◇ RESEARCH STATEMENT ◇

Jan Vitek

Professor of Computer Science, Northeastern University

Programming languages provide software developers with abstractions. These abstractions are key to the disciplined and efficient construction of correct software systems. The shift to multi-core architectures and the increasing availability of large-scale multiprocessors are the latest example of disruptive change to programming practice. To meet the challenges and to benefit from the opportunities presented by concurrency, it is necessary to adapt the entire software development stack including programming models, integrated development environment, compiler, and program verification technologies.

My research agenda is to investigate *new programming models for multi-core and massively parallel architectures* and their impact on the practice of programming. Rather than taking a narrow view of the problem, I have chosen a comprehensive approach spanning software engineering, programming languages and compilers. New programming abstractions are first given rigorous semantic foundations. As this is generally not sufficient to establish viability, the second and crucial step is thus build software systems that demonstrate the benefits of our ideas. The final step of the process is to transition ideas into software that can be widely used. For this I rely on collaboration with industry and involvement in standard bodies. I work closely with most major software vendors (IBM, Microsoft, SUN and Motorola) as well as companies like Boeing and Lockheed Martin and organizations such as the European Space Agency (ESA). Recently, some students and I have started a company, Fiji Systems, to allow us to apply some of our ideas and hopefully open new research directions in safety critical systems.

Advances in Concurrent and Distributed Programming

Presenting a simple yet efficient *programming model for parallel systems* remains an open problem. In collaboration with researchers at the IBM T.J Watson, I am working on two major initiatives. Firstly, as the leader of the Thorn project my responsibility is to coordinate work on a family of language for multi-core architectures. Our target is business logic running on backend servers with integrated XML and database support. We have identified different programmer communities with different trade-offs between simplicity and efficiency. We will deliver solutions customized to the needs of the different developer communities. Our first subgoal is a new dynamic language named `thorn`. Its goal is to cover the space occupied by scripting languages such as Python, Ruby, PHP and JavaScript. Unlike these language, `thorn` will have built in support for concurrency and a strong security model based on isolation and capabilities. The Thorn project is a long term collaboration between IBM, Universities and the open source community. The second, research effort I have been involved with is the IBM X10 language. X10 is being developed within the DARPA HPCS challenge and targets high-end massively parallel architectures. One of the innovation introduced in X10 is a set of abstractions for representing locality and inter-processor communication in systems made of thousands of cores. In X10, locality is encoded as type annotation in an ownership-type system and this information is used by the X10 compiler to very-lightweight threads.

The impetus for *location-aware and mobile computation* grew out of the need to address changes in the nature of distributed computing. The goal of my research in this area was to find abstractions for programming wide area networks with simple and clean semantics, and efficient implementations. My contributions are twofold. With Guiseppe Castagna, I defined the Seal calculus, a core model of mobile computational systems. In parallel, I designed and implemented a language called JavaSeal, an extension to Java inspired by the calculus. Both languages endow program logic with location-awareness and control over the logical and physical location of computation. The Seal project was among the first attempts to explore the design

space of programming languages for mobile systems from both theoretical and practical angles. JavaSeal provides a secure programming model ensuring that different seals can not interfere with one another and supports mobility of code and state. Numerous other mobile systems have been implemented in various other languages, but few were actually deployed. JavaSeal is perhaps unique in that we were able to implement and evaluate a medium sized commercial application. Seal influenced the work of a number of researchers in theory of mobility. Location mobility as a result of process synchronization which we introduced in Seal as a natural extension of π-calculus communication primitives was later adopted by Levi and Sangiorgi, and then in several other Ambient variants. The Seal calculus influenced the design of Boxed Ambients and also the Calculus of Mobile Resources which inherits the interaction pattern for exiting agents, and the fact that mobility takes place on the anonymous content of locations. The Seal calculus has also played a role in the design of the Crypto-Loc calculus which has a similar communication model. Finally Seal primitives are also at the basis of the definition of box-π, an extension of the π-calculus for securely integrating trusted and untrusted off-the-shelf software components.

Software transactional memory has received much attention of late as it holds the promise of being a programming model for concurrent systems that is, at the same time, simple and easy to prove correct. By transposing the decades of experience in the database community to programming languages it may possible to benefit from optimistic concurrency while at the same time making data races all but impossible. Transactional memory abstractions permit logically concurrent access to shared data, but ensure through some combination of hardware, compiler, and runtime support that such accesses do not violate intended serializability invariants. By doing so, pernicious errors such as data races can be eliminated with performance improvements. Atomicity and isolation properties make transactions composable with one another: the execution of one transaction can't affect the visible behavior of another. Together with Suresh Jagannathan and Tony Hosking, we have defined *semantic models* of nested, multi-threaded, transactions in object-oriented languages as well as coordination languages in the Linda family. We have also implemented the only software transactional memory infrastructure for real-time systems and shown that it outperforms traditional lock-based implementations in real application workloads. With Rachid Guerraoui and Michal Kalpka, I have worked on the only *open source workload for transactional memory* systems, STMBench7. A testimony to their usefulness is that our benchmarks have been ported to C#, Java, ML and Haskell by other groups. With Frank Tip and Mandana Vaiziri of IBM, I am working on *data-centric synchronization*. We have observed that transactional memory does not offer assistance to developers in selecting where transactional boundaries should be placed in their code. We intend to remedy this problem by allowing developers to write declarative specifications of data consistency.

