**Introduction**

At present, advancements in software architecture have led to numerous designs of **systems** fundamentally focusing on parameters (or software architecture goals) of **Connectivity, Compatibility, Localization**, **re-usability** and **scalability** which often leads to questions of **modularity**, **Ease of use** leading to levels of abstraction of any application. Object Oriented, a programming paradigm, has proved to be imperative in today’s software industry and has become a necessity if not a norm to adopt languages that support OOP to create systems which on a large scale must maintain demands of software architecture goals. Object Oriented programming does have its disadvantages and due to that industries are leaning towards *Component-based* architecture as one of the alternatives. Narrowing down to Video Game industry, existing game engines in market have already begun to standardize software architecture in game applications by integrating component model and functionality to their *Object* superclass that allow developers to engineer their own object with modular and cohesive components. However, even *Component-based* (C-B) architecture has its limitations:

* In-order to make components widely modular, parameters needs to be **absolutely simplified** which leads to the requirement of constructing variety of interfaces for that simplified component.
* As more independent parallel executing components are mounted on an object, creating dependency and accessing those components eventually starts building problems associated with concurrency. This is normally the result of unconstrained addition of components.
* Just as programming languages restrict developers to certain programming techniques, for example JAVA language does not support multiple inheritance, typically APIs associated with framework does not provide any structured way of handling components. This encompasses dependency management and setting limit to the number of components that can be added based on specific requirements. Defining such a limit becomes important since it directly influences performance.

Discussion of this very limit demands the research into a new topic of methods to engineer components based on means of recognizing core interfaces and the reflected use of reduced dependency.

A new exploration to software architecture called State Cell and its State Map is likely to overcome limitations of both OOP and C-B architectures. It will allow and create a standard for developers to create State Cell Schematics to engineer objects according to specifications. State Cell will have following properties:

* A Cell having an objective either existing to perform a function or storage can be instructed (given instruction set) to do so while maintaining security features of instructions not being overridden.
* A Cell will have distinct number of protocol defined ports for communication with other Cells.
* Cells connected or grouped to collaborate can form higher functional bodies of defined finite order.
* A Cell can be given set or controlled level of mutation allowing it expand its instruction set given initially during runtime. This gives new properties or re-defines the body a group of connected cells are the constituents of.
* A Cell can be allowed to clone or instantiate itself with inherited instruction set.

A Cell is a small system or an object consisting of fundamental pointers to function and memory with one input and multiple specialized outputs.

#### Statement of Problem

As stated earlier, any software system utilizing an architecture must meet software architecture goals and with that, system must ideally possess following characteristics:

* Standardised
* Independent
* Composable
* Deployable
* Documented

By now it should become apparent that there is a direct relationship between software architecture and system comprising of software elements, relation and their properties that are governed by programming paradigm, models and principles (software technologies are based on that).

*Impact on Design Method*: Design method is widely dependent on development environment , organization's practices, nature or type of the software being developed, the requirements of the users, the qualification and training of the software development team, the available hardware and software resources, the availability of existing design modules, and even the budget and the time schedule. These can viewed as constraints in development process. In an attempt to overcome that, developers agree upon few common Design Methodologies: Top-Down/ Bottom-Up Design, Stepwise Refinement, Structured Design, Structured Analysis and Design Technique, Jackson Systems Development, Structured Systems Development, Object Oriented Design. Now, consider a proposition, there would exist a universal Design Methodologies that itself be provided by the mechanics used to develop the system, for instance UML diagram that is specifically made for Object Oriented design. It would then be possible to programmatically build schematic of connections between modules similar to Interface definition Language (IDL). C-B architecture already provides input/output, data flow, and data transformation, the connections to other modules and the intra-module unity so Structured Analysis and Design Technique is well suited and even becomes an apparent selection of design method. To conclude from all mentioned points, development process constraints elimination and a universal methodology can simultaneously be achieved if all emphasis be laid upon designing system mechanics that also generates a unique architecture.

Components in C-B and classes in OOP have no classification of composited objects and components. For instance, a component is referred to both a body of component made up of other components and the individual base component itself. Classification becomes important in understanding a complex system from obtaining a broader and deeper perspective. Moreover, if finer control is to be attained, levels of parent-child relationship can be developed between components and specific parents (presently accessible or perhaps in future) can be referenced. It is inevitable that quantization of anything discrete, upon miniaturizing or extremely large finite expansion of discrete components will begin to reveal entirely different properties between established scales of size or amount of components.

*Profiling methods to calculate load on application*: Various profiling third party plugins and integrated tools already exist that typically calculate total number of executions, memory used, and inclusive, exclusive, average, max and min times of **functions** in a program. In the case of Component Based Software Engineering (CBSE), profiling for individual components that has not yet emerged in market. It is relatively simple to obtain information of components from profiling functions simply within the components and eventually filtering out the log. However, if components or even object can be assumed to be load on an open loop system with an input and an output with respect to time or any other parameter, it is possible to calculate and control time complexity and resource usage which might be equivalent to time response and gain of a system. Higher order analysis can also be performed if relationship between topics of CBSE and Control Systems Engineering is developed.



Below is an illustration of a feedback loop system. Such will a scenario if in a given system Component C is responsible for adjusting the output for a given requirement. It is reasonable to explore whether the output could be a function of memory resources or time.



*Evolutionary Process* :

*Dynamic Structure* :

#### Objectives

It is clear that new approach if not modification on top of current software technologies is needed. Normally, it is the selection of architectures that must fit into the adopted software development life cycle. Being that the case, it is obvious that limitations in architecture exist mainly due to building blocks of the system. The idea is to re-define mutable components so that it automatically or perhaps logically, through its design, conform to dynamic structure that can viewed to encompass complexity in a sub dividable manner.

Of all the problem aspects discussed above, further study into those topics required before verifying whether it is feasible to it fit in the design. To achieve targeted design, experiments and analysis need to be conducted constantly during this research phase.

#### Plan of Action

State Cell and its State Map is an ideal candidate for designing system that will give rise to a new architecture and if designed to fit all aspects, a new software development process will follow through. On contrary to a single architecture, its robust elementary nature can allow user to define their own architecture thus eliminating abstraction and improving efficiency. Following described is an early concept and is subjected to refinement and expansion. An in-depth study is mandatory to uncover further details.

**STATE CELL**

Similar to a basic component, a State Cell has 3 function pointers. Refer to illustration 1A:

**Initialize:** Pointer to a function for initialization of instruction set run in the body function when source is connected. Through implemented processor, this function attached is run only once.

**Body:** Pointer to the function containing instruction set. This is the where instruction set given to the cell is run i.e. the main functionality of the cell is defined by instruction set is run here. Unless the processor receives an interrupt or instruction set itself is programmed to end functionality of the cell, the body keeps looping.

**De-Initialize:**  Pointer to a function for de-initialization of instruction set run in the body function. Execution of this pointed function marks the end of execution of cell function. The flow is then directed towards a specified port. Port is specified through State Map.

Below are end-connections of a cell, each having a specific way to interact with the connected cell:

**Port A:**  This port is called *Inheritance Connection*. When a source is connected, the body function of the connected cell through this port is executed right after the body function of the current cell is executed. Illustration 2A captures visual representation of this flow.

**Port B:** This port is called *Switched Connection*. When an interrupt is received, the source is transferred to the cell connected to this port. Illustration 2A captures visual representation of this flow.

**Port C:**  The idea of this port is only a prototype because it is not yet confirmed whether it belongs as a fundamental element. This port called *Device Connection* is believed to form data pipe between connected cells. Interrupts to other cells connected to other sources (it also means cells are executing in parallel) cause them to activate/de-activate or share data between executing body functions.

**Source:** A source is a user created thread or thread assigned by thread pool. Life time of a created thread is expected to last until the connected hierarchy of State Cells have run its course. Refer to illustration 1C for simple flow diagram.



**Figure 1A:**  State Cell model illustrating simple flow diagram of its function.

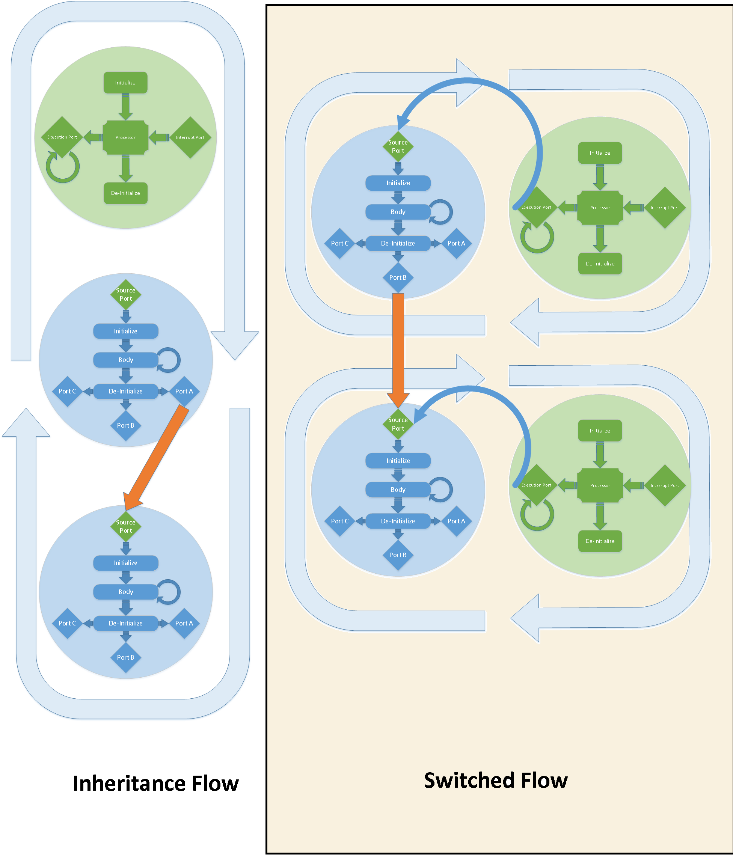
Inner working of a State Cell is describe in Illustration 1B. It is composed of connectors called **flow diverters**, function pointers and a **processor** consisting of small memory units tracking **state** and **interrupt**, **mode manager** and **identifier** that stores information on current state and port modes. Note that Port C isn’t present in this illustration. If that were to be added more internal components need to be added.



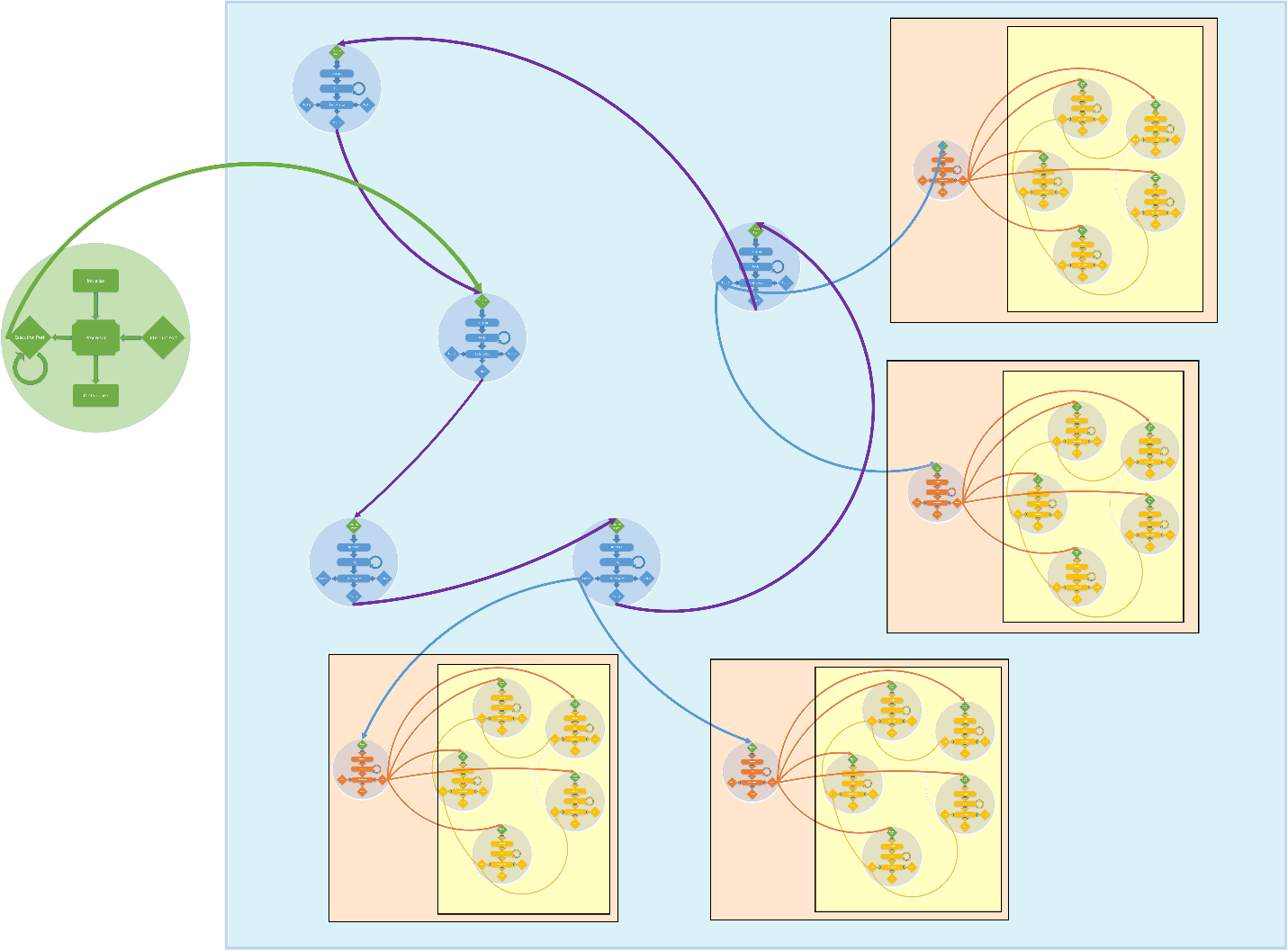
**Figure 1B:**  State Cell inner structure. Depicts flow diagram.



