# DATA AND COMPUTATION INTEROPERABILITY IN INTERNET SERVICES

Sergey Boldyrev, Dmitry Kolesnikov, Ora Lassila and Ian Oliver
*Nokia Corporation, Espoo, Finland*

Keywords: Service, Semantic Web, Data, Ontology, Computation, Privacy, Latency, Cloud Computing, Interoperability.

Abstract: Next generation distributed systems should be seamlessly spanned around heterogeneous concepts of the information providers, devices manufacturers and the cloud infrastructures. The enabling components such as Data, Computation, Scalable performance and Privacy aspects should be elaborated and leveraged in order to provide a foundation of such systems.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

## 1 INTRODUCTION

The ICT industry is undergoing a major shift towards *cloud computing*. From the technological standpoint, this entails incorporating both "in-house" and outsourced storage and computing infrastructures, integrating multiple partners and services, and providing users a seamless experience across multiple platforms and devices. On the business side, there is an anticipation of improved flexibility, including the flexibility of introducing new, multisided business models where value can be created through interactions with multiple (different types of) players. Many companies are trying to answer questions like "*Who* might find this information valuable?", "What would happen if we provided our service for *free of charge*", and "What if *my competitor(s)* did so?". The answers to such questions are hightlighting opportunities for disruption and identfication of vulnerabilities (). Thus, slicing and investigating technological components we are seeking the answers in further thinking and collabration work with industrial researches and academic society.

In a world of fully deployed and available cloud infrastructures we can anticipate the emergence of distributed systems that can seamlessly span the "information spaces" of multiple hardware, software, information and infrastructure providers. Such systems can take advantage of finely granular and accountable processing services, can integrate heterogeneous information sources by negotiating their associated semantics, and ultimately free users of mundane challenges and details of technology usage (such as connectivity, locus of data or computation, etc.). We foresee greater *reuse* of information *and* computational tasks.

In order for us to understand the opportunities and – perhaps more importantly – the challenges of such a future, we must establish a clear framework for *interoperability*. Aspects of data and computation semantics, performance and scalability of computation and networking, as well as threats to security and privacy must be discussed and elaborated. This paper aims to be the beginning of this discussion.

## 2 DATA

### 2.1 About Semantics

In the broader sense, in the context of data, the term *semantics* is understood as the "meaning" of data. In this document, we take a narrow, more pragmatic view, and informally define semantics to be the set of declared dependencies and constraints that define how data is to be processed. In a contemporary software system, some part of the system (some piece of executable software) always defines the semantics of any particular data item. In a very practical sense, the "meaning" of a data item arises from one of two sources:

1. The relationships this item has with other data items and/or definitions (including the semantics of the relationships themselves). In this case, software "interprets" what we know about the data item (e.g., the data schema or ontology). An elementary example of this is object-oriented poly-

morphism, where the runtime system can pick the right software to process an object because the class of the object happens to be a subclass of a "known" class; more elaborate cases include ontology-based reasoning that infers new things about an object.

2. Some software (executing in the system runtime) that "knows" how to process this data item directly (including the system runtime itself, as this typically "grounds" the semantics of primitive datatypes, etc.). In this case, the semantics is "hard-wired" in the software.

All software systems have semantics, whether the semantics is defined *implicitly* or *explicitly*. As for data specifically, the semantics are typically defined explicitly, in the form of datatype definitions, class definitions, ontology definitions, etc. (these fall under category #1 above; semantics of those things that fall under category #2 are typically either implicit or in some cases explicit in the form of external documentation).

## 2.2 Constituent Technologies of "Data Infrastructure"

From a practical standpoint, all applications need to query, transmit, store and manipulate data. All these functions have dependencies to technologies (or *categories* of technologies) which we in aggregate shall call *Data Infrastructure*. The constituent technologies themselves have various dependencies, illustrated (in a simplified form) in Figure 1. From the data semantics standpoint, we are mostly here focused on the technologies inside the box labeled "Data Modeling Framework".

