Artificial Intelligence in Robotic Communications, Integration, and Interoperability to Improve Situational Awareness

J. Wallace¹, S. Kambouris¹

¹Infinite Dimensions Integration, Washington, District of Columbia, United States

Abstract - A framework to integrate different artificial intelligence and machine learning algorithms is combined with an execution framework to create a powerful cloud computing system development platform. By providing an execution framework and control software that is native to cloud architectures and supports interactivity and time synchronization, the true utility of cloud computing and “big data systems” can be increased. Many “big data” software systems are not interactive, automated, or can run in real-time.

Keywords: artificial intelligence, cloud computing, communication, integration, robotics, interoperability

1 Introduction

There are three key features of this approach that are unique. First is a grammar that is capable of capturing human language and at the same time able to provide an API to computer and other electromechanical components to map language and intention to functionality. This allows any functional component to be activated or controlled based on the natural semantics of human communication. This also serves as a method for integrating a wide variety of artificial intelligence, machine learning, or other decision algorithms into a robust, complex decision making entity.

Second is the ability to synchronize the activity of components whether they are part of the same system or are in different systems, in other words location transparency that is familiar to network computing now exists for electromechanically systems. The flexible nature of the synchronization patterns allows all known interaction and control patterns to be expressed with a minimum of notation and functional components, which leads to execution efficiency and clarity of the system’s semantics. This allows precision control in a way that avoids the trap of simplistic frameworks that then require more code that is difficult to follow in order to express a typical real-world interaction. In general, this approach reduces the amount of code required to implement any particular functionality, by a factor of three.

Finally, we pose a question, “What is the essence of the human experience?”. Different researchers may provide a variety of answers, but the authors would argue that the ability to speculate about the future is certainly a foundational component of human consciousness that distinguishes us from most other species. The system therefore, provides the notion of a local time clock for every distinct entity in the system and supports each entity operating on its own clock. The system maintains the notion of universal virtual time (UVT) and execution barrier moves forward in time, rectifying data exchanges and interactions between entities in the system, providing the state of every entity in real time. The hardware world is quite familiar with speculative execution and this functionality now exists in the application layer as well.

The problems are unlikely to be solved without improvement in the underlying development environment and infrastructure. An application development and integration framework called CONDOR [1], address the challenges. CONDOR’s biologically inspired characteristics have the ability to:

- Create complex, realistic, and scalable networks of component inter-relationships
- Distribute autonomous controls and monitors
- Implement complex webs of cause and effect
- Dynamically alter the execution structure
- Adapt and evolve the system

In addition, CONDOR addresses the challenges by fusing:

- Advanced systems theory and practice
- Advanced software development
- Low-latency, high throughput, reliable, and robust computer communications
- Sophisticated software integration, interoperability, and synchronization

Common approaches to complex system infrastructure, such as systems based on Microsoft’s .NET framework [5], process-based programming (e.g., systems utilizing threads, semaphores, and locks) [6][7][8], object request brokers [9][10][11], ERP infrastructure [12][13][14], and cluttered web-based technologies, [16] fail in one or more of the problem areas listed above. The tremendous number of constructs causes significant setbacks with most application development and integration methods. CONDOR (with a macro-based sub-language) easily represents and constructs these complex system capabilities.
2 Architecture Description

CONDOR is an object-oriented, event-based, high performance execution system. CONDOR [1] provides high speed communications, which is central to its framework [2], and utilizes numerous messaging fabrics for inter-processor communication: shared memory, wireless, fiber optic, ATM, TCP, IP, and multicast (implemented in a variety of media).

Figure 1. CONDOR Architecture Layering

The Communications Services provide a variety of mechanisms linking clients to intelligent application services, or the hosting processors. Communications Services API (internal and external) is standardized for simplified integration [1]. An abstraction is then supported for unicast and multicast, and permits various implementations, and thus protocols, to work simultaneously. State-Saving Framework and State-Saving Services support reliability, synchronization, fault-tolerance, and implementation of persistence services. The Core Programming Services provides Standard Template Library (STL) programming API, and utilizes the state-saving and persistence features. Event Management Services (EMS) provide high-performance data structures to develop exceedingly complex, reliable, interactive intelligent applications more rapidly. Synchronization Management Services, closely coupled with EMS, control synchronization and timing, which are essential for real-time intelligent applications interfacing with hardware [2]. Standard Application and Integration API (SAIA) expedites development of complex, robust intelligent applications by code generation macros and APIs. SAIA synchronizes components as the overall system executes, and scales to simultaneously execute large numbers of components (an interactive synchronization mechanism hides programming complexity). Distributed Object Management Services and Data Translation Services provide location transparency and a powerful, yet easy-to-use, distributed object computing framework (e.g., the complexity in using different inter-processor communications). The Metadata Infrastructure (MDI) provides:

• A syntactic notation for relational associations
• Logical constraints, consequences, and role-based behaviors entailed by the relationships
• A system of primitive relational abstractions
• Method to change abstractions into conceptual hierarchies
• Means to compose and structure relationships into frameworks and architectures

Collectively, the extensions embodied in CONDOR signify precision in the art. All services adhere to the External Integration Framework, and easily integrate intelligent applications and WAN (Figure 2) [1].