Papers: [24, 45, 73, 88, 5, 31, 65, 66, 41, 44, 35, 14, 84, 34, 33, 18, 16, 15, 77, 50, 19, 13, 78, 75, 12, 79, 76, 70, 71].

Service: I founded the Mobile Object Systems (MOS) workshop which was held yearly between 1995 and 2005 in co-location with the ECOOP conference and gave a forum for the research in mobile computations. I have co-founded the ACM-SIGPLAN TRANSACT workshop for transactional memory systems, and was its first General Chair in 2005. The workshop is refereed and quite selective, attracting between 30 and 50 submissions each year since its inception. I have co-organized two International Summer Schools on Trends in Concurrency (TIC). The schools were held in Bertinoro in 2006 and Prague in 2008 with more than 50 participants and funding from IBM, Microsoft and the NSF. I have been PC chair of the International Conference on Coordination Models and Languages (COORDINATION) and am on its steering committee. I have served on the PCs of the workshops on the Foundations of Coordination Languages and Software Architectures (FOCLASA) and Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES), and the Conference on Agent Systems and Applications (ASA/MA). The STMBench7 benchmark suite, JavaSeal and Thorn have all been released under open source licenses.

High-level Hard Real-time Computing

Real-time system are fundamentally concurrent programs with strong timeliness constraints. They are thus extremely challenging to get right as correctness is time dependent. The costs of existing development methodologies are considerable in terms of software development times and the likelihood of software defect remains high. My research has focused on providing higher-level abstractions for programming hard real-time applications. My first research effort in the field was funded by DARPA and involved the development of the first high-performance and open source virtual machine implementing the *Real-Time Specification for Java* (RTSJ). Since we completed our work, the RTSJ has been adopted for multi-million line projects to develop shipboard computing systems and avionics software. The success of the RTSJ is due to a programming model that allows developers to write real-time programs in a type-safe language, thus reducing many opportunities for catastrophic failures, and also permits hard, soft- and non-real-time code to interoperate in the same execution environment. Our system remains the only viable open source platform for developing real-time applications in the Java programming language. Together with Boeing we implemented and deployed a real-time control system for an Unmanned Aerial Vehicle known as the ScanEagle. As a result, we were the first to fly a RTSJ enabled plane. This success was recognized by a Duke's Choice Award for innovation in Java technology. Since then we have begun a collaboration with the European Space Agency which as a first step involved porting our virtual machine on the LEON, a radiation hardened chip used in space missions.

Memory management is a critical issue for correctness and performance in real-time embedded systems. Recent work on *real-time garbage collectors* has shown that it is possible to provide guarantees on worst-case pause times and minimum mutator utilization time. In our research on real-time garbage collection (RTGC), we were the first (and only to date) to evaluate a RTGC on realistic hard real-time workloads. We compared several garbage collector with region based memory management and where able to demonstrate that the overhead of existing techniques were higher than previously believed. Inspired by those results we have proposed a new technique called Hierarchical Real-time Garbage Collection. Reminiscent of hierarchical schedulers, Hierarchical RTGC relies on hierarchy of garbage collectors that are scheduled independently and manage the memory in different parts of the address space of a virtual machine. Parts of this research are being conducted with Microsoft and IBM Research.

Even the best real-time garbage collectors will not be able to ensure response times in the microseconds. For this we have developed *Reflexes*, a new abstraction for writing highly responsive systems in Java and investigated the virtual machine support needed to add Reflexes to a Java environment. The key idea of Reflexes is to execute real-time thread in a partition of memory that is protected from interference from the GC and from plain Java threads. We have implemented our proposal in a high-performance Java virtual machine and extended the javac compiler to perform additional safety checks needed to ensure isolation. Our implementation of Reflexes was evaluated on several programs including an audio-processing application. We were able to run a Reflex at 22.05KHz with less than 0.2% missed deadlines. This is remarkable as it shows that it is possible to write real-time code in Java that, after heavy optimization, has the same performance and predictability as a low-level C implementation. We have further extended our programming model to support stream processing applications. Our ideas were adopted by IBM Research and are being integrated in the IBM FlexoTask system and released under an open source licence.

Papers: [68, 73, 72, 39, 60, 36, 58, 67, 37, 57, 52, 38, 7, 30, 90, 40, 20, 5, 56, 3, 61, 65, 54, 66, 55, 4, 2, 22, 6, 44, 91, 53].

