.*i* – A Complexity Theory based Platform for Model based System Engineering

Ravindra V Joshi
Research Scholar, Department of Computer Science and Engineering, Noorul Islam Centre for Higher Education, Kumaracoil, Kanyakumari District, Tamil Nadu, India.
E-mail: rvjoshi18@hotmail.com

N. Chandrashekhar
Professor, Department of Computer Science and Engineering, Noorul Islam Centre for Higher Education, Kumaracoil, Kanyakumari District, Tamil Nadu, India.

Received September 17, 2021; Accepted December 15, 2021
ISSN: 1735-188X
DOI: 10.14704/WEB/V19I1/WEB19220

Abstract

Complexity Theory and Complex Adaptive Systems is fast emerging as optimal and efficient design alternative many of the existing technologies to address various functional anon-functional criterion. However, it remains predominantly laboratory resident software. One of the main obstacles to convert it into mainstream is its abstract terminology and black box “emergent” philosophy. In this paper an attempt is made to create a platform on the core foundation of cognitive agent and complex world concepts. The platform can be used to develop industry strength products incorporating complexity theory principles.

Keywords

System Engineering, Framework and Components, UML and SysML.

Introduction

Model Based System Engineering (Albert Albers and Christian Zingel, 2013) is here to stay. Interdisciplinary nature of modern systems, among many others, is primary force driving this philosophy. It is practically impossible to develop a system by “hand coding” and everyone has-to resort to some model development and generating the source code from those models. While this practice is very common in On-Chip software developers (model sim), physical system (octave, cosim), ui developers (.net wpf designers), even generation of code from general purpose model languages like UML and SysML is catching up. However, the algorithms and philosophy used in these systems are those of regular systems. By very definition, these systems build on each other’s functionality. But when the nature of computation becomes emergent rather than compositional, these development approaches, tools are invalidated. In this paper, we will propose a platform based on which complex adaptive systems (Cladius Gros, 2008) can be developed. We do not invent any language or tool but show how existing arena of tools can be organized so that a product can be induced with complex properties and hence can be improved in many dimensions.

In following section, we describe, different components of.i platform..i naming style was inspired from.net; i symbolises many things, first, it is present in complex numbers (x + *i* * y). Secondly, it phonetically sounds "eye" representing cognition. Finally, it represents sense of 'self', identity or consciousness. Thus, i became the choice. Section 2 introduces central premises of [.i] framework Complex World with Cognitive Agents. It puts forth how challenging problem of evolution as well cognition can be reconciled to define an integrated system engineering methodology. Section 3.0 introduces challenging aspect on other side runtime or execution environment complex systems. When the system being built is mix of real and simulated components, hardware and software libraries, how runtime should be? Section 4 introduces a concept called *iunit* basic element of [.i] framework and components of [.i] are described in terms of it. Section 5 compares [.i] framework with other popular frameworks being in practice today. Section 6 concludes the paper and outlining future work.

Core Principle – Complex World and Cognitive Agent

The central idea of the platform is shown in the figure above. Modern day systems can be visualized as evolving worlds and cognitive agents with/through it (Ravindra V. Joshi and N. Chandrashekar, 2019). While world itself does not have any sense of cognition and reacts purely according to the laws of physics. Interactions of cognitive agents are more complex. They sense the world (other agents are part of world for an agent), determine what is best for them (utility-based approach) and react according to that. Thus, world is governed by complex or evolutionary dynamics whereas agent follows cognitive dynamics.

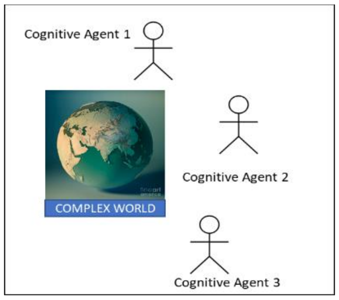

Figure 1 Complex World and Cognitive Agents

Evolutionary models particular to complex systems is treated as “Self-Organized Critical Systems” by (Per Bak *et.al*, 1988). There it is shown that networks with certain initial configuration, like game of life will evolve over time and organize themselves in such a way distribution of changes will follow Power Law pattern. They also argue that this all-pervasive phenomenon in nature, and this is how some kind of order is emerging in this chaotic universe. This idea of systems getting self-organized towards critical systems through evolution is key concept. It allows us to define certain starting conditions and rules and leave the system to operate by itself. Most of the components of the platform are designed towards making the world self-organized by (Martin V. Butz and Stewart W. Wilson, 2002).

