Probing Organic Computing Framework [OCF] + Informatics with JVMs/Java Virtual Machines + DSLs/Domain Specific Languages w.r.t Macros/Complex Macros + Related Software Tools based on Dr.Racket Programming Language & Java -> A Simple Idea towards Testing of Hybrid Information Processing Algorithms with Deep Learning [DL] + Smart Devices [SD] + IoT + HPC Heterogeneous Computing.

[Exploring : Java + Dr.Racket based -> Language Oriented Programming -> Generate Novel Algorithms for Software R&D]

Nirmal Tej Kumar - Senior Staff Scientist/Independent Consultant - Informatics/Imaging/AI/HPC R&D.
Current Member - antE Inst UTD Dallas TX USA.
email id : hmfg2014@gmail.com

Gagik Shmavonyan - Senior Professor/Scientist/Consultant - Physics + Computation - Department of Physics.
SEUA - Yerevan - Republic of Armenia.
email id : gshmavon@yahoo.com

ABSTRACT :

As per our TITLE mentioned above,our novel idea is very interesting to try and succeed.Hence,we are presenting this simple and short technical communication.

index words/keywords : JVM,IoT HPC,Smart Devices,Organic Computing,BIGDATA.

[I] Main Idea + Inspiration + Introduction :

"Racket is a general-purpose, multi-paradigm programming language and a multi-platform distribution that includes the Racket language, compiler, large standard library, IDE, development tools, and a set of additional languages including Typed Racket (a sister language of Racket with a static type-checker), Swindle, FrTime, Lazy Racket, R5RS & R6RS Scheme, Scribble, Datalog, Racklog, Algol 60 and several teaching languages.

The Racket language is a modern dialect of Lisp and a descendant of Scheme. It is designed as a platform for programming language design and implementation.[9] In addition to the core Racket language, *Racket* is also used to refer to the family of programming languages[10] and set of tools supporting development on and with Racket.[11] Racket is also used for scripting, computer science education, and research.

The Racket platform provides an implementation of the Racket language (including a runtime system,[12] libraries, and compiler supporting several compilation modes: machine code, machine-independent, interpreted, and JIT) along with the DrRacket integrated development environment (IDE) written in Racket.[13] Racket is used by the ProgramByDesign outreach program, which aims to turn computer science into "an indispensable part of the liberal arts curriculum".[14][15]

The core Racket language is known for its extensive macro system which enables creating embedded and domain-specific languages, language constructs such as classes or modules, and separate dialects of Racket with different semantics.[16][17][18][19]"

[**Source : https://en.wikipedia.org/wiki/Racket_(programming_language)**]

"**Java** is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let programmers *write once, run anywhere* (WORA),[17] meaning that compiled Java code can run on all platforms that support Java without the need to recompile.[18] Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++, but has fewer low-level facilities than either of them. The Java runtime provides dynamic capabilities (such as reflection and runtime code modification) that are typically not available in traditional compiled languages. As of 2019, Java was one of the most popular programming languages in use according to GitHub,[19][20] particularly for client–server web applications, with a reported 9 million developers.[21]

Java was originally developed by James Gosling at Sun Microsystems. It was released in May 1995 as a core component of Sun Microsystems' Java platform. The original and reference implementation Java compilers, virtual machines, and class libraries were originally released by Sun under proprietary licenses. As of May 2007, in compliance with the specifications of the Java Community Process, Sun had relicensed most of its Java technologies under the GPL-2.0-only license. Oracle offers its own HotSpot Java Virtual Machine, however the official reference implementation is the OpenJDK JVM which is free open-source software and used by most developers and is the default JVM for almost all Linux distributions."

[**Source - https://en.wikipedia.org/wiki/Java_(programming_language)**]

"**Scala** (/ˈskɑːlɑː/ *SKAH-lah*)[8] is a strong statically typed general-purpose programming language which supports both object-oriented programming and functional programming. Designed to be concise,[9] many of Scala's design decisions are aimed to address criticisms of Java.[7] Scala source code can be compiled to Java bytecode and run on a Java virtual machine (JVM). Scala can also be compiled to JavaScript to run in a browser, or directly to a native executable. On the JVM Scala provides language interoperability with Java so that libraries written in either language may be referenced directly in Scala or Java code.[10] Like Java, Scala is object-oriented, and uses a syntax termed *curly-brace* which is similar to the language C. Since Scala 3, there is also an option to use the off-side rule (indenting) to structure blocks, and its use is advised. Martin Odersky has said that this turned out to be the most productive change introduced in Scala 3.[11] "

