Control of a Complex Process:
Challenge to Create a System Infrastructure

Has your flight been grounded, medical records temporarily unavailable, work at the office halted, or purchase delayed for no apparent reason? You are not alone. The systems managing tasks in everyday life are improving and becoming more efficient at a lower cost, but they are still imperfect. Our expectations change as things work better and more often. When things go wrong, as they inevitably will, it is even more unexpected, disruptive, and upsetting.

A system can be significantly better when cost is no object. Therefore, it should be possible to find a way for any system controlling a complex process to be significantly better at a reasonable cost. The following approach is conceptually like a standard building structure or a vehicle chassis. A standard system foundation offers a way to design and implement a better system at lower cost. The potential advantages include:

- Reduce disruptions and resulting economic impact by a *factor of ten to a hundred or more*
- Reduce administration effort by a significant amount
- Reduce effort to design and build a complex software system

For the purposes of this discussion, a *controlled complex computational process* is defined as having most of the following attributes:

- Supports many human users who may be geographically distributed;
- Requires multiple computers, also typically geographically distributed and communicating over a networked infrastructure;
- Obtains input data from multiple devices and people, also geographically distributed, possibly mobile, and often served by a sometimes slow and unreliable network infrastructure;
- Requires multiple operators to intelligently direct actions during atypical situations;
- Employs one or more large databases;
- Typically involves complex computational algorithms, and/or
- Manages time-critical or resource-critical processes.

Simple tasks can be managed by one or a few people using one or a few computers. A complex process like managing the power grid, a nationwide point of sale system, a multi-site manufacturing business, or an inventory and distribution management system requires a large team of people, a lot of computers, and a highly reliable infrastructure to ensure that those people and computers are in constant contact with each other and coordinate their actions. All of this "stuff" is often widely distributed and connected via a high-speed network infrastructure. The computers that are part of such a system provide human interface capability, computational capability, database management, data acquisition, decision support applications, data archival, cyber security, process control, and various other capabilities.

Because the systems we are discussing are expensive, we want them to be long-lived. It follows that system infrastructure should be designed in an open manner to facilitate both hardware and software upgrades and enhancements. This includes the ability to design and develop new software components that can be added. Many such systems must be available 24/7/365 and they must be designed so that the system can detect errors, correct them, and possibly reconfigure itself on the fly when necessary. Of course, hardware is much more reliable than it was just a few years ago. But we all know that things still go wrong. Disk drives crash, power failures happen, the IT person still occasionally unplugs the wrong cable, tornadoes are not fake news, levees still break, backhoes still dig up fiber optic lines, and even the

best software written by the best programmers continue to exhibit bugs. A mission critical distributed system must be able to detect and compensate for both software errors and hardware failures without data loss or corruption. These requirements add significantly to the complexity of designing, developing, and testing mission critical systems.

If you need a distributed data acquisition, analysis, decision support, and process control system, there are plenty of solutions available for most every mission critical need. They are used, for example, to manage electric power generation and air traffic control. They are highly specialized, highly proprietary, and overly expensive.

One approach to improve or create a better system that costs less is to create an infrastructure that is reusable and satisfies a wide range of requirements. This approach is how a real time executive, operating system, database management system, networking infrastructure, middleware and similar products came into existence.

The Apache Hadoop project (read more about Hadoop at https://hadoop.apache.org/) has created software which performs "big data" analysis. Hadoop addresses the handling of failure conditions by separating the software performing data analysis from an infrastructure performing the tasks of splitting an analysis into pieces, assigning each piece to a computer for computation, and integrating the results. The expectation is that the infrastructure will handle all problems that occur in the communications infrastructure and the occasional loss of a computational computer.

The popular Apache Kafka project (read more about Kafka at https://kafka.apache.org/) is an infrastructure that separates application logic from a high-performance redundant data cache used to build what are called real-time data pipes and streaming applications.

The "cloud" is provided by commercial products that combine a database with leased data storage and computing. There are a myriad number of services to choose from with utterly amazing capabilities. However, the "cloud" does not supply many features needed by a mission critical system.

The challenge is to create an infrastructure suitable for most any system that controls a complex computational process. The current solutions for mission critical systems each have a proprietary infrastructure suitable for the specific mission. The challenge is to learn from these proprietary infrastructures, identify what could be improved, and create a general-purpose infrastructure adaptable for use in any mission critical system. A new, better, general-purpose infrastructure can then be added to improve existing solutions or used as a base for a new, better, and cheaper solution.

