Position Statement

Alexander Paar

WeST Spring Mindswap

May 03-04, 2012

Widely used object-oriented programming languages such as Java or C# include a built-in static type system. Programming language type systems provide a conceptual framework that makes it particularly easy to design, understand, and maintain object-oriented systems. Understanding systems and searching for information are the most frequent developer activities [5]. Facilitating maintenance – in the presence of a continuously changing workforce – is crucial.

Position 1 *Static type systems are de facto indispensable (no need for Curry-style in practice).*

In recent years, Semantic Web technologies such as the Resource Description Framework (RDF) and a variety of ontology languages have paved the way for standardized formal conceptualizations of all kinds of knowledge. Numerous ontologies have been developed to conceptualize a plethora of domains of discourse. Unfortunately, statically typed object-oriented programming languages such as Java and C# on the one hand and ontology languages such as the Web Ontology Language (OWL) on the other hand have different conceptual bases [4].

Position 2 *With the emergence of a variety of schema and ontology languages conventional built-in programming language type systems have reached their limits.*

The impedance mismatch [2] between the formal foundations of domain models and the conceptual bases of programming languages leads to brittle software. However, versatile type definition mechanisms provided by schema languages such as XML Schema (XSD) and rich sets of concept constructors offered by a variety of, for example, Description Logics-based ontology languages have the potential to speed up design time. The open world assumption and the absence of the unique name assumption are particularly applicable in many Internet-scale application scenarios. Automatic reasoning can facilitate the development of intelligent applications.

Position 3 *Schema and ontology languages need to be integrated with programming languages.*

Proposals and solutions to solve the integration problem exist [1, 3, 4]. The integration with programming language type systems will facilitate the transformation of domain models into succinct program code and will be beneficial for a number of program analysis activities (e.g., searching for concept references). Improved accessibility of different kinds of information sources will potentially expose software applications to increased amounts of data (the amount of data stored in the world's databases is estimated to double every 20 months anyway).

Position 4 *Programming languages must provide for efficient and configurable processing of big data.*

Default evaluation strategies can be readily used for moderate data sizes, customized strategies can be implemented for multi-core processors and big data (i.e. concurrent execution, lazy evaluation).

References

[1] G. Bracha. Pluggable type systems. In *OOPSLA Workshop on Revival of Dynamic Languages*, October 2004. 1

[2] R. Lämmel and E. Meijer. Revealing the X/O impedance mismatch (changing lead into gold). In *Datatype-Generic Programming*, volume 4719 of *Lecture Notes in Computer Science*, page 80. Springer-Verlag, June 2007. 1

[3] E. Meijer and P. Drayton. Static typing where possible, dynamic typing when needed. In *OOPSLA Workshop on Revival of Dynamic Languages*, 2004. 1

[4] A. Paar and D. Vrandečić. Zhi# – OWL aware compilation. In G. Antoniou, M. Grobelnik, E. P. B. Simperl, B. Parsia, D. Plexousakis, P. D. Leenheer, and J. Z. Pan, editors, *ESWC '11: Proceedings of the 8th Extended Semantic Web Conference*, volume 6644 of *Lecture Notes in Computer Science*, pages 315–329. Springer, June 2011. 1

[5] J. Singer, T. Lethbridge, N. Vinson, and N. Anquetil. An examination of software engineering work practices. In *Conference of the Centre for Advanced Studies on Collaborative research (CASCON)*, pages 21 – 35. IBM Press, 1997. 1