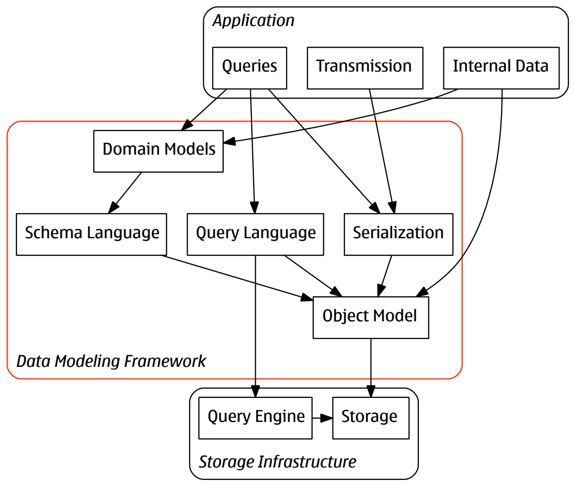

Figure 1: Dependencies in "Data Infrastructure".

Domain Models: Application semantics is largely defined via a set of *domain models* which establish the necessary vocabulary of the (real-world) domain of an application. Meaningful interoperability between two systems typically requires the interpretation of such models.

Schema Language: Domain models are expressed using some *schema language*. Typically such languages establish a vocabulary for discussing types (classes), their structure and their relationship with other types. Examples of schema languages include *SQL DDL*, W3C's *OWL* and *XML Schema* as well as Microsoft's *CSDL*. Class definitions in object-oriented programming languages (e.g., Java) can also be considered as schema languages.

Query Language: An application's queries are expressed in some *query language*; this language may either be *explicit* (such as *SQL* or *SPARQL*) or *implicit* (in the sense that what effectively are queries are embedded in a programming language). Queries are written using vocabularies established by a query language (the "mechanics" of a query) and relevant domain models.

Serialization: For the purposes of transmission (and sometimes also for storage) a "syntactic", external form of data has to be established. A *standardized* serialization syntax allows interoperating systems to exchange data in an implementation-neutral fashion. Serialization syntaxes do not have to be dependent on domain models.[1]

Object Model: Ultimately, there are some definitions about the structure common to all data; we refer to these definitions as an *object model* (in some literature and in other contexts, the term *metamodel* is also used). Essentially, an object model is the definition of "what all data looks like" and thus forms the basis for a schema language, a query language and a serialization syntax. Examples of object models include W3C's *XML DOM* and *RDF*.[2]

## 2.3 On Metadata and Provenance

Interoperable exchange of data can often benefit from *metadata* to describe the data being interchanged. Typically, such exchange occurs in use cases involving management of data that originates in *multiple*

[1] In practice, developers often have a hard time separating conceptual views of data, domain models, and like, from concrete, syntactic manifestations of data. This has lead to all kinds of unnecessary dependencies in the implementations of large systems. Consider yourselves warned.

[2] We are referring to RDF's graph-based metamodel, not *RDF Schema* which should be considered a schema language.

*sources* (and may – independently of sources – have *multiple owners*) and/or data that is subject to one or more *policies* (security, privacy, etc.). Much of the metainformation we aspire to record about data is in various ways associated with *data provenance*. Over the years lots of work has been invested in provenance in various ways – () provides a general overview of these efforts in the database field. In cases where a multi-cloud solution is used to implement *workflows* that process data, we may also be interested in solutions for *workflow provenance*.

In practice, incorporating all or even any metainformation into one's (domain) data schema or ontology is difficult, and easily results in a very complicated data model that is hard to understand and maintain. It can, in fact, be observed that provenance-related metadata attributes (such as *source* and *owner*) are *cross-cutting aspects* of associated data, and by and large are independent of the domain models in question. The problem is similar to the problem of cross-cutting aspects in software system design, a situation[3] that has led to the development of *Aspect-Oriented Programming* ().

From the pragmatic standpoint, the *granularity* of data described by specific metadata is an important consideration: Do we record metadata at the level of individual objects (in which case "mixing" metadata with the domain schema is achievable in practice), or even at finer (i.e., "intra-object") granularity; a possible solution to the latter is presented in ().