Service: I have been the PC chair and a member of the Steering Committee of the Java Technologies for Real-time and Embedded Systems (JTRes) workshop, since 2005. I am serving of the Java Community Process expert group for JSR-302 "Safety Critical Java" which will set the standard for safety critical extensions to the RTSJ. I have been on the PC of real-time conferences such as the International Real-Time Systems Symposium (RTSS) and the IEEE/IFIP International Conference On Embedded and Ubiquitous Computing (EUC), the Workshop on Concurrency, Real-Time and Distribution in Eiffel (CORDIE).

Foundation of Encapsulation in Object-Oriented Systems

Object identity is the foundation of object-oriented programming. Objects are useful for modeling application domain abstractions as their identity always remains the same during the execution of a program. Even if the state or behavior of an object changes, the object always represents the same phenomenon in the application domain. However, identity causes practical problems due to the presence of *aliasing*, when a particular object can be accessed through more than one reference. A change to a specific object can affect any number of objects that refer to it, even though the object may have no information about the referring objects themselves. Aliasing has a large impact on the process of developing software systems. In the presence of aliases, reasoning about the behavior of the system becomes extremely difficult. Especially in combination with concurrency, we are faced with problems that are beyond the reach of current technologies such as model checking, or program analysis.

I helped invent the concept of *ownership types*, the seemingly simple idea of using static declarative constraints to impose a topology on the runtime heap of an object-oriented program. Since our 1998 ECOOP paper on the topic, there were over 100 papers on variants of ownership types in major conferences and hundreds of references. Our original proposal introduced the idea of parameterizing class definition with ownership annotations. These annotations were then used to enforce a structure on the heap. When an object is the owner of another one, it act as a dominator in the object graph induced by object references. While ownership types have proven to be a successful vehicle for experimenting with new language features, practical adoption has been hampered by the programmer overhead inherent to the fine-grained annotations needed to define an encapsulation policy. This observation led me to work on lighter-weight notions of confinement. Rather than aiming for the most expressive confinement mechanism, I have looked for the least disruptive one. A mechanism that require as few changes as possible to the tool chain (compilers, verifiers, virtual machines, etc.) and the smallest possible changes to the program. Ideally, the confinement should simply codify software engineering best practice already familiar to programmers. My Confined types proposal is a non-trivial encapsulation property which requires very few annotations. Furthermore we have shown that confinement can be inferred. Using inference on over 100MB of Java bytecode shows that 32% of the candidate classes are confined with no programmer intervention. This is encouraging as it suggests that experienced programmers tend to write code that does not violate confinement. It is thus reasonable to expect that confinement can be widely adopted as a way to reinforce good programming style. In more recent work, I have proposed a simple ownership discipline to enforce thread-locality in Java. This proposal lets programmers annotate objects as thread-local and enforces that such objects will never be referenced from a different thread.

Our latest work on ownership has focused on its application in real-time Java where ownership is the key to static type safety of region-based memory management. In this system, type annotations are used to declare life-time of region-allocated data and prevent dangling pointers. We have found a very simple programming model that requires almost no programming annotations. The programming model was validated by refactoring parts of 100K lines of code Real-time Java CORBA ORB to abide by the ownership type system constraints. As of this writing, our type system is being considered for inclusion in the standard for safety critical Java.

Papers: [46, 69, 67, 87, 88, 5, 90, 65, 3, 2, 29, 93, 91, 92, 28, 74, 9, 47]

Service: I founded the International Workshop on Aliasing, Confinement and Ownership (IWACO) in 1999 and was on its PC several times. After ten years the workshop is still a meeting place for researchers in ownership types. I organized of the Dagstuhl Seminar on Types for Tools: Applications of Type Theoretic Techniques in June of 2005. I was a PC chair of the European Conference on Object-Oriented Programming (ECOOP) and the Formal Techniques for Java-like Programs (FTfJP) workshop. I was on the PC of conferences such as the ACM SIGPLAN Conference on Principles of Programming Languages (POPL) and the European Symposium on Programming (ESOP) and the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA).

Language Implementation and Virtual Machine Technologies

Empirical experimentation is essential dependent on the availability of high-quality open infrastructures. In programming language research one needs open source compilers and virtual machines. I initiated the development of a new *virtual machine framework* called Ovm. This framework is the vehicle of all of the research in my group and the basis of our real-time Java virtual machine. Over the last 6 years over 20 man/years were invested to produce a system that is competitive with commercial Java virtual machines such as SUN's Hotpost. From a research position, Ovm is an experiment in object-oriented software engineering, as we wanted a highly-componentized system that can be customized to different application domains. To achieve this we had to push the limits of the object-oriented model and borrow some ideas from Aspect Oriented programming. As of this writing, the Ovm code base has grown to over 150,000 lines of code. It is is composed of a number of tools and subsystems (interpreter, verifier, rewriters, compilers) which are all written in Java.

Object-oriented languages present implementers with practical challenges. My contributions in this field were algorithms for two of the most frequent operations performed by object-oriented programs, namely method dispatch and subtype tests. *Method dispatch* is a fundamental operation in object-oriented programs. For every call site, the instruction sequence to be executed is selected at runtime based on the class of the first argument. Since the early days of object-oriented languages, the algorithms used for method dispatch were an important source of inefficiency. Nigel Horspool and I published a paper proposing a compact solution to the dispatching problem in dynamically typed languages such as Smalltalk in '94 and two years later came up with an algorithm that further reduced the space requirements of our solution, This work was subsequently referenced and extended by other researchers. *Dynamic subtype tests* are another frequent operation in object-oriented programs. They are executed in a variety of contexts, such as checked casts, array updates, exception handling and `instanceof` queries. While subtype tests can be performed in linear time by traversing the type hierarchy, this is not acceptable in practice. The research question is thus to find data structures, optimized for space and time, for answering reachability queries on an acyclic ordered graph. In '97 we proposed a compact bitset encoding of the type hierarchy based on graph coloring. We also invented a constant time algorithm that adapted my method dispatch work to the task of compacting a matrix encoding of the type hierarchy. In '03, we presented a constant time subtype test algorithm with a very low space footprint and fast recomputation times. This algorithm is suited to systems requiring predictable performance and dynamic loading. We also demonstrated that the same technique can be used to implement the write barriers needed in Real-time Java.

Papers: [1, 54, 8, 17, 55, 4, 89, 22, 49, 51, 48, 50, 10, 43, 80, 82, 25, 81, 83, 32, 23, 21, 42]

Service: I was the first Program Chair of the ACM/USENIX Conference on Virtual Execution Environments (VEE) in 2005. This conference is an important venue for virtualization research in both programming languages and operating systems. I have served on the program committees of related conferences and workshops such as the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), the International Conference on Compiler Construction (CC), the Workshop on Virtual Machines and Intermediate Languages (VMIL), and the Implementation, Compilation, Optimization of Ob ject-Oriented Languages, Programs and Systems (ICOOOLPS). Ovm has been released under an open source license, and used by Boeing and a number of academic research projects.

Information Assurance and Security

Host-based intrusion detection systems attempt to identify attacks by discovering program behaviors that deviate from expected patterns. While the idea of performing behavior validation on-the-fly and terminating errant tasks as soon as a violation is detected is appealing, existing systems exhibit serious shortcomings in terms of accuracy and/or efficiency. With Rajeev Gopalakrishna and Eugene Spafford, we implemented the first host-based intrusion detector which does not require human intervention and is efficient enough to be practical. Our approach is based on a static analysis algorithm for constructing a flow- and context-sensitive model of a program that allows for efficient online validation. Context-sensitivity is essential to reduce the number of impossible control-flow paths accepted by the intrusion detection system because such paths provide opportunities for attackers to evade detection. An important consideration for on-the-fly intrusion detection is to reduce the performance overhead caused by monitoring. Compared to the existing approaches, our inlined automaton model (IAM) presents a good tradeoff between accuracy and performance. On a 32K line program, the monitoring overhead is negligible.

With Peter Sewell, I looked at the formal foundation of *secure software composition*, focusing on the rigorous statement and proof of their security properties. To express and reason about components we developed a small programming language that allows the composition of concurrently-executing software components and supports the enforcement of security policies. The essential aspects of the problem were abstracted in a process calculus: the box-π calculus, a minimal extension of the π-calculus with encapsulation. In this setting we were able to prove secrecy for non-trivial wrappers by a type system capable of capturing causality. With Dominic Duggan and Tom Chotia, I defined distributed access control as a weak form of information flow control. We associate access restrictions with data in perpetuity as the data propagates through the system. As data is produced based on the content of other data, the produced data inherits the access restrictions of the consumed data, and may have additional access restrictions. This is weaker than information flow control as we deliberately ignore "covert channels," focusing only on the enforcement of access control restrictions. Also, rather than enforcing mandatory access control, distributed access control is discretionary, in the sense that principals can collaborate to release previously classified data. The motivation for distributed access control is to enforce *accountability*, tracking which principals are responsible for allowing access to particular data. The contribution of our work is to show how network security can be moved out of the TCB into the application, in a system where distributed access control is enforced through the type system. We use the type system to enforce access control in a distributed environment and allow applications to secure network communication themselves using cryptographic techniques.

In earlier work, I designed a secure coordination model for distributed systems. The system, Secure Spaces, extends the coordination language Linda with fine-grained access control. Secure spaces answer to the difficulty of engineering a comprehensive security architecture that enforces the security requirements of a variety of applications without being overly restrictive. Instead of enforcing a single specific security policy, we chose to define a set of simple mechanisms that can be used by application logic to implement a range of security policies. The core idea of secure spaces is simple: we protect every field of a tuple with a lock. Locks prevent unauthorized processes from gaining access to the data held in the fields of a tuple.

Papers: [59, 27, 16, 15, 77, 64, 63, 78, 62, 12, 85, 86].

Service: I have served on the PC of related conferences and workshops such as the IEEE Computer Security Foundations Symposium (CSF), the Symposium on Access Control Models and Technologies (SACMAT), the Workshop on Programming Languages Analysis for Security (PLAS), the International Workshop on Security Issues in Coordination Models, Languages and Systems, the Workshop on Program Analysis for Security and Safety (PASSWORD) and the Distributed Ob ject Security Workshop (DOS).

References

[1] Terra: a multi-stage language for high-performance computing. In *Proceedings of the ACM Programming Language Design and Implementation Conference (PLDI)*, 2013.

[2] Chris Andreae, Yvonne Coady, Celina Gibbs, James Noble, Jan Vitek, and Tian Zhao. Scoped Types and Aspects for Real-Time Java. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, pages 124–147, Nantes, France, July 2006.

[3] Chris Andreae, Yvonne Coady, Celina Gibbs, James Noble, Jan Vitek, and Tian Zhao. Scoped types and aspects for real-time Java memory management. *Realtime Systems Journal*, 37(1):1–44, October 2007.

[4] Austin Armbuster, Jason Baker, Antonio Cunei, David Holmes, Chapman Flack, Filip Pizlo, Edward Pla, Marek Prochazka, and Jan Vitek. A Real-time Java virtual machine with applications in avionics. *ACM Transactions in Embedded Computing Systems (TECS)*, 7(1):1–49, 2007.

[5] Joshua S. Auerbach, David F. Bacon, Rachid Guerraoui, Jesper Honig Spring, and Jan Vitek. Flexible task graphs: a unified restricted thread programming model for Java. In *Proceedings of the SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'08)*, pages 1–11. ACM, June 2008.

[6] Jason Baker, Antonio Cunei, Chapman Flack, Filip Pizlo, Marek Prochazka, Jan Vitek, Austin Armbuster, Edward Pla, and David Holmes. A real-time Java virtual machine for avionics. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2006.

[7] Jason Baker, Antonio Cunei, Tomas Kabilera, Filip Pizlo, and Jan Vitek. Accurate garbage collection in uncooperative environments revisited. *Concurrency and Computation: Practice and Experience*, 2009.

[8] Jason Baker, Antonio Cunei, Filip Pizlo, and Jan Vitek. Accurate garbage collection in uncooperative environments with lazy pointer stacks. In *International Conference on Compiler Construction (CC)*, 2007.

[9] Boris Bokowski and Jan Vitek. Confined Types. In *Proceedings 14th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, Denver, Colorado, USA, November 1999.

[10] Quetzal Bradley, R. Nigel Horspool, and Jan Vitek. Jazz: Tailored compression of Java bytecode. In *Proceedings of the IBM CASCON'98 Conference*, Mississauga, Ontario, Canada, 1998.

[11] C. Bryce, M. Oriol, and J. Vitek. A Coordination Model for Agents Based on Secure Spaces. In *Proceedings 3rd Int. Conference on Coordination Models and Languages (COORDINATION)*, pages 4–20, Amsterdam, Netherland, April 1999.

[12] Ciarán Bryce and Jan Vitek. The JavaSeal mobile agent kernel. *Autonomous Agents and Multi-Agent Systems*, 4(4):359–384, December 2001.

[13] Guiseppe Castagna, Jan Vitek, and Franseco Zappa Nardeli. The seal calculus. *Information and Computation*, 201(1), August 2005.

[14] Tom Chothia, Dominic Duggan, and Jan Vitek. Type-based distributed access control. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW)*, June 2003.

[15] Tom Chothia, Dominic Duggan, and Jan Vitek. Principals, policies and keys in a secure distributed programming language. In *Foundations of Computer Security (FCS)*, Turku, Finland, July 2004.

[16] Yvonne Coady, Celina Gibbs, Michael Haupt, Jan Vitek, and Hiroshi Yamauchi. Towards a domain specific language for virtual machines. In *First Domain-Specific Aspect Languages Workshop*, October 2006.

[17] Bogdan Cărbunar, M. T. Valente, and Jan Vitek. Coordination and mobility in corelime. *Mathematical Structures in Computer Science*, 14(3), 2004.

[18] Bogdan Cărbunar, Marco Tulio Valente, and Jan Vitek. Lime revisited. In *Proceedings of the 5th International Conference on Mobile Agents (MA)*, pages 54–69, 2001.

[19] Antonio Cunei, Rachid Guerraoui, Jesper Honig Spring, Jean Privat, and Jan Vitek. High-performance transactional event processing. In *International Conference on Coordination Models and Languages (COORDINATION)*, June 2009.

[20] Antonio Cunei and Jan Vitek. PolyD: a flexible dispatching framework. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, pages 487–503, October 2005.

[21] Antonio Cunei and Jan Vitek. A new approach to real-time checkpointing. In *Proceedings of the 2nd International Conference on Virtual Execution Environments (VEE)*, pages 68–77, Ottawa, Ontario, Canada, 2006.

[22] Antonio Cunei and Jan Vitek. An efficient and flexible toolkit for composing customized method dispatchers. *Software–Practice & Experience*, 38(1):33–73, January 2008.

[23] Delphine Demange, Vincent Laporte, Lei Zhao, Suresh Jagannathan, David Pichardie, and Jan Vitek. Plan b: a buffered memory model for java. In *Symposium on Principles of Programming Languages (POPL)*, pages 329–342, 2013.

[24] Karel Driesen, Urs Hölzle, and Jan Vitek. Message dispatch on pipelined processors. In Walter G. Olthoff, editor, *Proceedings of the 9th European Conference of Object-Oriented Programming (ECOOP)*, pages 253–282, Åarhus, Denmark, 7–11 August 1995.

[25] Rajeev Gopalakrishna, Eugene H. Spafford, and Jan Vitek. Efficient intrusion detection using automaton inlining. In *IEEE Symposium on Security and Privacy*, pages 18–31, 2005.

[26] Christian Grothoff, Jens Palsberg, and Jan Vitek. Encapsulating objects with confined types. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, pages 241–255, Nov 2001.

[27] Christian Grothoff, Jens Palsberg, and Jan Vitek. Encapsulating objects with confined types. *Transactions on Programming Languages and Systems*, 29(6):32–73, 2007.

[28] Thomas Henties, James Hunt, Doug Locke, Kelvin Nilsen, Martin Schoeberl, and Jan Vitek. Java for safety-critical applications. In *Certification of Safety-Critical Software Controlled Systems (SafeCert)*, March 2009.

[29] Martin Hirzel, Nathaniel Nystrom, Bard Bloom, and Jan Vitek. Matchete: Paths through the pattern matching jungle. In *10th International Symposium on Practical Aspects of Declarative Languages (PADL)*, pages 150–166. Springer, 2008.

[30] R. Nigel Horspool and Jan Vitek. Static analysis of Postscript code. In *Proceedings of the 1992 International Conference on Computer Languages (ICCL)*, pages 14–23, 1992.

[31] Suresh Jagannathan, Marek Prochazka, Filip Pizlo, and Jan Vitek. Transactional lock-free objects for real-time Java. In *Workshop on Synchronization and Currency in Java Programs (CSJP)*, 2004.

[32] Suresh Jagannathan and Jan Vitek. Optimistic concurrency semantics for transactions in coordination languages. In *Proceedings of the International Conference on Coordination Models and Languages (COORDINATION)*, Pisa, Italy, March 2004.

[33] Suresh Jagannathan, Jan Vitek, Adam Welc, and Antony Hosking. A transactional object calculus. *Science of Computer Programming*, 57(2):164–186, August 2005.

[34] Tomas Kalibera, Jeff Hagelberg, Petr Maj, Filip Pizlo, Ben Titzer, and Jan Vitek. A family of real-time java benchmarks. *Concurrency and Computation: Practice and Experience*, 23:1679–1700, 2011.

[35] Tomás Kalibera, Pavel Parizek, Ghaith Haddad, Gary T. Leavens, and Jan Vitek. Challenge benchmarks for verification of real-time programs. In *Programming Languages meets Program Verification Workshop (PLPV)*, pages 57–62, 2010.

[36] Tomas Kalibera, Filip Pizlo, Antony L. Hosking, and Jan Vitek. Scheduling hard real-time garbage collection. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, December 2009.

[37] Tomas Kalibera, Filip Pizlo, Antony L. Hosking, and Jan Vitek. Scheduling real-time garbage collection on uniprocessors. *ACM Trans. Comput. Syst.*, 29(3):8, 2011.

[38] Tomas Kalibera, Marek Prochazka, Filip Pizlo, Jan Vitek, Marco Zulianello, and Martin Decky. Real-time Java in space: Potential benefits and open challenges. In *Proceedings of DAta Systems In Aerospace (DASIA)*, 2009.

[39] Michal Kalpka, Rachid Guerraoui, and Jan Vitek. Stmbench7: A benchmark for software transactional memory. In *EuroSys Conference*, March 2007.

[40] Andreas Krall and Jan Vitek. On extending java. In *Proceedings of the Joint Modular Languages Conference, JMLC'97*, Linz, Austria, March 1997.

[41] Andreas Krall, Jan Vitek, and Nigel R. Horspool. Near optimal hierarchical encoding of types. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Jyvaskyla, Finland, June 1997.

[42] Jeremy Manson, Jason Baker, Antonio Cunei, Suresh Jagannathan, Marek Prochazka, Bin Xin, and Jan Vitek. Preemptible atomic regions for real-time Java. In *Proceedings of the 26th IEEE Real-Time Systems Symposium (RTSS)*, December 2005.

[43] Daniel Marino, Christian Hammer, Julian Dolby, Mandana Vaziri, Frank Tip, and Jan Vitek. Detecting deadlock in programs with data-centric synchronization. In *International Conference on Software Engineering (ICSE)*, pages 322–331, 2013.

[44] Ana Milanova and Jan Vitek. Static dominance inference. In *International Conference on Objects, Models, Components, Patterns (TOOLS'49)*, pages 211–227, 2011.

[45] James Noble, John Potter, and Jan Vitek. Flexible alias protection. In *Proceedings of the 12th European Conference on Object-Oriented Programming (ECOOP)*, Brussels, Belgium, July 1998.

[46] Krzysztof Palacz, Jason Baker, Chapman Flack, Christian Grothoff, Hiroshi Yamauchi, and Jan Vitek. Engineering a customizable intermediate representation. In *Proceedings of the ACM SIGPLAN Workshop on Interpreters, Virtual Machines and Emulators, (IVME)*, San Diego, California, June 2003.

[47] Krzysztof Palacz, Jason Baker, Chapman Flack, Christian Grothoff, Hiroshi Yamauchi, and Jan Vitek. Engineering a common intermediate representation for the Ovm framework. *The Science of Computer Programming*, 57(3):357–378, September 2005.

[48] Krzysztof Palacz, Grzegorz Czakowski, Laurent Daynes, and Jan Vitek. Incommunicado: Fast communication for isolates. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, pages 262–274, November 4–8 2002.

[49] Krzysztof Palacz and Jan Vitek. Java subtype tests in real-time. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, pages 378–404, Darmstadt, Germany, July 2003.

[50] Filip Pizlo, Ethan Blanton, Anthony Hosking, Petr Maj, Jan Vitek, and Lukas Ziarek. Schism: Fragmentation-tolerant real-time garbage collection. In *Proceedings of the ACM Programming Language Design and Implementation Conference (PLDI)*, June 2010.

[51] Filip Pizlo, Jason Fox, David Holmes, and Jan Vitek. Real-time Java scoped memory: design patterns and semantics. In *Proceedings of the IEEE International Symposium on Object-oriented Real-Time Distributed Computing (ISORC)*, Vienna, Austria, May 2004.

[52] Filip Pizlo, Athony L. Hosking, and Jan Vitek. Hiearchical real-time garbage collection. In *Proceeedings of ACM SIGPLAN/SIGBED 2007 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 123–133, 2007.

[53] Filip Pizlo and Jan Vitek. An empirical evalutation of memory management alternatives for Real-time Java. In *Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS)*, December 2006.

[54] Filip Pizlo and Jan Vitek. Memory management for real-time java: State of the art. In *Proceedings of the IEEE International Symposium on Object-oriented Real-Time Distributed Computing (ISORC)*, Orlando, FL, May 2008.

[55] Filip Pizlo, Lukasz Ziarek, Ethan Blanton, Petr Maj, and Jan Vitek. High-level programming of embedded hard real-time devices. In *EuroSys Conference*, April 2010.

[56] Ales Plsek, Lei Zhao, Veysel H. Sahin, Daniel Tang, Tomas Kalibera, and Jan Vitek. Developing Safety Critical Java applications with oSCJ/L0. In *Proceedings of the International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES)*, 2010.

[57] Gregor Richards, Christian Hammer, Francesco Zappa Nardelli, Suresh Jagannathan, and Jan Vitek. Flexible access control policies with delimited histories and revocation. In *Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, 2013.

[58] Martin Schoeberl, Florian Brandner, and Jan Vitek. Rttm: Real-time transactional memory. In *Symposium on Applied Computing (SAC)*, pages 326–333, 2010.

[59] Martin Schoeberl and Jan Vitek. Garbage collection for safety critical Java. In *International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES)*, September 2007.

[60] Peter Sewell and Jan Vitek. Secure composition of insecure components. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW-12)*, June 1999.

[61] Peter Sewell and Jan Vitek. Secure composition of untrusted code: Wrappers and causality types. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW-13)*, July 2000.

[62] Peter Sewell and Jan Vitek. Secure composition of untrusted code: Box-π, wrappers and causality types. *Journal of Computer Security*, 11(2):135–188, 2003.

[63] Jesper H. Spring, Jean Privat, Rachid Guerraoui, and Jan Vitek. StreamFlex: High-throughput stream programming in Java. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, October 2007.

[64] Jesper Honig Spring, Filip Pizlo, Rachid Guerraoui, and Jan Vitek. Reflexes: Abstractions for highly responsive systems. In *Proceedings of the 2nd International Conference on Virtual Execution Environments (VEE)*, 2007.

[65] Daniel Tang, Ales Plsek, and Jan Vitek. Static checking of safety critical Java annotations. In *Proceedings of the International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES)*, 2010.

[66] Daniel Tang, Ales Plsek, and Jan Vitek. Memory safety for safety critical java. In M. Teresa Higuera-Toledano and Andy J. Wellings, editors, *Distributed, Embedded and Real-time Java Systems*, pages 235–254. Springer, 2012.

[67] Mandana Vaziri, Frank Tip, Julian Dolby, Christian Hammer, and Jan Vitek. A type system for data-centric synchronization. In *Proceedings of the European Conference on Object Oriented Programming (ECOOP)*, pages 304–328, 2010.

