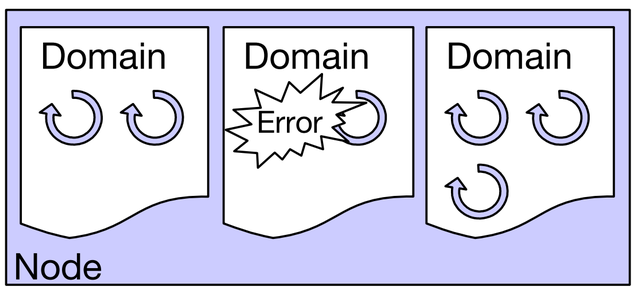
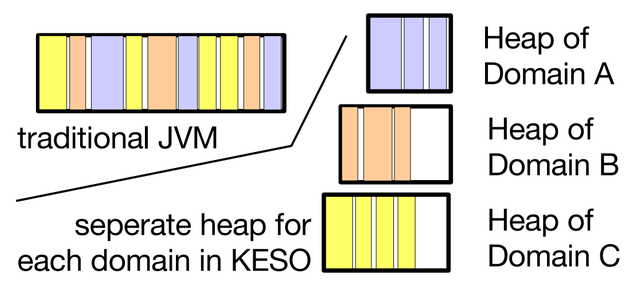
KESO • A Type-Safe Middleware for Embedded Systems

strong isolation



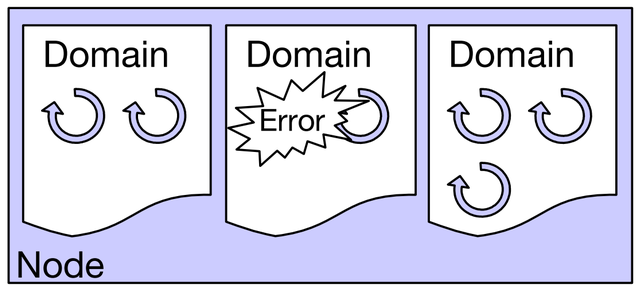
Domains are the realms of memory protection in KESO similar to processes in modern operating systems. KESO's protection is built on Java's type-safety combined with the strict seperation of the objec heaps of each domain and replication of mutuable global data in each domain. The isolation inhibits the spreading of an error and allows the safe integration of different tasks on a node.

seperated heaps



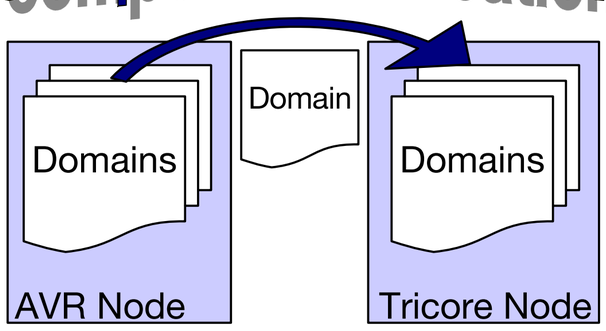
Contrary to traditional JVMs, KESO maintains a seperate heap for each domain. There are no cross-references between the heaps of two different domains, which allows garbage collection to be performed per domain and also enables the deployment of different garbage collection strategies in different domains. Memory is allocated per domain by configuring the size of the heaps.

garbage collection

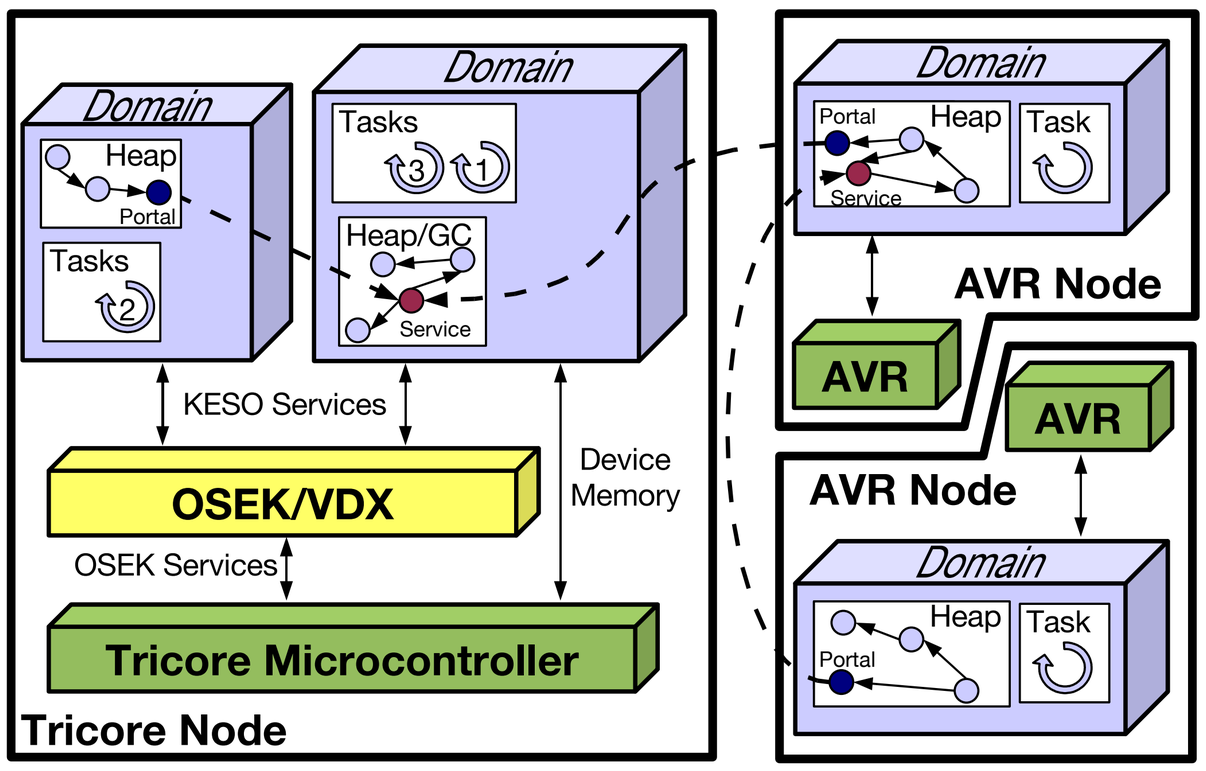


Garbage collection strategies can be configured for seperately for each domain. Domain's that do not dynamically allocate memory can run without a garbage collector and thus completely avoid the associated overhead. KESO contains a highly preemptable garbage collector with worst-case latencies (8us) lower than those of the underlying OSEK/VDX system (12 us).

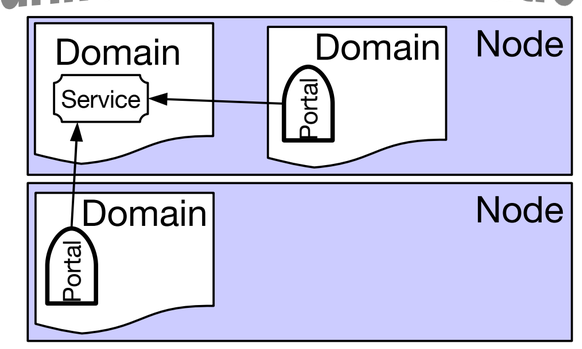
component relocation



KESO enables you to relocate domains from one node to a different node in the system by only changing the configuration. Inter-domain communication continues to work without changing the application due to the uniform portal mechanism.



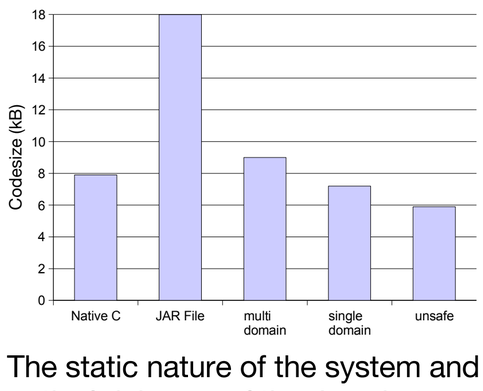
uniform communication



With portals, KESO offers a uniform and safe way of inter-domain communication. The communication mechanism is location-transparent, i.e. the client domain does not need to know on which node the service domain resides. A service is a Java interface providing a well-defined set of methods to the client domains.

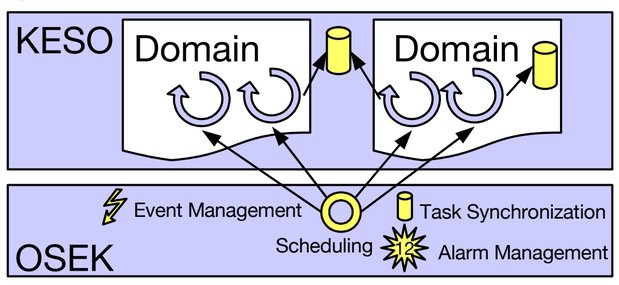
small footprint

Code Size Comparison of a Sample Application



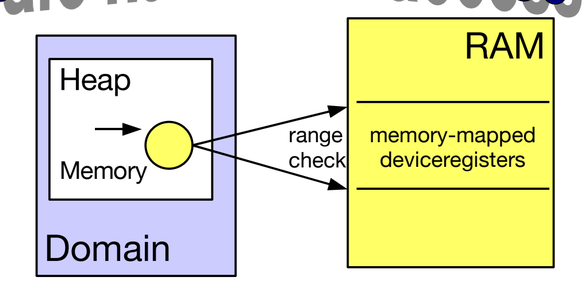
The static nature of the system and the semantical richness of the Java bytecode allow the KESO compiler to perform aggressive global optimizations. Tests with a ported C-application to KESO have shown that the resulting KESO system has a similar footprint as the original unsafe C application.

mature base



KESO is firmly based on an event-driven OSEK OS that is widely spread in the automotive industry. The priority-based scheduler of the OSEK OS is used for the scheduling of the tasks in a KESO system. Other features of the OSEK OS such as the priority-ceiling-based synchronization mechanisms are also provided by the KESO API to the Java applications, whereby KESO allows to enforce domain-based access restrictions.

safe hardware access



KESO provides special memory objects that allow to r/w access to a specific region of memory. Only primitive datatypes can be read and written through this interface. Range checks are performed on each access to prevent memory corruptions. Memory objects can also be used to provide shared memory between domains which allows the efficient exchange of large amouts of data.

Friedrich-Alexander-Universität Erlangen-Nürnberg

Department of Computer Sciences
Distributed Systems and Operating Systems
http://www4.cs.fau.de/Research

Christian Wawersich • Michael Stilkerich