An infrastructure should address issues likely not found in any current infrastructure such as cyber security between computers within a distributed system, recovery of in progress work upon a failure, visibility of the internal workings of a distributed system, and change management without the need for a changed application to work in both an old and new way.

Cyber security tools and techniques attempt to prevent a wide range of attacks, as well as collect forensic data required to prove a crime and improve tools, techniques, and recovery strategies. However, today's techniques are imperfect; it is extremely difficult if not impossible to prevent a smart, determined, and well-funded adversary from accessing any system which is connected to a public network. In fact, it is extremely difficult to *even determine if a complex system is connected to a public network*. After gaining access, an adversary can steal data, modify data, damage the system, prevent access to it, and even hold the data for ransom. Addressing cyber security issues in today's systems is complicated by the reality that many current cyber security solutions are added on to existing systems rather than designed

and built into those systems. A distributed system should work better than a one computer system by preventing the infection of one computer from spreading to other computers within a system.

A distributed system must handle a failure condition in a predictable manner. Simply stopping operation until a normal situation returns is unacceptable. An operating system check point feature copies the state of an application to persistent storage so that it can be restarted upon a failure without losing the work accomplished up to the time of the checkpoint. However, today there is no such thing as an "application and network checkpoint". This means that there is no generally available system or networking infrastructure which can recover an application or data in transit after a failure occurs as if there was no disruption. Today, if an application or network failure occurs while two software applications are engaged in data exchange across a network, the responsibility to detect and recover from that failure is owned by the applications themselves. This is not an easy problem to solve and the solution inevitably increases the complexity (and cost) of application design, development, testing, and maintenance. As a result, this functionality is frequently poorly implemented or absent. Ultimately, this can result in a need for human intervention to understand and correct the issue. If a system infrastructure could help a backup instance of an application to recover in progress work and replace a failed instance without help from a connected application, application software would be simpler, less expensive, and more reliable.

Computers represent data and perform calculations in ways that are much different than humans. Not only that, but different computers and different computer programming languages do not even agree on formats used to represent data. So, in a distributed system with often a mix of heterogenous hardware and software, it becomes necessary to translate from Application A's representation of data to Application B's representation and often to a human understandable representation. Add Application C through Z and a database or two, and this means added application complexity and its associated cost. It would be better for an application and infrastructure to work together such that an infrastructure can do a format conversion of any information as needed.

Change control management ensures that all parts of a system will work properly together for every change that happens. If a change affects the interaction between applications on different computers, then a changed application must work in both the old and new way when 24/7/365 operation is required. This is understandably difficult and expensive to program and test. It would be better for an infrastructure to support simultaneous versions of an application where the infrastructure connects the application instances that work properly together. There would be no need for an application to work two ways which eliminates application complexity (and its cost).

Ideally a computerized control system has a multilayered defense. A computer is located within an enclosure, room, and building that provides a suitable environment including physical means to limit access. Information is encrypted into what appears to be random or white noise to an unauthorized observer. A user is authenticated via a username and password or multifactor authentication combining something a user knows with something a user possesses. Another user can be required to approve user access to a critical function. A plan to change a complex process should require reviewers to approve the plan before implementation.

A control system of a complex process should distribute work among computers at several locations. Corruption of a computer interfacing to a user must be prevented from spreading to a server computer storing data and performing critical work. Defensive functions such as authentication and access approval to a function are performed on an isolated server computer. Multiple instances of user interface and server software exist on different computers at several locations to balance a dynamic workload and to survive

a catastrophic event. Ideally in progress work of a server is recovered by an alternative computer upon most failures without requiring a user action.

A system infrastructure can help a system of computers and applications to work together as a team. System infrastructure requirements in software terminology include:

- Minimize effort to create applications that distribute, isolate, and recover work.
- Provide reasonable cost and performance.
- Enable applications to converse peer to peer, client to server, or publisher to subscriber.
- Encrypt application conversations.
- Enable a client to server conversation to be recoverable.
- Help to ensure and prove software corruption cannot spread.
- Enable an administrator to control and audit data flowing between applications.
- Enable change during operation by supporting multiple simultaneous versions and ensuring conversing parties are the same version.

An existing infrastructure likely cannot and almost certainly does not achieve all these goals. Join in to devise a new and radically different infrastructure needed to achieve these requirements. Visit web site SoftEcoSDK.com or send an email to SoftEcoSDK@gmail.com to learn about, discuss, research, and build a new kind of distributed application system infrastructure.