PERGAMON

Computer Languages, Systems & Structures 28 (2002) 1–2

COMPUTER LANGUAGES, SYSTEMS & STRUCTURES

www.elsevier.com/locate/cl

Editorial

Computer languages and security

The aim of this Special Issue of Computer Languages is to outline the new challenges that mobile computing and security problems address to the development of programming languages.

The first paper, *Information Flow for Algol-like Languages*, by David Clarke, Chris Hankin, and Sebastian Hunt, deals with static security analysis for detecting illegal information flow in imperative programs. Both direct (explicit assignment to variables) and indirect information flows (information that can be deduced from the flow of control) are taken into account. The proposed analyses have the distinctive feature of being sensitive to the flow of control in the program, meaning that they can take into account that variables can have different security levels at different program points. This distinguishes the analysis from other information-flow analyses proposed in the literature. The analyses are first developed for a simple imperative while-language and then extended to a language called Idealized Algol featuring higher-order functions and a richer store-model based on references.

The following two papers aim at designing type systems that provide programmers with various safety and security guarantees. They both refer to the Ambient Calculus, introduced by Cardelli and Gordon with the main purpose of explicitly modelling mobility.

The paper *Orderly Communication in the Ambient Calculus* by Torben Amtoft, Assaf J. Kfoury, and Santiago M. Pericas-Geertsen introduces and studies a novel-type system, which is a strict extension of the original-type system by Cardelli and Gordon. This type system allows different types of data to be exchanged within the same ambient, with no type confusion. The type system is built over new classes of types and behaviors that convey information on the input/output behavior of processes. Behaviors, assigned to processes, trace the types of the data exchanged within ambients, while behavior contexts, the constituents of capability types, provide for the representation of the composition of behaviors resulting from exercising capabilities.

The article *Behavioural Typing of Safe Ambients* by Michele Bugliesi and Giuseppe Castagna, proposes a type system for Safe Ambients (a variant of the Ambient Calculus) aimed at controlling ambients' behavior: where ambients may move and where they may be opened. Process types describe the capabilities that processes may exercise depending on the nesting level at which the effect can be observed. This static information can be applied to enforce several security properties based on classifications of names into domains. In particular, observing the nesting level at which capabilities are exercised allows to detect implicit mobility.

Also the paper *Information Flow Analysis of Mobile Ambients through Security Boundaries* by Chiara Braghin, Agostino Cortesi, and Riccardo Focardi, contributes in the scenario of Ambient Calculus. The paper considers the problem of guaranteeing confidentiality of certain "high-level" ambients. The solution is based on allowing these ambients to move around only when cloaked in

1477-8424/02/$ - see front matter
PII: S0096-0551(02)00005-X

"boundary" ambients. This new notion of boundary allows to express information flow properties to model a multilevel security policy. The paper proposes a refinement of the Control Flow Analysis of Hansen–Jensen–Nielson for Mobile Ambients. The goal of the proposed analysis is that of verifying that high-level data do not move outside boundary ambients. The modified analysis is obtained by maintaining more information about the ambient nestings, keeping track of potentially dangerous boundary crossings.

Finally, the last paper, *Distributed Call-Tracking for Security* by Dilsun Kirli, presents a functional core language for code mobility, equipped with a type system that is able to describe not only functionality of terms, but, exploiting type annotations, also keeps track of which (and how many times) function calls are potentially involved in the evaluation of an expression. This allow to foresee computational resources needed in the execution of mobile code. Code mobility is achieved by means of remote evaluation of expressions. Type information is useful to detect potentially resource-consuming computations and to prevent denial-of-service attacks.

Many thanks to Blaire Mossman, managing editor of Computer Languages, for inviting us to edit this issue, and to directly contribute to it. We are grateful to the authors for providing high quality papers, and to all the reviewers for their qualified criticisms, suggestions, and advices.

Agostino Cortesi
Riccardo Focardi
Department of Computer Science,
Ca' Foscari University, *Venice*, *Italy*
E-mail address: cortesi@dsi.unive.it (A. Cortesi).