October 2006, [doi:10.1000/182].
30. E. Gamma, R.Helm, R.Johnson, and J.Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1997.
31. R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 407–416, New York, NY, USA, 2000. ACM.
32. B. Chandrasekaran, John R. Josephson, and V. Richard Benjamins. What are ontologies, and why do we need them? *IEEE Intelligent Systems*, 14(1):20–26, 1999.
33. H. Lieberman. The continuing quest for abstraction. In *Proceedings of the 20th European Conference on Object Oriented Programming (ECOOP)*, pages 192–197, 2006. doi: 10.1007/11785477_12.
34. G. Bracha and W. R. Cook. Mixin-based Inheritance. In *OOPSLA / ECOOP*, pages 303–311, 1990.
35. Y. Smaragdakis and D. Batory. Implementing layered designs with mixin layers. In *Proceeding of the European Conference on Object Oriented Programming (ECOOP)*, 1998.
36. N. Scharli, S. Ducasse, O. Nierstrasz, and A. Black. Traits: Composable units of behaviour. In *ECOOP 2003 – Object-Oriented Programming*, volume 2743 of *Lecture Notes in Computer Science*, pages 327–339. Springer Berlin / Heidelberg, 2003.