Tue 10:30-12:00 pm - Rose Ballroom B

Registration-Based Language Abstractions

- Samuel Davis, University of British Columbia, Canada
- Gregor Kiczales, University of British Columbia, Canada

Programming language innovation has been hindered by the difficulty of making changes to existing languages. A key source of difficulty is the tyrannical nature of existing approaches to realizing languages—adding a new language construct means that any tool, document or programmer that works with the language must be prepared to deal with that construct.

A registration-based approach makes it possible to define language constructs that are not tyrannical. They are instead transient—the program appears to be written using the constructs only so long as a given programmer wants to see it that way. This approach may have the potential to greatly facilitate programming language innovation.

Pinocchio: Bringing Reflection to Life with First-Class Interpreters

- Toon Verwaest, SCG University of Berne, Switzerland, Switzerland
- Camillo Bruni, SCG University of Berne, Switzerland, Switzerland
- David Gurtner, SCG University of Berne, Switzerland, Switzerland
- Adrian Lienhard, SCG University of Berne, Switzerland, Switzerland
- Oscar Nierstrasz, SCG University of Berne, Switzerland, Switzerland

To support development tools like debuggers, runtime systems need to provide a meta-programming interface to alter their semantics and access internal data. Reflective capabilities are typically fixed by the Virtual Machine (VM). Unanticipated reflective features must either be simulated by complex program transformations, or they require the development of a specially tailored VM. We propose a novel approach to behavioral reflection that eliminates the barrier between applications and the VM by manipulating an explicit tower of first-class interpreters. Pinocchio is a proof-of-concept implementation of our approach which enables radical changes to the interpretation of programs by explicitly instantiating subclasses of the base interpreter. We illustrate the design of Pinocchio through non-trivial examples that extend runtime semantics to support debugging, parallel debugging, and back-in-time object-flow debugging. Although performance is not yet addressed, we also discuss numerous opportunities for optimization, which we believe will lead to a practical approach to behavioral reflection.

Concurrency by Modularity: Design Patterns, a Case in Point

- Hridesh Rajan, Iowa State University, United States
- Steven Kautz, Iowa State University, United States
- Wayne Rowcliffe, Iowa State University, United States

General purpose object-oriented programs typically aren't embarrassingly parallel. For these applications, finding enough concurrency remains a challenge in program design. To address this challenge, in the Panini project we are looking at reconciling concurrent program design goals with modular program design goals. The main idea is that if programmers improve the modularity of their programs they should get concurrency for free. In this work we describe one of our directions to reconcile these two goals by enhancing Gang-of-Four (GOF) object-oriented design patterns. GOF patterns are commonly used to improve the modularity of object-oriented software. These patterns describe strategies to decouple components in design space and specify how these components should interact. Our hypothesis is that if these patterns are enhanced to also decouple components in execution space applying them will concomitantly improve the design and potentially available concurrency in software systems. To evaluate our hypothesis we have studied all 23 GOF patterns. For 18 patterns out of 23, our hypothesis has held true. Another interesting preliminary result reported here is that for 17 out of these 18 studied patterns, concurrency and synchronization concerns were completely encapsulated in our concurrent design pattern framework.

Wed 10:30-12:00 pm - Rose Ballroom B

Understanding Reduced Resource Computing

- Martin C Rinard, MIT, United States
- Henry Hoffman, MIT, United States
- Sasa Misailovic, MIT, United States
- Stelios Sidiroglou, MIT, United States

We present several general, broadly applicable mechanisms that enable computations to execute with reduced resources, typically at the cost of some loss in the accuracy of the result they produce. We discuss several general computational patterns that interact well with these resource reduction mechanisms, present a concrete manifestation of these patterns in the form of simple model programs, perform simulation-based explorations of the quantitative consequences of applying these mechanisms to our model programs, and relate the model computations (and their interaction with the resource reduction mechanisms) to more complex benchmark applications drawn from a variety of fields.

Programming With Time

- Andrew Sorensen, Australian National University, Australia
- Henry Gardner, Australian National University, Australia

The act of computer programming is generally considered to be temporally removed from a computer program's execution. In this paper we discuss the idea of programming as an activity that takes place within the temporal bounds of a real-time computational process and its interactions with the physical world. We ground these ideas within the context of livecoding – a live audiovisual performance practice. We then describe how the development of the programming environment "Impromptu" has addressed our ideas of programming with time and the notion of the programmer as an agent in a cyber-physical system.