On the other hand, cognitive agents (Masafumi Oizumi *et al.*, 2020) observe the external state and by explicitly analysing regarding the goals they need to reach, they react. This is different from evolutionary dynamics in the sense that same rule is applied on every entity. A key question that needs to be addressed is what they are observing and what awareness they are building about the situation around them. Among the hundreds of sensory perceptions that are occurring, which should be focussed on? How many pieces of information must be stitched together to call it “the correct experience”? Once such awareness comes, deciding action may be of lesser complexity and can be done by many algorithms. Integrated Information Theory 3.0 treats this problem precisely. It analyses the question how information received by a mechanism (sub-network) in a system (network) should satisfy the properties of existence, information, integration, maximalist and exclusion. A stream of input satisfying these properties at given instance of time will contribute maximally to consciousness of mechanism. The evolutionary principle of self-organized-criticality and cognitive principle of integrated information theory together lay the foundation of.i platform

i Run Time – Basic Infrastructure

A Common Run Time is the first piece of Infrastructure such platform should have. Motivation is clear. Most of the underlying platforms need runtime. Since .i is being looked as an integration platform, we have to support all those run times. When applications are built from multiple sources, they can’t be run any of their parent environments and hence we need it. This also is going to be most complex part of platform as most of the other things dealt with source language of their native platform, whereas this should both run and interpret binaries. A first attempt can be made by executing each application in their environment and by exchanging just data. Function call-return semantics also works across boundaries in most of the modern languages.

However, as performance must scale, common run time becomes mandatory Common Run Time should meet following characteristics.

1. Should be able to execute programs developed in heterogeneous platforms.
2. Symbolic Debugging should be possible for any source.
3. Minimum Garbage Collection and Managed Code support.
4. Metadata for Managed Components should be available.
5. (Stretch)Universal and Distributed Execution (like JVM) should be possible.

Other .i Components

An overview of .i components with its architectural structure is shown below. i unit is basic unit or element of .i platform. It can engage in various roles. In fact, same i unit can become domain element in one project and application element in another. However, i unit itself is immutable.

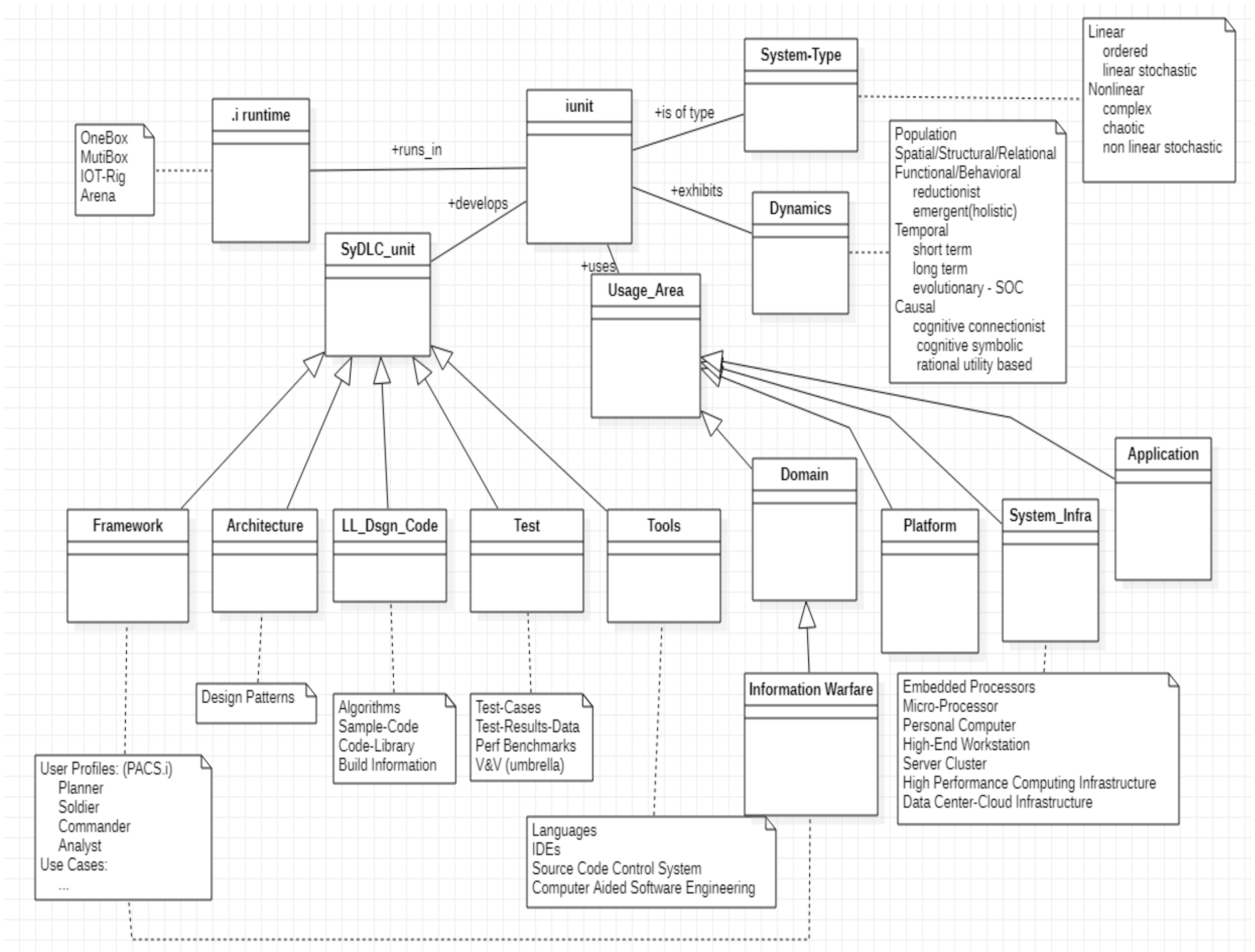

Figure 2 .i Platform Overview

Different kinds of iunits are defined below:

1. **Usage based iunits/Usage Area: .**i platform is neither too general nor too specialized. It tries fill gap in between. For example, .i expects to leverage peculiarities available to specific area. Also, final deliverable is supposed to be part of domain artifact (to be used across application), or to add to platform itself, or to add a particular system/infrastructure where applications are executed, and finally to direct application itself.

Infrastructure refers deployment architecture where final application will run. This should be supported on scale ranging (Embedded Processor) like FPGA to Cloud Centric Infrastructure. This is an essential requirement because, [.i] platform is supposed automate the process from one level of abstraction (say ABM vs Embedded Processor). In short term, the focus will be explicitly on Defense, especially in Information Warfare. So entire.i platform, the complex properties trying to be developed in products will IW centric. However, it should be noted that.i does not have implicit dependency on IW. In its framework, Domain's role is taken IW. It can be equally assumed by any other domain. But,.i platform expects one to connect to at-least a domain.

2. **.i runtime:** As explained in previous section, runtime is the essential i unit. It can be realized in many forms based on the scope of the execution. Some of the commonly used environments in Industry can be adapted to the idea of .i runtime. In addition to features to common runtimes following scenarios are envisaged.
   a. **One-Box** refers to an environment where entire system, no matter however big and complex is realized in a single application. Mostly all the functionalities are implemented as functions controlled by a single main in the application. This helps to bring algorithmic aspects.
   b. **Multi-Box:** In Multi-Box, main components of the product are deployed on different computers as per the final design. They all communicate either through Ethernet or specific protocols as per the facilities available. Idea is to take the system realistic to message flowing architecture through communication with proper protocols. This implements distributed features of the system.
   c. **IOT-Rig:** This is also called Sensor-Controller-Network. Here an array of Sensors and Controllers are connected *via* Network. The sensing, processing, and controlling is done by processors of low complexity distributed across the rig. Here basic sensing and control logics are tested. When complex equipment like are involved, this environment may become very sophisticated

like an echoic chamber, but typically simple tests are envisioned. Full scale qualifications are deferred to later environments.

d. **Arena:** Arena is a typically indoor environment, where limited movement of the objects can be verified. Tracking moving object, navigating around obstacles holding and moving objects such experiments common to robotics will be executed in such environments. Limited Flying Avenue also must be present for validation of airborne scenarios.

3. **System Type:** Systems can belong to different categories how their constituents are structured and interact. First level classification can be said to be linear vs nonlinear. These two require completely different treatments. Further grouping under each type is shown in the diagram. In one of early classic paper Dr Weaver has called out four categories of systems called Ordered, Organized Complex and Disorganized Complex. With discovery of Chaos theory disorganized can further be split into chaotic and stochastic. This platform should be able to handle each kind of system. Detailed discussion can be found under (Warren Weaver, 1948).

Sl. No.	**Attribute (Warren Weaver, 1948)**	**Complexity Properties**	**Complexity Theory Constructs**
1.	Population Dynamics	1. Generation of Population with given Statistical Distribution 2. Assignment of basic attributes – development traits 3. Based on Cellular Dynamics	Random Boolean Networks
2.	Structural Dynamics [Robustness and Scalability]	1. Robustness of a system can be measured by its connectivity, redundant connections and vulnerable centers 2. Scaling out – Adding more members without saturation 3. Self-Similar structures are naturally more scalable.	1. Complex Network Theory 2. Fractals
3a.	Temporal Dynamics-1 [Instantaneous]	1. State of the whole system aggregated together. 2. Next State is calculated by applying delta to each cell. 3. Even nonlinear signal prediction over short time should be possible.	1. Cellular Automata 2. Agent Based Model
3b.	Temporal Dynamics – 2 [Evolutionary Growth]	1. Long term changes in Complex System 2. Changes in bursts, Punctuated Equilibria 3. Self-Organized Criticality, Power Law Distribution	1. Cellular Automata 2. Graphs
4a.	Causal Dynamics-1 [Cognitive AI - data mining]	1. Mining, Recognizing Data 2. Deep Neural Networks 3. Integrated Information Theory	1. Deep Neural Networks
4b.	Causal Dynamics -2 [Explanatory AI]	1. Justification of Decisions 2. Robustness to Noise 3. Decision based on Local criteria 4. Decision based on Global Criteria	1. Lime 2. GA^2M
4c.	Causal Dynamics – 3 [Utility Based AI]	1. Single Objective Optimization 2. Multi Objective Optimization.	1. Differential Equations 2. Finite Elements Analysis 3. Soft Computing – Ant Colony Optimization

4. **SyDLC System Development Life Cycle:** All the elements defined above define an environment in which Complex Systems can be built. However, these are not sufficient. For successful design and development of systems must be available.

These are pre-built components using above concepts. Developers of i units may be researchers, open source developers, third party vendors or from own team in previous project. This section, based on Software Factory and Model Based System Engineering (MBSysE) approach, defines how pre-built knowledge should be expressed. It is also highly encouraged to develop new components confirming to new approaches.

a. **User Profile:** This is domain specific component. It lists what are the different roles, and use-cases for each role. Given autonomous nature of battle systems, profiles are made not just for humans, but also for intelligent systems. Detailed Profile is PACS. i is defined in (Lazaros Moysisa *et al.*, 2019) specifically for Information intensive mission.
b. **(High Level) Design Patterns:** Design Patterns document solutions to common problem occurring in a domain/practice-area and how they should interact. A standard way of documenting these patterns has been published with a ready catalogue of patterns.
c. **Low Level Design Code:** Low level design support consists of algorithms, sample-code and in bigger projects, library and build information
d. **Test Related:** Test Cases, Test Data and Test Results. Automation Framework and Scripts. Support for Verification and Validation
e. **Tools:** Languages and Scripts automating routine tasks. Integrated Development Environments (IDEs) stitching all tools together.

Comparative Analysis with Active Protection System as Case Study

In this section, we will compare three successful platforms with [.i] specifications. To make comparisons concrete, we will consider development of active protection system (David Adamy, 2003), a system used to protect from shells, ammunition or guided targets attacked from near-by eye of the sight sources hand-held propellers and rocket launchers. Even elevated attacks from helicopters are possible. Now, we would analyse how different requirements can be addressed each of the platforms. The three platforms considered for comparison are:

.NET: A platform for development of windows-based applications in its flavour. It is built on top of strong foundation of Common Language Run Time (CLR), Common Type System (CTS), Intermediate Language (IL) and Visual Studio (VS). Object Oriented Methodology is the foundation of library. **Xilinx Embedded Processors:** Xilinx has rich product line of embedded processors ranging from Simple Microcontrollers to complex MP-System on Chip and MP-RFSoC Using Vivado as tailorable IDE it has enabled development of very complex embedded systems in truly short life cycle. N**etLogo:** Developed in academic and still maintained in academic environment, it maintains a repository of working code with elaborate documentation and reference dictionary. It is interpreted, interactive simulation environment based once extremely popular logo language. It is causally related to Agent Based Modeling principles and hence Complex Dynamics.

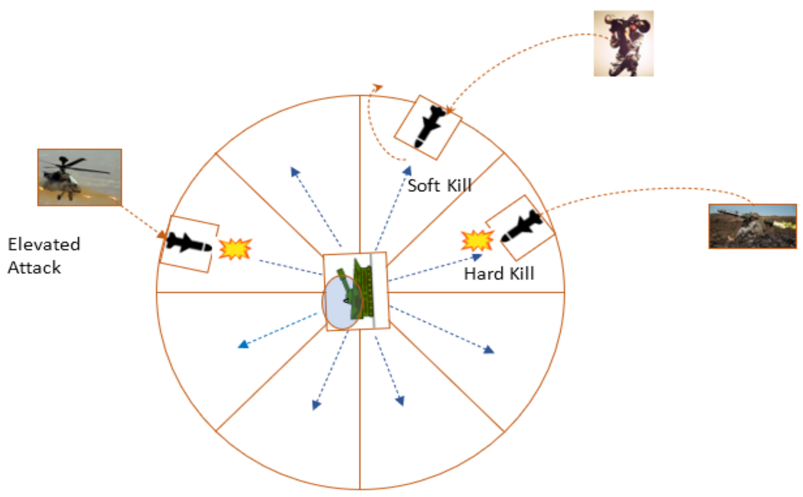

Figure 3 Active Protection System

COMPARISION OF [.i] FRAMEWORK WITH OTHER CHARTS

Table 1 Comparison of Platforms

Sl. No.	**Platform Feature**	**APS Feature**	**.NET**	**XILINX**	**NETLOGO**	**[.i]**
1.	Usage Area	Information Warfare [4]	Not bound to any usage area, domain neutral [.NET does not have built in library for Information Warfare]	Because of embedded nature, certain class of applications are supported. Verticals like automotive are explicitly supported	Not linked to any domain.	1. Demands that should be associated with any one domain. 2. Neutrality is achieved by bottom-up approach
2.	System Type	Complex	Generic [It does not have built in support for Complex system]	Ordered [Same as before]	Complex (Nonlinear)	Complex (Nonlinear)
3.	Dynamics	Temporal	Limited built in Dynamics	Third party features available	Primitive analysis methods available	Should support all dynamics
4.	Run Time	Onebox to Arena	Excellent Onebox, Multibox LOW on SCN integration	GOOD on SCN. Limited Onebox, Multi-Box	Only Onebox	Limited features on all environments
5.	User Profiles	PACS	NONE	NONE	NONE	PACS
6.	Design and Code	NA	Very Rich	Rich	Average	NONE [Existing]
7.	Test	APS Test Cases	NONE	NONE	NONE	NONE

Conclusion

It can be seen from the table above, that the parameters are not covered, esp. domain, system, dynamics related parameters are not covered by existing platforms. While they excellent towards execution side of the system, shortcomings from requirements side are many. Also, in recent times, specialized tools like ORCAD for electronic design, CAD for mechanical, 3D printing form lightweight fabrication are fast establishing themselves. The philosophy [.i] should assume is to fill the gap on requirement side rather than reinvent the wheel on execution of.NET, Vivado. Rather, it should build translators from one of these to other. NETLOGO already has demonstrated how a small, consistent solution can prove especially useful instead of trying to be all-in-all.

Next, [.i] should tackle problems of type "Complex World, Cognitive Agent" as first priority and associated concepts strongly rooted on complexity theory can take up later. Using PACS framework, different Information Centric Battle (Management) Systems should be built and benchmarks be established.

References

Ravindra, V.J., & Chandrashekhar, N. *Optimizing Probability of Intercept using Extended Classifier System.* Poster Presentation at Inter-Research-Institute Student Seminar in Computer Science which will be held on 6-7 February 2019 at Rajagiri School of Engineering & Technology, Kochi, Kerala, India

Joshi, R.V., & Chandrashekhar, N. (2018). Discrete time vs agent based techniques for finding optimal radar scan rate-a comparative analysis. *In International Conference on Soft Computing Systems*, Springer, 541-547.

Ravindra, V.J., & Chandrashekhar, N. PACS.i A complexity theory-based framework for Role-Based Battle Management.

Sun Tzu: Art of War 1st Edition, Jaico (2010).

David Adamy: EW 101: A First Course in Electronic Warfare (Artech House Radar Library), 2020.

Albers, A., & Zingel, C. (2013). Challenges of model-based systems engineering: A study towards unified term understanding and the state of usage of SysML. In *Smart Product Engineering, Springer*, 83-92.

Lazaros, M., Eleftherios, P., Christos, V., Hector, N., Ioannis, S. (2019). A Chaotic Path Planning Generator Based on Logistic Map and Modulo Tactics. *Robotics and Autonomous Systems*.

Per, B., Chao, T., & Kurt, W. Self Organized Criticality-July 1988 Physical Review A, *38*(1), 364-367.

Oizumi, M., Albantakis, L., & Tononi, G. (2014). From the phenomenology to the mechanisms of consciousness: integrated information theory 3.0. *PLoS computational biology*, *10*(5).

Warren Weaver – Science and Complexity, American Scientist, *36*(536), 1948.

Claudius Gros – Complex and Adaptive Dynamical Systems, a Primer – Springer 2008.

Butz, M.V., & Wilson, S.W. (2002). An algorithmic description of XCS. *Soft Computing*, *6*(3), 144-153.