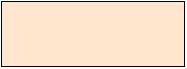
**Figure 1C:**  Source Cell model illustrating simple flow diagram of its function.

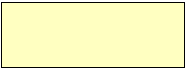


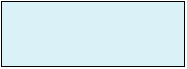
**Figure 2A:**  Inherited Flow versus Switched Flow. Note that In Switched Flow diagram, only one Source Cell exists. Two Pair of cells (horizontal to each other) shown depicts before and after (in vertical manner) Switch Connection is triggered.



**Figure 2B:**  This illustrates a simple system composed of specialised Cells. Together they form a module.

** Module:**  Cells playing the role of an interface. Cell community is connected to Port A.

** Community:**  Cells having same functionality and connected to each other sharing same data. Cells in this region are connected by properties of Port C (prototype).

** Component:** Higher order functioning Cell that control Modules. These Cells mimic different States (similar to State Machine).

#### Management Plan

#### Conclusion

#### References

Bose, D. (n.d.). *COMPONENT BASED DEVELOPMENT*. 1st ed. [ebook] INDIAN STATISTICAL INSTITUTE, pp.1-25. Available at: http://arxiv.org/ftp/arxiv/papers/1011/1011.2163.pdf [Accessed 24 Apr. 2014].

Hernández, J. and Clemente, P. (n.d.). *Aspect Component Based Software Engineering1*. 1st ed. [ebook] Spain: University of Extremadura, pp.1-4. Available at: http://www.cs.ubc.ca/~ycoady/acp4is03/papers/clemente.pdf [Accessed 24 Apr. 2014].

programming musings, (2006). *Beyond mainstream object-oriented programming*. [online] Available at: http://programming-musings.org/2006/02/05/beyond-mainstream-object-oriented-programming/ [Accessed 24 Apr. 2014].

Sei.cmu.edu, (2014). *Software Architecture | Research | Previous Architecture Research | Software Architecture Life-Cycle Integration*. [online] Available at: http://www.sei.cmu.edu/architecture/research/previousresearch/lifecycle.cfm [Accessed 24 Apr. 2014].

Spica.com, (2014). *Software architecture, scalability, modularity, connectivity and localization*. [online] Available at: http://www.spica.com/products/software-architecture.asp [Accessed 24 Apr. 2014].

Triz-journal.com, (2014). *The Laws of System Evolution*. [online] Available at: http://www.triz-journal.com/archives/2002/03/b/ [Accessed 24 Apr. 2014].

Userpages.umbc.edu, (2014). *A Survey of Major Software Design Methodologies*. [online] Available at: http://userpages.umbc.edu/~khoo/survey2.html [Accessed 24 Apr. 2014].