[**Source - https://en.wikipedia.org/wiki/Scala_(programming_language)**]

"The **OpenJIT** project is an ongoing Java™ the programming language JIT compiler project as a collaborative effort between Tokyo Institute of Technology and Fujitsu Laboratory, partly sponsored by the Information Promotion Agency of Japan. OpenJIT is a "reflective" JIT compiler in that not only it is almost entirely written in Java, but also that it bootstraps and compiles itself during execution of the user program, and compiler components coexist as first-class objects in user heap space. Thus, users can tailor and customize the compilation of classes at runtime for variety of purposes such as application-specific optimization and partial evaluation, dynamic, compiler-assisted environment adaptation of programs, debugging, language extension and experimentation, and other types of advanced compiler-based research and applications. OpenJIT even allows full dynamic update of itself by loading the compiler classes on the fly from the network.

OpenJIT is fully JDK compliant, and plugs into standard JVMs several Unix platforms such as Solaris (Sparc), Linux (x86), and FreeBSD (x86). On Linux/x86 platform, OpenJIT 1.1.14 (the current release) is faster than the JDK 1.2 classic VM compiler, runs more or less the same speed as other commercial JIT compilers on classic VM. We are releasing the project Web page which includes the release version of the backend (version 1.1.14), partially completed frontend, various papers and presentation, and documents. OpenJIT is completely free so long as it is used for non-commercial purposes. Its source, binaries, etc. can be freely distributed and modified without restriction. Information on OpenJIT can be found on www.openjit.org, that is,on this website."

[**Source - https://www.openjit.org/**]

JikesRVM -> "Jikes RVM (Research Virtual Machine) provides a flexible open testbed to prototype virtual machine technologies and experiment with a large variety of design alternatives. The system is licensed under an OSI approved license. Jikes RVM runs on many platforms and advances the state-of-the-art of virtual machine technologies for dynamic compilation, adaptive optimization, garbage collection, thread scheduling, and synchronization. A distinguishing characteristic of Jikes RVM is that it is implemented in the Java™ programming language and is self-hosted i.e., its Java code runs on itself without requiring a second virtual machine. Most other virtual machines for the Java platform are written in native code (typically, C or C++). A Java implementation provides ease of portability, and a seamless integration of virtual machine and application resources such as objects, threads, and operating-system interfaces."

[Source - https://www.jikesrvm.org/]

Jam VM -> "JamVM is an open-source Java Virtual Machine that aims to support the latest version of the JVM specification, while at the same time being compact and easy to understand."

[Source - http://jamvm.sourceforge.net/]

JADE -> "JADE (Java Agent DEvelopment Framework) is a software Framework fully implemented in the Java language. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications and through a set of graphical tools that support the debugging and deployment phases. A JADE-based system can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another, as and when required. JADE is completely implemented in Java language and the minimal system requirement is the version 5 of JAVA (the run time environment or the JDK)."

[Source - https://jade.tilab.com/]

"*Organic Computing* has emerged as a challenging vision for future information processing systems. Its basis is the insight that we will increasingly be surrounded by and depend on large collections of autonomous systems, which are equipped with sensors and actuators, aware of their environment, communicating freely, and organizing themselves in order to perform actions and services required by the users."