## 3 COMPUTATION

From the computation perspective we are faced with a task to enable and create *consistent* (from the semantics standpoint) and *accountable* components that can be leveraged and reused in a larger, distributed information system. This translates to several pragmatic objectives, including (but not limited to) the following:

- Design and implementation of scenarios where computation can be described and represented in a system-wide understandable way, enabling computation to be "migrated" (transmitted) from one computational environment to another. This transmission would involve reconstruction of the computational task at the receiving end to match the defined semantics of task and data involved. It should be noted that in scenarios like this the semantics of computation are described at a fairly high, abstract level (rather than, say, in terms of low-level machine instructions).
- Construction of a system where larger computational tasks can be decomposed into smaller tasks or units of computation, independent of what the eventual execution environment(s) of these tasks (that is, independent of hardware platforms or operating systems).

[3]The motivating situation is sometimes referred to as the "tyranny of dominant decomposition" ().

Traditional approaches to "mobile code" have included various ways to standardize the runtime execution environment for code, ranging from hardware architectures and operating systems to various types of *virtual machines*. In a heterogeneous configuration of multiple device and cloud environments, these approaches may or may not be feasible, and we may want to opt for a very high level (possibly purely declarative) description of computation (e.g., describing computation as a set of goals to be satisfied).

High-level, declarative descriptions of computation (e.g., functional programming languages such as *Erlang*) afford considerable latitude for the target execution environment to decide how computation is to be organized and effected. This way, computation can be scheduled optimally with respect to various constraints of the execution environment (latency budget, power or other resource consumption, remaining battery level, etc.) and decisions can also be made to migrate computation to other environments if needed.

## 4 MANAGED CLOUD PERFORMANCE

A new outlined paradigm of distributed systems requires an explicit orchestration of computation across Edge, Cloud and Core depending on components states and resource demand, through the proper exploit of granular and therefore accountable mechanisms. The behavior analysis of the software components using live telemetry gathered as the cloud sustain production load is mandatory.

End-to-end latency is seen by us as a major metric for quality assessment of software architecture and underlying infrastructure. It yields both user-oriented SLA, the objectives for each Cloud stage. The ultimate goal is the ability to quantitatively evaluate and trade-off software quality attributes to ensure competitive end-to-end latency of Internet Services.

On another hand, the task of performance analysis is to design systems as cost effectively as possible with a predefined grade of service when we know the future traffic demand and the capacity of system elements. Furthermore, it is the task to specify methods for controlling that the actual grade of service is fulfilling the requirements, and also to specify emergency actions when systems are overloaded or technical faults occur.

When applying Managed Cloud Performance in practice, a series of decision problem concerning both short-term and long-term arrangements appears:

- provide granular latency control for diverse data and computational load in hybrid Cloud architectures;
- resolution short term capacity decisions include for example the determination of optimal configuration, the number of lives servers or migration of computation;
- resolution of long term capacity decisions such as decisions concerning the development and extension of data- and service architecture, choice of technology, etc;
- control of SLA at different levels, e.g. consumer services, platform components, etc;
- quality assessment of distributed software architecture.

If performance metrics of a solution is inadequate then business is impacted. It has been reported by multiple sources (): 100ms of latency on Amazon cost them 1% in sales; Goldman Sachs makes profit of a 500 ms trading advantage. Goldman Sachs used Erlang for the high-frequency trading programs.

The role of software behavioral analysis therefore is crucial in order to achieve proper level of accountability of the whole system, constructed out of the number of the granular components of Data and Computation, thus, creating a solid fundament for the Privacy requirements.

## 5 PRIVACY IN THE CLOUD

The scope of *privacy enforcement* being proposed here is to provide mechanisms by which users can finely tune the ownership, granularity, content, semantics and visibility of their data towards other parties. We should note that *security* is very closely linked with privacy; security, however, tends to be about *access control*, whether an agent has access to a particular piece of data, whereas privacy is about *constraints on the usage of data*. Because we have not had the technological means to limit usage, we typically cast privacy as an access control problem; this precludes us from implementing some use cases that would be perfectly legitimate, but because the agent cannot access relevant data we cannot even discuss what usage is legal and what is not. In other words, security controls whether a particular party has access to a particular item of data, while privacy controls whether that data can or will be used for a particular purpose.

