Chapter 3.3
Interoperability:
Introduction and State of the Art

Florian Matthes
Technical University Hamburg-Harburg
Harburger Schlösserstraße 20
D-21071 Hamburg, Germany

Today, successful application development is rarely carried out by coding application programs from scratch, instead there is a strong tendency to exploit services provided through open and modular environments already populated with prefabricated and packaged functionality and information.

In this scenario, fully integrated persistent programming environments excel through their persistence abstraction, their elaborate data modelling support and their well-organized component libraries all described in earlier chapters. However, for the construction of large-scale industrial applications, application programmers still have to utilise commercially-available system services and tools outside of the persistent programming environment, for example, to access legacy data and code. Furthermore, programmers have to be able to make persistent data and code maintained by a persistent programming environment accessible to other systems, for example, for data and system integration purposes.

The work described in this chapter applies the database and programming language technology developed in the FIDE project to improve the interoperability between independently developed, generic system servers. At present, each of these servers comes with its own naming, typing, binding and persistence concepts so that application developers who wish to exploit multiple services within a single application find themselves working in a quite complex and fairly unfriendly and unsafe environment. Examples for these difficulties can be found at the interfaces to file systems, SQL databases, window systems or RPC packages.

Distributed object management is viewed as a promising approach to build scalable distributed systems that are also capable of integrating legacy (database) systems by means of a unified object paradigm [6]. There are numerous proposals for specific object models like DSOM of IBM [4], DOM of GTE [6], Network Objects of Modula-3 [2] and future versions of Microsoft’s OLE [7] and there are several related standardization efforts like CORBA of the OMG [3] and the OSF DCE/DME [9]. For a detailed feature analysis of these models see [8, 5].

Similar to object models, the high-level type systems of the persistent languages presented in Chapters 1.1.1 to 1.1.3 provide mechanisms like type abstraction, type quantification and subtyping to write detailed specifications
of external service interfaces and to classify services based on their signatures. Persistent
languages go beyond object models since they also define a rich (higher-order) computational
model to describe behaviour, as it is required to “glue together” services from several providers.

By re-interpreting schemas as type definitions and databases as typed variables and by treating
lifetime as a type-independent property, a uniform linguistic interface for data modelling, computation and communication
can be developed (see also Chapter 3.3.3). As a consequence of such an integrated
view, formerly disjoint concepts such as databases, program and module
libraries, files or repositories can now be treated uniformly as POSs
differentiated essentially by the types of objects they contain and by the operational
abstractions they provide [10]. Therefore, distributed databases, multi-databases and federated databases can be understood as restricted (particularly interesting) cases of interoperating persistent application systems.

Instead of using persistent languages to glue together services developed with incompatible technologies, one could also envision a scalable persistent architecture based on a core “low level persistent language” (LLPL) that provides a stable, secure and platform-independent basis for the construction of multi-paradigm systems. As discussed in more detail in [1] (see Chapter
2.1.1), a major concern of this approach is to achieve high levels of longevity (data and programs have to be accessible for several $10^2$ upto $10^3$ years) without compromising data integrity and security through uncontrolled (non-typed) data access.

Chapters 3.3.1 and 3.3.2 show how database technology can be applied
to semi-structured data stored in files and how database languages can be utilised to query and update files at a high level of abstraction. The mapping between structured database objects and linear file representations is defined by means of grammars. The results of this work in the direction of heterogeneity and data integration are applied in the O2 Views system.

Chapter 3.3.3 introduces a canonical model of persistent object systems based on generalised notions of values, types, bindings and signatures to de-
scribe the issues that have to be solved to achieve a type-safe interoperation between persistent objects supported by independently-developed generic
servers. The model utilised in tgar Chapter underlies the Tycoon programming
language TL and the scalable and interoperable Tycoon programming
environment, both described in Chapters 1.1.1 and 2.1.4.

References

   FIDE Technical Report Series FIDE/92/51, FIDE Project Coordinator, Depart-