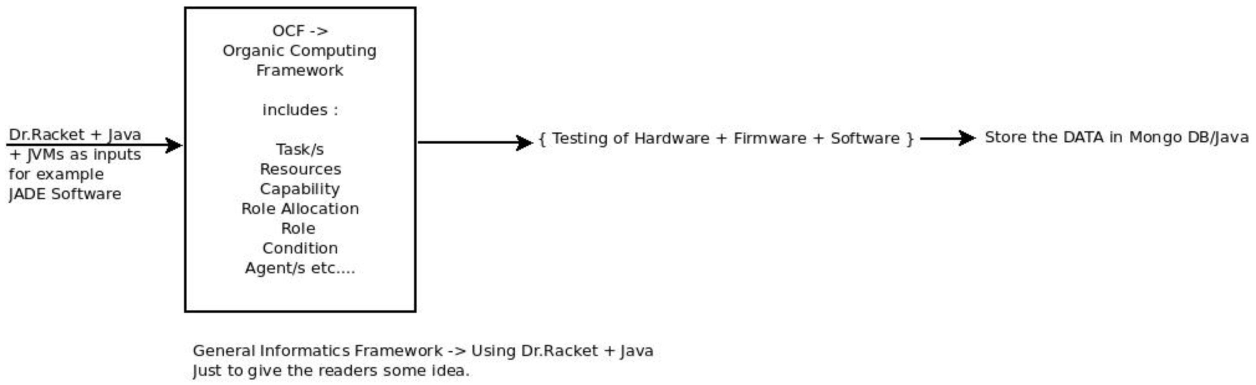
[**Source : https://vdoc.pub/documents/organic-computing-a-paradigm-shift-for-complex-systems-5f0972rjcdq0**]

"***Organic Computing Systems*** are systems which have the capability to autonomously (re-)organize and adapt themselves. The benefit of such systems with self-x properties is that they are more dependable, as they can compensate for some failures. They are easier to maintain, because they can automatically configure themselves and are more convenient to use because of automatic adaptation to new situations. While Organic Computing systems have a lot of desired properties, there still exists only little knowledge on how they can be designed and built."

[**Source -** ***A Specification and Construction Paradigm for Organic Computing Systems*** by M. Güdemann, F. Nafz, F. Ortmeier, H. Seebach and W. Reif Lehrstuhl für Softwaretechnik und Programmiersprachen,Universität Augsburg, D-86135 Augsburg {guedemann,nafz,ortmeier,seebach,reif}@informatik.uni-augsburg.de]*

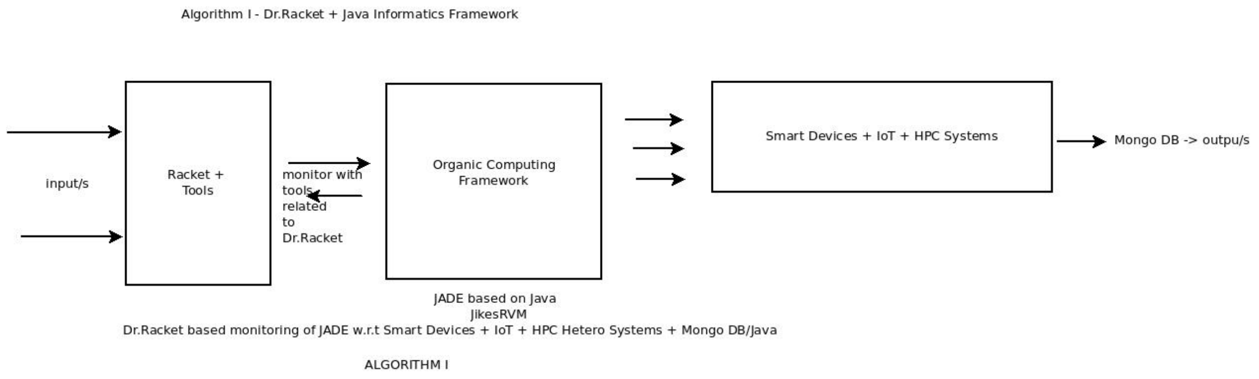
*Our inspiration is from the above mentioned paper,so to understand our present technical communication,please read the above mentioned paper.

[II] Dr.Racket + Java + JVM based Informatics Framework with Some Results + Reviews :

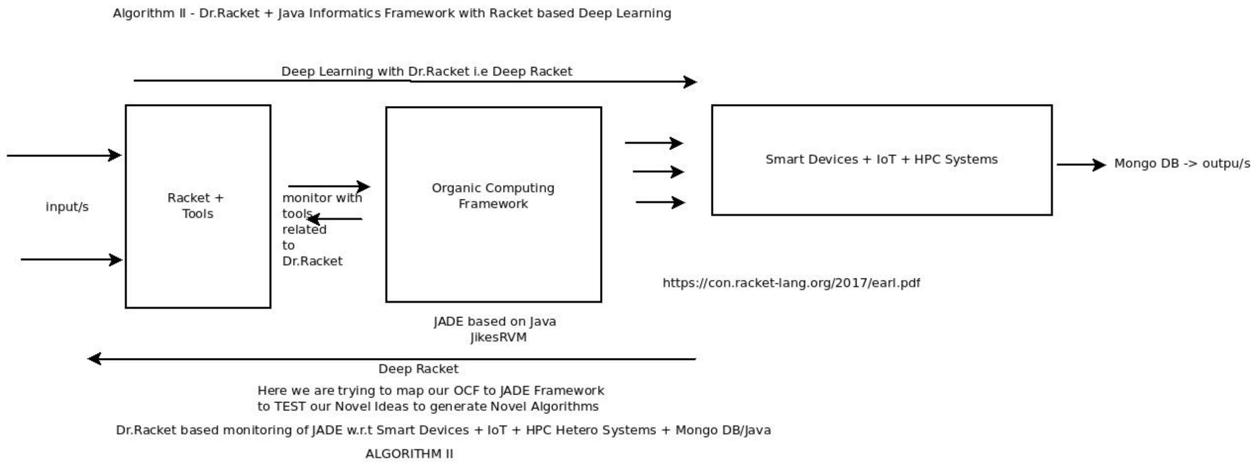

Dr.Racket + Java + JVMs as inputs for example JADE Software
OCF -> Organic Computing Framework
includes :
Task/s
Resources
Capability
Role Allocation
Role
Condition
Agent/s etc....
{ Testing of Hardware + Firmware + Software }
Store the DATA in Mongo DB/Java
General Informatics Framework -> Using Dr.Racket + Java
Just to give the readers some idea.

[Figure I - General Concept w.r.t OCF + Dr.Racket + Java + JADE Software - Approximate Algorithm]

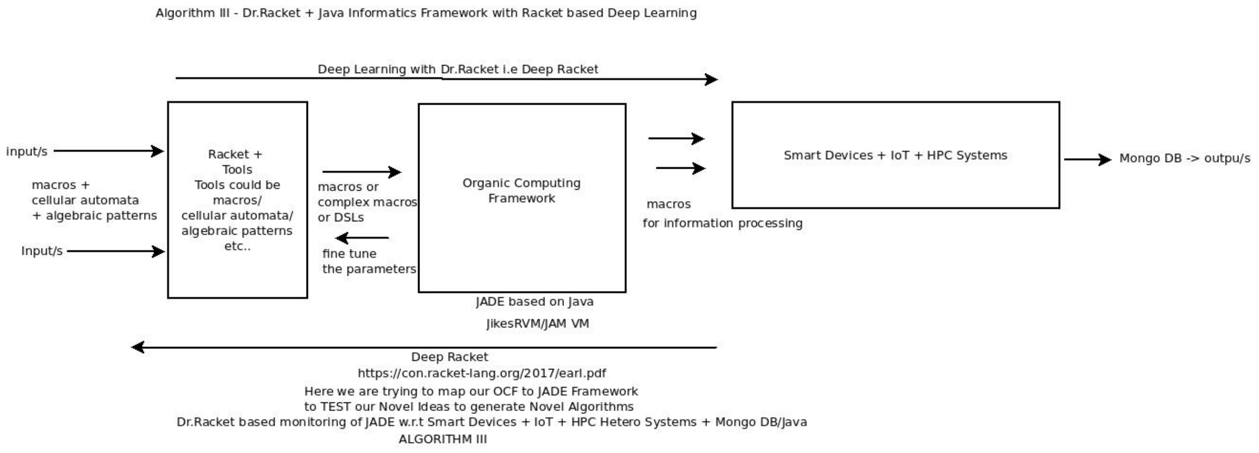
"Languages Emerge as Abstractions to Conquer Complexity".


Algorithm I - Dr.Racket + Java Informatics Framework
input/s
Racket + Tools
monitor with tools related to Dr.Racket
Organic Computing Framework
Smart Devices + IoT + HPC Systems
Mongo DB -> outpu/s
JADE based on Java JikesRVM
Dr.Racket based monitoring of JADE w.r.t Smart Devices + IoT + HPC Hetero Systems + Mongo DB/Java
ALGORITHM I

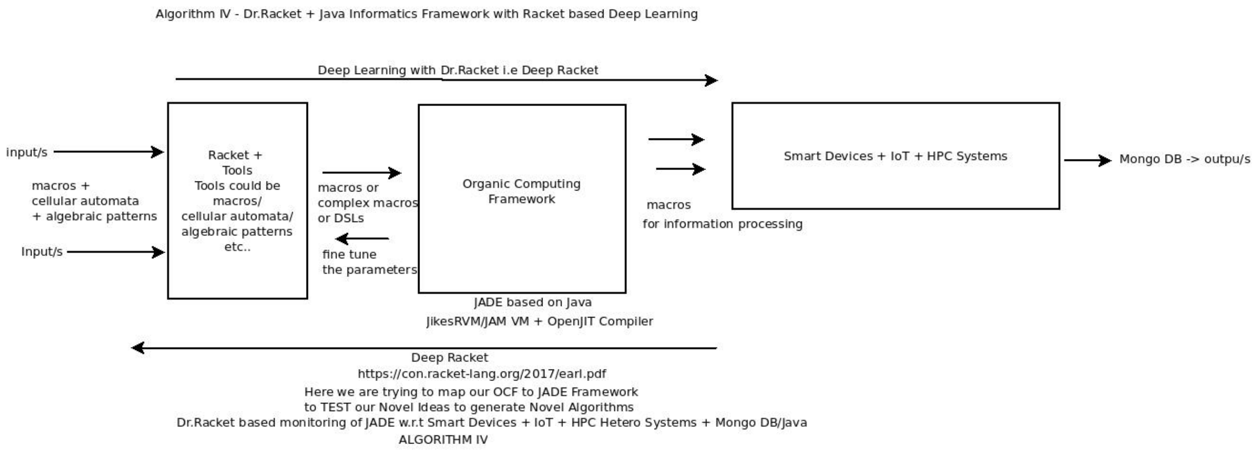
[Figure II - OCF + Dr.Racket + Java w.r.t Testing our Approximate Algorithm I]


Algorithm II - Dr.Racket + Java Informatics Framework with Racket based Deep Learning
Deep Learning with Dr.Racket i.e Deep Racket
input/s
Racket + Tools
monitor with tools related to Dr.Racket
Organic Computing Framework
Smart Devices + IoT + HPC Systems
Mongo DB -> outpu/s
https://con.racket-lang.org/2017/earl.pdf
JADE based on Java JikesRVM
Deep Racket
Here we are trying to map our OCF to JADE Framework to TEST our Novel Ideas to generate Novel Algorithms
Dr.Racket based monitoring of JADE w.r.t Smart Devices + IoT + HPC Hetero Systems + Mongo DB/Java
ALGORITHM II

[Figure III - OCF + Dr.Racket + JADE + Deep Racket - Approximate Algorithm II]


Algorithm III - Dr.Racket + Java Informatics Framework with Racket based Deep Learning
Deep Learning with Dr.Racket i.e Deep Racket
input/s
macros + cellular automata + algebraic patterns
Input/s
Racket + Tools
Tools could be macros/ cellular automata/ algebraic patterns etc..
macros or complex macros or DSLs
fine tune the parameters
Organic Computing Framework
macros for information processing
Smart Devices + IoT + HPC Systems
Mongo DB -> outpu/s
JADE based on Java
JikesRVM/JAM VM
Deep Racket
https://con.racket-lang.org/2017/earl.pdf
Here we are trying to map our OCF to JADE Framework to TEST our Novel Ideas to generate Novel Algorithms
Dr.Racket based monitoring of JADE w.r.t Smart Devices + IoT + HPC Hetero Systems + Mongo DB/Java
ALGORITHM III

[Figure IV - OCF + Dr.Racket + JADE + DEEP Racket + Macros + DSLs - Approximate Algorithm III]


Algorithm IV - Dr.Racket + Java Informatics Framework with Racket based Deep Learning
Deep Learning with Dr.Racket i.e Deep Racket
input/s
macros + cellular automata + algebraic patterns
Input/s
Racket + Tools Tools could be macros/ cellular automata/ algebraic patterns etc..
macros or complex macros or DSLs
fine tune the parameters
Organic Computing Framework
JADE based on Java
JikesRVM/JAM VM + OpenJIT Compiler
macros for information processing
Smart Devices + IoT + HPC Systems
Mongo DB -> outpu/s
Deep Racket
https://con.racket-lang.org/2017/earl.pdf
Here we are trying to map our OCF to JADE Framework to TEST our Novel Ideas to generate Novel Algorithms
Dr.Racket based monitoring of JADE w.r.t Smart Devices + IoT + HPC Hetero Systems + Mongo DB/Java
ALGORITHM IV

[Figure V - OCF + Dr.Racket + JADE + DEEP Racket + Macros + DSLs - Approximate Algorithm IV]

Why Dr.Racket ?

Dr.Racket w.r.t LoP -> Language-Oriented Programming Features & its Advantages :

[a] Program DSLs quickly.

[b] Program in them at the same time using IDE.

[c] Connecting these programs smoothly.

[d] Make these connections SAFE + SECURE.

Further,we are also interested in probing :

[a] macros defining macros.

[b] module crossing syntax information.

[c] expander defining macros.

[d] multi-pass compilation with macros.

[e] parsing ugly-syntax into macros.

[III] Additional Information on Applied Mathematics + Software Used :

[a] https://github.com/tejdnk-2019-ShortNotes -> Plenty of examples covering many topics for your R&D Efforts.

[b] https://github.com/JikesRVM/JikesRVM ; https://github.com/lihaoyi/Metascala ;

[c] http://www.sablevm.org/ ; http://jamvm.sourceforge.net/ ;

[d] https://github.com/douchuan/jvm-> JVM Using Rust - 100% Experimental -> We are testing this as one of the pioneering efforts.

[e] https://www.openjit.org/ ; [f] https://www.rust-lang.org/ ; [g] https://llvm.org/;

[h] https://github.com/bacam/coqjvm ; [i] https://isabelle.in.tum.de/ & http://www.jiprolog.com/ ;

[j] https://www.antlr.org/ ; [k] https://polly.llvm.org/; [l] https://github.com/charlescearl/DeepRacket ;

[l] ***OpenJIT Publications for further reading :***

https://www.openjit.org/publications/ecoop2000/ecoop2000.pdf ; https://www.openjit.org/publications/lncs1826/lncs1826-openjit.pdf ;

https://www.openjit.org/publications/pro1999-08/frontend-pro-199908.pdf ;

https://www.openjit.org/publications/pro1999-06/decompiler-pro-199906.pdf ;

https://www.openjit.org/publications/oopsla98-workshop/OpenJIT-OOPSLA98.pdf ;

[IV] Acknowledgment/s :

Sincere Thanks to all WHO made this happen in my LIFE.Non-Profit R&D.Inspire others always.

[V] Conclusion/s + Future Perspectives :

One of the pioneering R&D Efforts in this interesting direction.Hope many will follow us.Based on the requests we have received w.r.t JikesRVM - Research Virtual Machine [RVM] we are presenting our simple but useful JVM based Testing of Smart Devices + IoT + HPC Systems in Hi-End Computing Environments targeting : Space + Medicine + Telecoms + High Performance Computing [HPC] Applications.

[VI] References :

[1].https://www.jikesrvm.org/

[2].https://en.wikipedia.org/wiki/Java_virtual_machine

[3].http://teavm.org/

[4].https://www.cs.cornell.edu/projects/polyglot/

[5].https://www.usenix.org/conference/hotos-xii/hera-jvm-abstracting-processor-heterogeneity-behind-virtual-machine

[6].https://www.sciencedirect.com/science/article/pii/S016764230700175X

[7].https://www.javatpoint.com/multithreading-in-java

[8].https://www.geeksforgeeks.org/differences-jdk-jre-jvm/ -> Very useful in understanding our technical notes.

[9].https://news.ycombinator.com/item?id=19232068

[10].https://racket-lang.org/

[11].https://www.whizlabs.com/blog/jvm-languages/ ;