As Nokia builds its infrastructure for data, the placement and scope of security, identity and analysis of that data become imperative. We can think of the placement of privacy as described in Figure 2 (note that in this diagram, by "ontology" we really mean *semantics*).

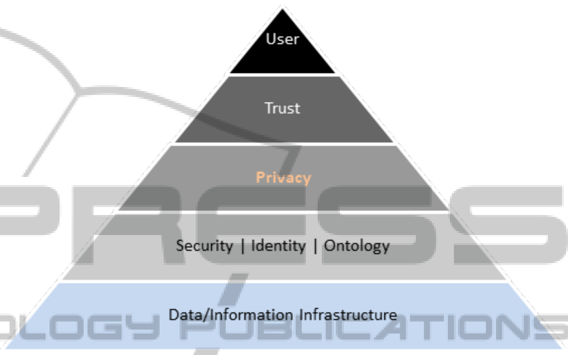
User
Trust
Privacy
Security | Identity | Ontology
Data/Information Infrastructure

Figure 2: Privacy, overview.

Given any data infrastructure we require at minimum a security model to support the integrity of the data, an identity model to denote the owner or owners of data (and ideally also provenance of the data) and an ontology to provide semantics or meaning to the data. Given these we can construct a privacy model where the flow of data through the system can be monitored and to establish boundaries where the control points for the data can be placed.

## 6 CONCLUSIONS

As stated in above, the rapidly shifting technology environment raises serious questions for executives about how to help their companies to capitalize on the transformation that is underway. Three trends – anything as a service, multisided business models and innovation from the bottom of the pyramid – augur far-reaching changes in the business environment and require radical shifts in strategy. The fine-grained (and thus easily accountable) software frameworks allow customers to monitor, measure, customize, and bill for asset use at much more fine-grained level than before. Particularly, b2b customers would benefit since such granular services would allow the purchasing of fractions of service and to account for them as a *variable cost* rather than consider some *capital investments* and corresponding overhead.

Therefore, the driving agenda above can be seen as a creation of tools for the next developer and consumer applications, computing platform and backend infrastructure, backed up by application technologies to enable qualitative leap in services' offering, from the perspectives of scalable technology and business models with high elasticity, supporting future strategic growth areas and maintenance requirements.

## REFERENCES

G. Kiczales, "Aspect-Oriented Programming," *ACM Computing Surveys*, vol. 28, no. 4, 1996.

P. Buneman and W.-C. Tan, "Provenance in databases," in *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 2007, pp. 1171–1173.

H. Ossher and P. Tarr, "Multi-Dimensional Separation of Concerns in Hyperspace," in *Proceedings of Aspect-Oriented Programming Workshop at ECOOP'99*, C. Lopes, L. Bergmans, A. Black, and L. Kendall, Eds., 1999, pp. 80–83.

O. Lassila, M. Mannermaa, I. Oliver, and M. Sabbouh, "Aspect-Oriented Data," accepted submission to the W3C Workshop on RDF Next Steps, World Wide Web Consortium, June 2010.

Marko A. Rodriguez, "The RDF virtual machine," in *Knowledge-Based Systems*, Volume 24, Issue 6, August 2011, Pages 890-903.

Uday Reddy, "Objects as closures: abstract semantics of object-oriented languages," Published in: Proceedings of the 1988 ACM conference on LISP and functional programming , Proceeding LFP '88, ACM New York, NY, USA 1988.

QtClosure project https://gitorious.org/qtclosure

SlipStream project https://github.com/verdyr/SlipStream

Cisco System Inc, White Paper, Design Best Practices for Latency Optimization

ELATA project https://github.com/fogfish/elata

Souder S., High Performance Web Sites, O'Relly, 2007, ISBN 0-596-52930-9

The Psychology of Web Performance, http://www.website optimization.com/speed/tweak/psychology-web-perfor mance

L. Kleinrock, Queueing Systems, Wiley Interscience, 1976., vol. II: Computer Applications (Published in Russian, 1979. Published in Japanese, 1979.)

http://tools.ietf.org/html/rfc3393

McKinsey Column, "Clouds, big data, and smart assets," Financial Times, Septemeber 22 2010.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS