A High-Level Symbolic Language for Distributed Web Programming

Cees T. Visser
Computer Science Department
Vrije Universiteit Amsterdam
The Netherlands

Anton Eliëns
Computer Science Department
Vrije Universiteit Amsterdam
The Netherlands

Abstract *We present a flexible language framework (DLP) for distributed active object configurations that facilitates the construction of networked software systems for symbolic application domains on the Web.*

The language DLP consists of a number of high-level modeling and multi-language collaboration concepts, suitable for architecting distributed object systems: active multi-threaded symbolic as well as imperative objects are the basic building blocks for the construction of distributed software systems. Symbolic autonomous objects cooperate by means of high-level synchronous or asynchronous conditional communication primitives and the current underlying run-time system supports incremental dynamic loading and linking of objects at arbitrary nodes in an Internet environment.

Innovating aspects of our current research are the integration of symbolic processing, imperative processing and object distribution, combined with several object collaboration services for distributed processing on the Web. For a realistic evaluation of the language concepts we've used a prototype system to reimplement all aspects of the system in DLP itself, including the compiler-frontend, compiler-backend and the object collaboration services.

Keywords and phrases: distributed symbolic Web processing; interoperable multi-paradigm active objects; distributed object trader services.

1 Introduction

Distributed imperative as well as declarative languages differ substantially with respect to their general concurrency models and provide often rather different synchronization techniques. This divergence is influenced by particular application or modeling requirements and the corresponding need for different levels of abstraction, orthogonality of language constructs, distribution transparency, and scalability. These four characteristics are often to some extent conflicting from an efficiency point of view. The main theme language designers are often faced with is to find a careful balance between efficiency and ease of use in order to be able to construct software architectures that are still manageable from several distributed software engineering related perspectives.

Currently, Java is without doubt the most popular language for programming the Internet. However, according to our experience, the imperative nature of the Java language as well as for example the available concurrency concepts are too low-level for a number of important distributed application domains.

Compared to higher-level declarative languages, the imperative characteristics of the Java language often require an explicit and

detailed description of specific application features. One concrete example that illustrates this lower-level involvement are the required specifications regarding message marshaling. Especially when prototyping and exploring new distributed application domains in the context of the Internet this is a rather undesirable requirement.

Although a more detailed overview and analysis of other aspects is beyond the scope of this paper, similar observations can be made with respect to the Java multithreading and synchronization model as illustrated in the next sections.

Our major objective with respect to distributed processing on the Internet was to provide a significantly more convenient language framework, while maintaining a high-degree of interoperability between the language DLP and its companion language Java. The language DLP solves the above-mentioned issues by means of distributed communication facilities with transparent method message marshaling and code mobility. At the same time intra-object and inter-object synchronization is supported by asynchronous or synchronous conditional communication primitives which allows the software developer to focus on application specific modeling aspects instead of the distribution related restrictions as imposed by the Java language.

This paper will not discuss the general implementation techniques for high-level declarative functional or logic programming languages. Introductions for these programming paradigms can be found in [11], and [8]. More advanced research efforts or overviews are descibed in [5], [6], [10] and several other publications. More specific Internet related studies of security and access control policies can be found in, for example, [12].

2 The Run-Time System

The basic architectural entities for the construction of distributed Internet software in DLP are multi-threaded symbolic objects. These symbolic objects can cooperate with Java objects and their activities are supported by a *local object service* layer as well as a *distributed object service* layer.

Figure 1 gives a schematic overview of the local object services for both DLP and Java objects in a single Internet node. The incremental loader handles both DLP objects and Java objects. All local object and method references are registered by the loader and exported information can be used by other nodes to establish a particular object collaboration or to invoke a remote object method.

Typically, when activating a single active object, the DLP incremental dynamic loader and linker resolves all intra-object method references and transfers the execution to the object. The major scheme for resolving object and method references is based on an incremental loading-on-demand protocol. This is an important and necessary requirement for languages like DLP that support code mobility. Migrating code can, for example, refer to objects (by name) that are not yet known at a remote Internet node, in which case the run-time loader and linker is responsible for resolving the references from a code repository. Once resolved, new passive or active instances of an object can be created dynamically, and active objects can establish their own distributed collaboration patterns.

We considered code migration support as an important characteristic in a highly dynamic environment like the Internet. It allows for tailoring remote service requests without the need to foresee all possible object interactions. Code migration in the language DLP inherits the security and safety properties of the Java execution environ-

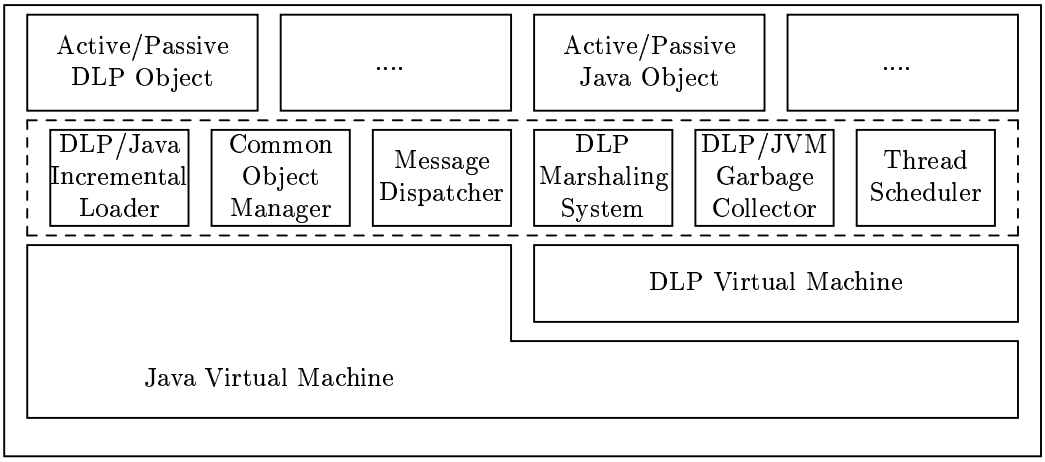

Figure 1: Local Object Services

ment since all the language constructs execute directly on top of the Java Virtual Machine [9].

The distributed object service layer provides for a general object naming and reference service that allows for a flexible distributed cooperation of objects (Figure 2). In general this facility will be used for collaboration with autonomous objects that provide major services in a particular software architecture. The location transparency as provided by the distributed object service layer facilitates the invocation of methods without the need for explicit knowledge about the topology of a distributed software configuration. To some extent, the current run-time infrastructure reflects several important architectural characteristics of the CORBA [4] and ODP [3] processing models.

3 Communication and Synchronization

As mentioned, distributed objects cooperate by means of high-level conditional synchronous or asynchronous communication primitives. When an active object invokes a method in a particular remote executing object environment all the arguments as specified in the invocation are transparently serialized. As opposed to Java, this holds for method arguments that point to e.g. nested data structures as well as method arguments that refer to executable language constructs. At the callee side messages are automatically de-serialized, symbol references are updated locally and possibly still unknown object references in mobile code are dynamically resolved by the DLP loader and linker.

Conditional synchronous or asynchronous communication in DLP is expressed by means of so called accept statements. Each accept statement consists of a disjunction of accept expressions.

A single accept expression can specify a method with its associated parameters that an object is willing to accept. At the same time, a single accept expression can specify a sequence of additional conditions (guards). Variables that occur in a method or condition specification of an accept expression can refer to the local state of the object that's willing to accept a message or can refer to the arguments of a particular incoming method requests. In case an incoming method request matches the specification of a method entry of an accept expression and also satisfies the corresponding condition then the message will be accepted.

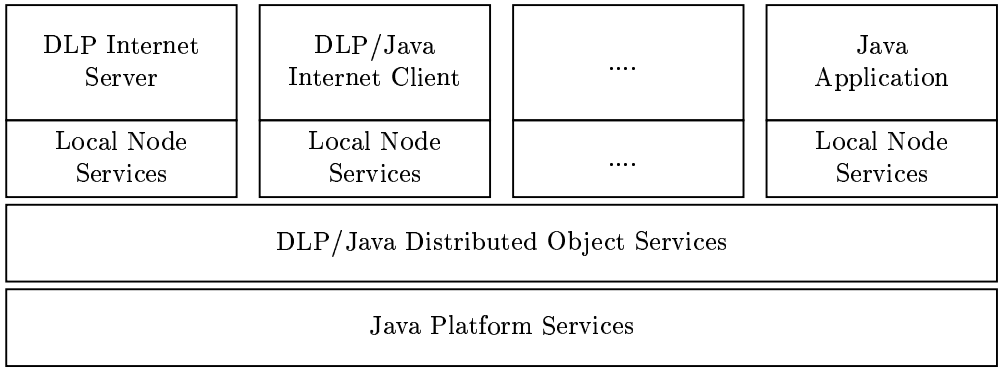

Figure 2: Distributed Object Services

We would like to conclude this overview with a summary of the most important characteristics of the language DLP: (1) multi-threaded active objects, (2) transparent method message marshaling, (3) high-level synchronous and asynchronous conditional communication primitives, (4) automatic garbage collection, (5) mobile code support, and (6) distributed object trader services. For further information, a detailed description and motivation as well as examples of DLP are given in [2].