Language Virtualization for Heterogeneous Parallel Computing

- Hassan Chafi, Stanford University, United States
- Zach DeVito, Stanford University, United States
- Adriaan Moors, EPFL, Switzerland
- Tiark Rompf, EPFL, Switzerland
- Arvind Sujeeth, Stanford University, United States
- Pat Hanrahan, Stanford University, United States
- Kunle Olukotun, Stanford University, United States
- Martin Odersky, EPFL, Switzerland

As heterogeneous parallel systems become dominant, application developers are being forced to turn to an incompatible mix of low level programming models (e.g. OpenMP, MPI, CUDA, OpenCL). However, these models do little to shield developers from the difficult problems of parallelization, data decomposition and machine-specific details. Ordinary programmers are having a difficult time using these programming models effectively. To provide a programming model that addresses the productivity and performance requirements for the average programmer, we explore a domain-specific approach to heterogeneous parallel programming. We propose language virtualization as a new principle that enables the construction of highly efficient parallel domain specific languages that are embedded in a common host language. We define criteria for language virtualization and present techniques to achieve them.We present two concrete case studies of domain-specific languages that are implemented using our virtualization approach.

Thu 10:30-12:00 pm - Rose Ballroom B

Flexible Modeling Tools for Pre-Requirements Analysis: Conceptual Architecture and Research Challenges

- Harold Ossher, IBM TJ Watson Research Center, United States
- Rachel Bellamy, IBM TJ Watson Research Center, United States
- Ian Simmonds, IBM TJ Watson Research Center, Israel
- David Amid, IBM Haifa Research Center, Israel
- Ateret Anaby-Tavor, IBM Haifa Research Center, Israel
- Matthew Callery, IBM TJ Watson Research Center, United States
- Michael Desmond, IBM T.J. Watson Research Center, United States
- Jacqueline de Vries, IBM T.J. Watson Research Center, United States
- Amit Fisher, IBM Haifa Research Center, United States
- Sophia Krasikov, IBM T.J. Watson Research Center, United States

There is a serious tool gap at the very start of the software lifecycle, before requirements are formulated. Pre-requirements analysts gather information, organize it to gain insight, envision alternative possible futures, and present insights and recommendations to stakeholders. They typically use office tools, which give them great freedom, but no help with consistency management, change propagation or migration of information to downstream modeling tools. Despite these downsides to office tools, they are still used in preference to modeling tools, which are too hard to learn and too constraining. This paper introduces the notion of flexible modeling tools, which blend the advantages of office and modeling tools. We propose a conceptual architecture for such tools, and outline a series of research challenges to be met in the course of realizing them. We also briefly describe the Business Insight Toolkit (BITKit), a prototype tool that embodies this architecture.

To Upgrade or Not to Upgrade: Impact of Online Upgrades across Multiple Administrative Domains

- Tudor Dumitras, Carnegie Mellon University, United States
- Priya Narasimhan, Carnegie Mellon University, United States
- Eli Tilevich, Virginia Tech, United States

Online software upgrades are often plagued by runtime behaviors that are poorly understood and difficult to ascertain. For example, the interactions among multiple versions of the software expose the system to race conditions that can introduce latent errors or data corruption. Moreover, industry trends suggest that online upgrades are currently needed in large-scale enterprise systems, which often span multiple administrative domains (e.g., Web 2.0 applications that rely on AJAX client-side code or systems that lease cloudcomputing resources). In such systems, the enterprise does not control all the tiers of the system and cannot coordinate the upgrade process, making existing techniques inadequate to prevent mixed-version races. In this paper, we present an analytical framework for impact assessment, which allows system administrators to directly compare the risk of following an online upgrade plan with the risk of delaying or canceling the upgrade. We also describe an executable model

that implements our formal impact assessment. Through three case studies, we demonstrate that our model correctly identifies the situations where the risk of incorrect behavior during an online upgrade exceeds that of continuing the execute an older version with known software defects. Our model enables a systematic approach for deciding whether an online upgrade is appropriate and when is the best time to execute it, thereby reducing unexpected service interruptions and undesirable program behaviors.

Managing Ambiguity in Programming by Finding Unambiguous Examples

- Kenneth C. Arnold, MIT Media Lab, MIT Mind Machine Project, United States
- Henry Lieberman, MIT Media Lab, MIT Mind Machine Project, United States

We propose a new way to raise the level of discourse in the programming process: permit ambiguity, but manage it by linking it to unambiguous examples. This allows programming environments to work with high-level descriptions that lack precise semantics, such as natural language descriptions or conceptual diagrams, without requiring programmers to formulate their ideas in a formal language first. As an example of this idea, we present Zones, a code search and reuse interface that connects code with ambiguous natural language statements about its purpose. The backend, called ProcedureSpace, induces relationships between statements purpose, static code analysis features, and natural language background knowledge. ProcedureSpace can search for code given statements of purpose or vice versa, and can find code that was never annotated or commented. Since completed Zones searches become annotations, system coverage grows with user interaction. Users in a preliminary study found that reasoning jointly over natural language and programming language helped them reuse code.