[68] Jan Vitek. New paradigms for distributed programming. In *European Research Seminar in Advanced Distributed Systems, ERSADS'97*, Zinal, Switzerland, March 1997.

[69] Jan Vitek. New paradigms in distributed computing. In *Proceedings of the European Research Seminar in Advanced Distributed Systems (ERSADS)*, pages 117–122, Zinal, Switzerland, March 1997.

[70] Jan Vitek. Virtualizing real-time embedded systems with java. In *Proceedings of the Design Automation Conference (DAC)*, pages 906–911, 2011.

[71] Jan Vitek. Atomicity in real-time computing. In Karin Breitman and Nigel Horspool, editors, *Patterns, Programming and Everything*. 2012.

[72] Jan Vitek and Boris Bokowski. Confined types in Java. *Software Practice & Experience*, 31(6):507–532, 2001.

[73] Jan Vitek and Ciarán Bryce. The JavaSeal mobile agent kernel. In *Proceedings of the 1st International Symposium on Agent Systems and Applications / 3rd International Symposium on Mobile Agents (ASA/MA)*, pages 103–116, October 1999.

[74] Jan Vitek, Ciarán Bryce, and Walter Binder. Designing JavaSeal, or how to make Java safe for agents. Technical report, In *Electronic Commerce Objects*, D. Tsichritzis (Ed.), Centre Universitaire d'Informatique, University of Geneva, July 1998.

[75] Jan Vitek, Ciarán Bryce, and Manuel Oriol. Coordinating processes with secure spaces. *Science of Computer Programming*, 46(1-2):163–193, 2003.

[76] Jan Vitek and Giuseppe Castagna. Mobile agents and hostile hosts. In *Proceedings of the 10th JFLA*, Avoriaz, France, January 1999.

[77] Jan Vitek and Giuseppe Castagna. Seal: A framework for secure mobile computations. In *Internet Programming Languages*, volume 1686 of *LNCS*, Berlin, 1999. Springer Verlag.

[78] Jan Vitek, Nigel Horspool, and Andreas Krall. Efficient type inclusion tests. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, pages 142–157, October 5–9 1997.

[79] Jan Vitek and R. Nigel Horspool. Taming message passing: Efficient method look-up for dynamically typed languages. In Mario Tokoro and Remo Pareschi, editors, *Proceedings of the 8th European Conference of Object-Oriented Programming (ECOOP)*, pages 432–449, Bologna, Italy, 4–8 July 1994.

[80] Jan Vitek and R. Nigel Horspool. Compact dispatch tables for dynamically typed object oriented languages. In Tibor Gyimothy, editor, *Proceedings of the 6th International Conference on Compiler Construction (CC)*, pages 309–325, Linköping, Sweden, 24–26 April 1996.

[81] Jan Vitek, R. Nigel Horspool, and James S. Uhl. Compile-time analysis of object-oriented programs. In *Proceedings of the 4th International Conference on Compiler Construction (CC)*, pages 236–250, Paderborn, Germany, October 1992.

[82] Jan Vitek, Suresh Jagannathan, Adam Welc, and Antony L. Hosking. A semantic framework for designer transactions. In *Proceedings of the European Symposium on Programming (ESOP)*, Barcelona, Spain, April 2004.

[83] Jan Vitek and Christian Jensen (Eds.). *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1603 of *LNCS*. Springer Verlag, Berlin, 1999.

[84] Jan Vitek, Manuel Serrano, and Dimitris Thanos. Security and communication in mobile object systems. In *Mobile Object Systems: Towards the Programmable Internet*, pages 177–200, April 1997.

[85] Tobias Wrigstad, Francesco Zappa Nardelli, Sylvain Lebresne, Johan Östlund, and Jan Vitek. Integrating typed and untyped code in a scripting language. In *Symposium on Principles of Programming Languages (POPL)*, pages 377–388, 2010.

[86] Tobias Wrigstad, Filip Pizlo, Fadi Meawad, Lei Zhao, and Jan Vitek. Loci: Simple thread-locality for Java. In *Proceedings of the European Conference on Object Oriented Programming (ECOOP)*, July 2009.

[87] Hiroshi Yamauchi and Jan Vitek. Combining offline and online optimizations: Register allocation and method inlining. In *Proceedings of the Fourth ASIAN Symposium on Programming Languages and Systems (APLAS)*, Sydney, Australia, November 2006.

[88] Tian Zhao, Jason Baker, James Hunt, James Noble, and Jan Vitek. Implicit ownership types for memory management. *Science of Computer Programming*, 71(3):213–241, 2008.

[89] Tian Zhao, James Noble, and Jan Vitek. Scoped types for real-time Java. In *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS)*, Lisbon, Portugal, December 2004.

[90] Tian Zhao, Jens Palsberg, and Jan Vitek. Lightweight confinement for featherweight Java. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, pages 135–148, October 2003.

[91] Tian Zhao, Jens Palsberg, and Jan Vitek. Type-based confinement. *Journal of Functional Programming*, 16(1), January 2006.