4 Performance Aspects

The highly dynamic run-time characteristics of objects are an important aspect of the DLP language. In particular with respect to the incremental loading and linking activities at run-time (e.g. as the result of migrating code) that automatically result in the update of an object configuration in an executing environment.

As an illustration of the involved run-time costs consider the current DLP system library. This library results in the processing of more than 100 objects that facilitate many run-time services. The objects and a number of meta-programming related object characteristics are transparently available to all DLP client or server environments.

Incremental loading, linking and object initialization of the entire DLP library requires currently about 365 milliseconds on a Ultra SPARC II and 350 milliseconds on a Pentium III workstation. The corresponding processing time of a single incremental object update is a number of milliseconds which is often acceptable for distributed applications in an Internet context.

After the loading and linking process the run-time overhead of the invocation of library methods is similar to the regular invocation of Java object methods. However, several performance aspects can be improved as discussed in the next section.

5 Current Status and Future Work

We have currently implemented the language DLP on top of the Java platform. Although the language has more attractive distributed processing capabilities, it's still interesting to have occasionally Java as a sort of companion language. The Java platform has a wealth of additional functionality (libraries) that can be used in the language DLP or in a Java context.

One of the main aspects that needs to be explored in more detail is related to several important optimization techniques when compiling DLP to the Java plat-

form. The compilation of non-imperative languages to the Java platform is relatively new. Several implementation efforts for higher-level languages have been described in more detail (e.g. [1]) but more advanced compile-time optimization schemes are required.

The intended optimization strategies are from several perspectives very different from the techniques used for imperative languages like Java. Preliminary experiments showed however that the current implementation will benefit in several important ways from techniques like compile-time abstract interpretation and partial evaluation strategies [7], [6], [5], [10], especially because these approaches are to a large extent independent of a number of limitations and restrictions as imposed by the JVM.

These sort of optimizations are very likely the most promising approach for improving the performance of innovating high-level languages and will make new programming paradigms more suitable for distributed processing on the Internet.

References

[1] P. Bothner. *Kawa: Compiling Scheme to Java*, in *Usenix Conference Proceedings*, Dec 1998

[2] Anton Eliëns. *DLP : A Language for Distributed Logic Programming: Design, Semantics and Implementation.* Wiley, 1992

[3] ITU/ISO/IEC. *Reference Model of Open Distributed Processing*, International Organization for Standardization, 1997

[4] Object Management Group. *CORBA 2.3.1 Specification, The Common Object Request Broker: Architecture and Specification*, Oct 1999

[5] Thomas W. Getzinger. *Abstract Interpretation for the Compile-Time Analysis of Logic Programs.* Ph.D. dissertation, University of Southern California, Dec 1993

[6] Manuel V. Hermenegildo, Richard Warren, and Saumya K. Debray. *Global Flow Analysis as a Practical Compilation Tool.* Journal of Logic Programming, 13(4), pp. 349–366, Aug 1992

[7] Gerda Janssens and Maurice Bruynooghe. *Deriving Descriptions of Possible Values of Program Variables by Means of Abstract Interpretation.* Journal of Logic Programming, 13(2 & 3), pp. 205–258, July 1992

[8] Peter M. Kogge. *The Architecture of Symbolic Computers.* McGraw-Hill, 1991

[9] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification.* Addison-Wesley, 1999

[10] Per Mildner. *Type Domains for Abstract Interpretation: A Critical Study.* Ph.D. dissertation, Uppsala University, May 1999

[11] S.L. Peyton Jones. *The Implementation of Functional Programming Languages.* Prentice-Hall, 1987

[12] Jan Vitek and Christian D. Jensen (eds.), *Secure Internet Programming: Security Issues for Mobile and Distributed Objects.* Lecture Notes in Computer Science, 1603, Springer-Verlag, 1999