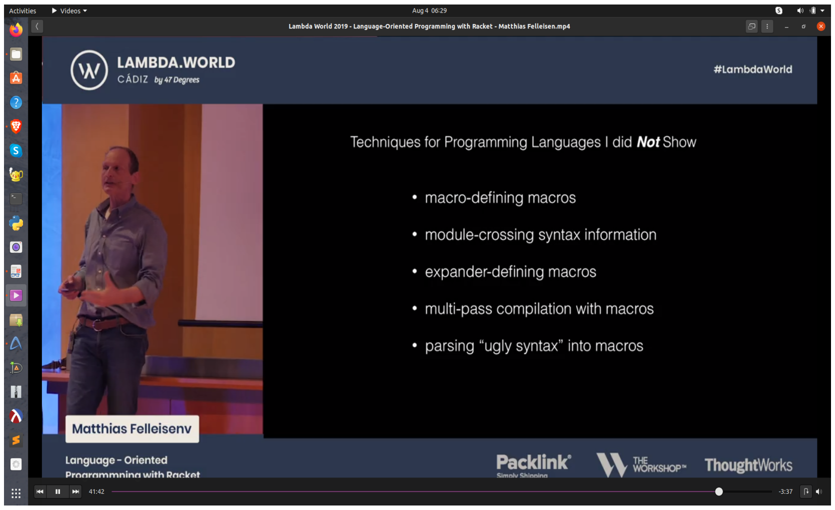
[12].https://www.organic-computing.de/

[13].https://www.semanticscholar.org/paper/Organic-Computing-A-Paradigm-Shift-for-Complex-Mller-Schloer-Schmeck

[14].https://www.networkworld.com/article/3532094/ai-everywhere-iot-chips-coming-from-arm.html

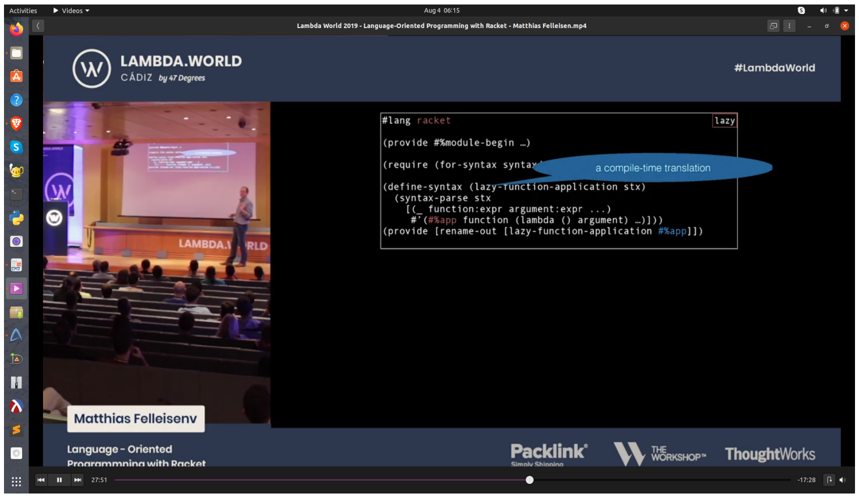
[15].https://embeddedcomputing.com/technology/software-and-os/ides-application-programming/10-programming-languages-for-iot-development-in-2021

[VII] Appendix for Reference or for encouragement only :



Techniques for Programming Languages I did ***Not*** Show

- macro-defining macros
- module-crossing syntax information
- expander-defining macros
- multi-pass compilation with macros
- parsing “ugly syntax” into macros



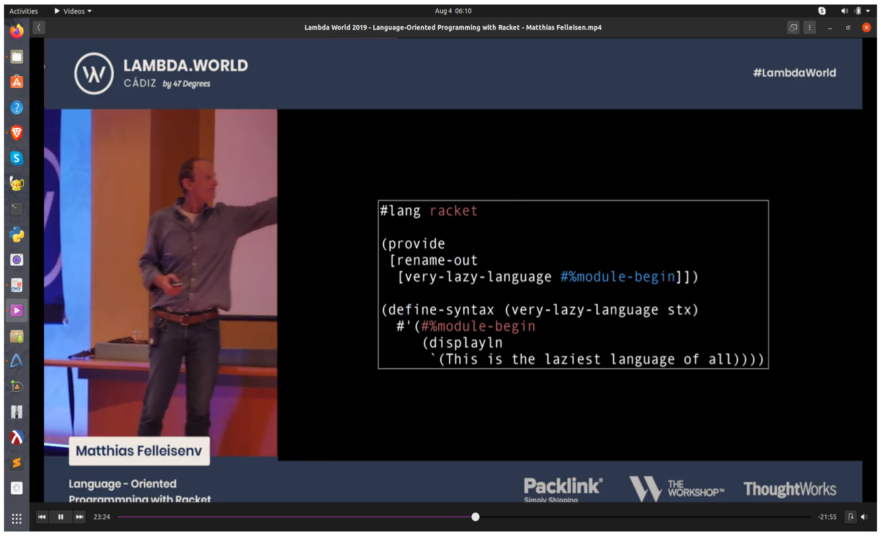
```
#lang racket                                                    lazy

(provide #%module-begin …)

(require (for-syntax syntax

(define-syntax (lazy-function-application stx)
  (syntax-parse stx
    [(_ function:expr argument:expr ...)
     #'(#%app function (lambda () argument) …)]))
(provide [rename-out [lazy-function-application #%app]])
```

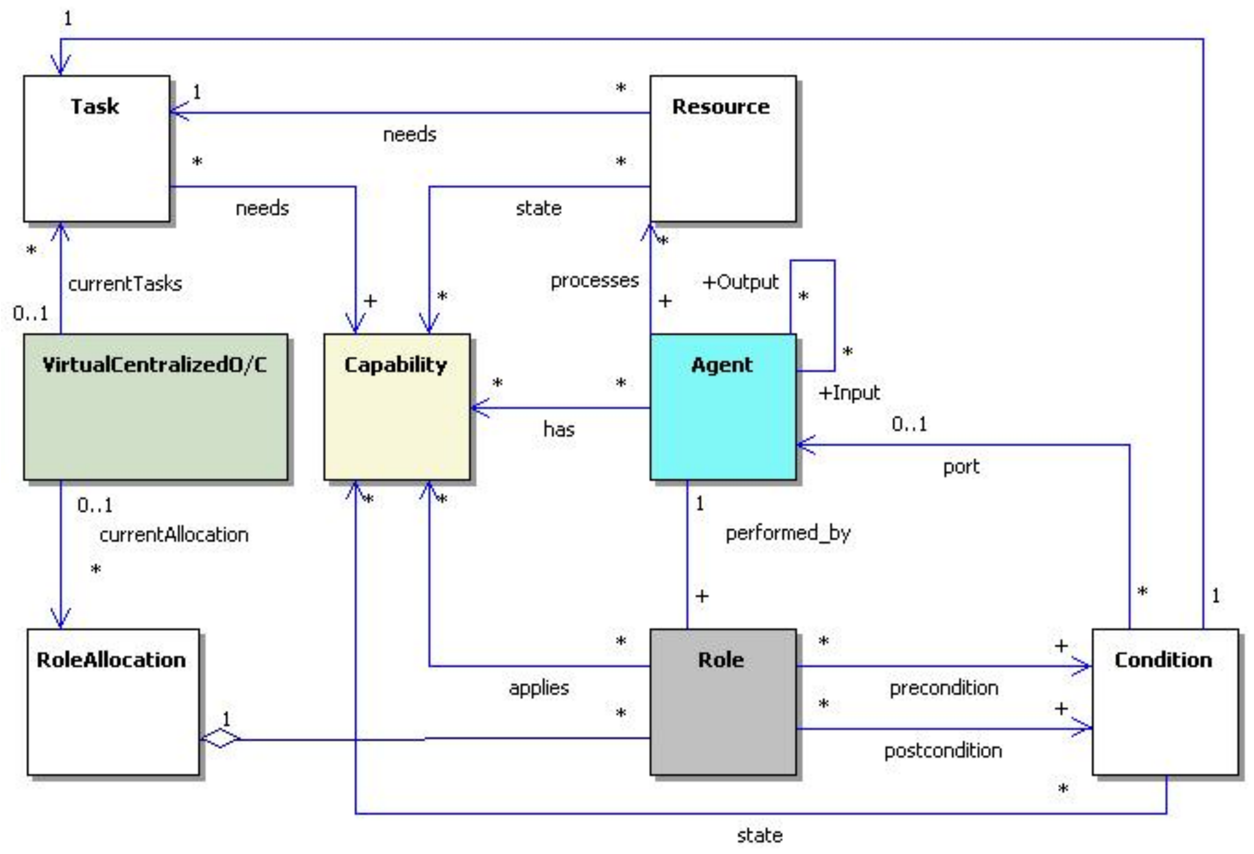
a compile-time translation



```
#lang racket

(provide
 [rename-out
  [very-lazy-language #%module-begin]])

(define-syntax (very-lazy-language stx)
  #'(#%module-begin
      (displayln
       `(This is the laziest language of all))))
```



[Figure -> This is from one of the above references to derive our own Frameworks]

[THE END]