Figure 2. CONDOR System Layout

3 Intelligent System Development and Integration

The overhead costs of information flow between software and hardware continue to increase. Systems integrators attempt to reduce the costs by repeating the information flow, hopeful to attain greater accuracy and to meet the timeframe of need. The attempts do not solve challenges indicative of increased complex systems, and have not been successful due to lack of the following:

• Hardware compatibility
• Communication protocols
• Understanding transfer information
• Well defined system interrelationships
• Mapping for adequate neutral-format standards
• Mutual acceptance of the communication purpose
• Operational meaning in terminology
• Advancement in the core development environment and infrastructure

CONDOR utilizes CGs [1] to integrate hardware and software systems. Visually, a CG mimics knowledge representation in common diagrams for discussions (using whiteboards, slides, or even table napkins). The diagrams, or drawings, are often text snippets (typically enclosed in squares or ovals) and lines
(such as labels) connecting one snippet to another (e.g. A Cat sits on a mat). Experts often use visual aids to quickly communicate complex details during brainstorming sessions (see Figure 3).

Relationship nodes provide critical semantic structure to system descriptions. They frequently represent modifiers, qualifiers, and constraints. CG Relationships have the simplest rules - one Concept node must be connected by an incoming arc to a Relationship; and one Concept node must be connected by an outgoing arc. Relationships act as modifiers (adjectives and adverbs). Concept nodes may be connected to Relationship nodes or Actor nodes.

A Concept node may have any number of incoming arcs or outgoing arcs, and represent a variety of system features. Concept nodes represent a variety of system features. Concepts may be components (or objects), or they may represent actions (or verbs). Concepts may be components (or objects), or they may represent actions (or verbs). Direct connections between Concept nodes is not permitted.

In CGs, text snippets (in a square) are called Concepts. The line connections are enhanced with ovals, called Relationships, containing additional text. Hence, representation of semantic relations, between various concepts, occurs in a manner consistent with common “brainstorming pictures”. Actors (diamond shaped symbols) provide a method to encapsulate interfaces to hardware or software components, and indicate data or signal transforming activity is occurring. Relationships act as modifiers (adjectives and adverbs). Structurally, a CG provides the following advantages for representing and integrating complex systems:

- Inherently hierarchical permitting operation at increased aggregated levels
- Can decompose components to appropriate levels of detail, to meet requirements
- Ability to conceptualize the entire system, or one specific concept
- Capable of capturing any aspect of the system

Using CG’s, CONDOR simplifies hardware and software component integration and concisely represents the entire system control logic. Actor nodes provide the critical ability to encapsulate hardware or software components. Actor nodes enable system integration. Actor nodes can have any number of incoming arcs from other nodes. However, they have only one outgoing arc. The outgoing arc may be connected to another Actor node, or a Concept node.

Relationship nodes provide critical semantic structure to system descriptions. They frequently represent modifiers, qualifiers, and constraints. CG Relationships have the simplest rules - one Concept node must be connected by an incoming arc to a Relationship; and one Concept node must be connected by an outgoing arc. Relationships act as modifiers (adjectives and adverbs). Concept nodes may be connected to Relationship nodes or Actor nodes.

A Concept node may have any number of incoming arcs or outgoing arcs, and represent a variety of system features. Concept nodes represent a variety of system features. Concepts may be components (or objects), or they may represent actions (or verbs). Concepts may be components (or objects), or they may represent actions (or verbs). Direct connections between Concept nodes is not permitted.

4 Conceptual Graphs and the Knowledge Execution Engine

One of the purposes of the Knowledge Execution Engine (KEE) involves controlling the execution of a collection of hardware and software components, as a cohesive and robust system. A valid CG for system integration must contain at least one input Concept node and one output Concept node. Concepts and relationships alone enable a wide range of expression through use of CGs. The basic rules governing CG formation are shown in Figure 4.

In CGs, text snippets (in a square) are called Concepts. The line connections are enhanced with ovals, called Relationships, containing additional text. Hence, representation of semantic relations, between various concepts, occurs in a manner consistent with common “brainstorming pictures”. Actors (diamond shaped symbols) provide a method to encapsulate interfaces to hardware or software components, and indicate data or signal transforming activity is occurring. Relationships act as modifiers (adjectives and adverbs). Structurally, a CG provides the following advantages for representing and integrating complex systems:

- Inherently hierarchical permitting operation at increased aggregated levels
- Can decompose components to appropriate levels of detail, to meet requirements
- Ability to conceptualize the entire system, or one specific concept
- Capable of capturing any aspect of the system

Using CG’s, CONDOR simplifies hardware and software component integration and concisely represents the entire system control logic. Actor nodes provide the critical ability...
• Collection of user-defined concept nodes
• Collection of user-defined relationship nodes
• Collection of user-defined actor nodes
• Unique ID numbering scheme to identify every node, regardless of type
• Description of the connected nodes, and route of the connection
• List of references to input concept nodes (those with no incoming arcs); a valid CG must contain at least one input concept node
• List of references to the output concept nodes (those with no outgoing arcs); a valid CG must contain at least one output concept node
• Data structure that records the truth value of each node (those skilled in the art recognize this may be included in the node data structure)

Data structure that records the truth value of each node

To complete the KEE summary, the CG Execution Cycle (EC) operation must be described. Recall that each node has a truth value – a binary variable. When the system is initialized, the truth value for each node is set to false. Each node can be queried for current truth value. Truth values can be set to true, or reset to false.

Figure 5. Sample CG Implementing a Simple Formula

The KEE reads standard CGIF files and parses it. The KEE includes the CG parser, as it accomplishes much more than standard CG parsers. The KEE’s CG parser must establish all connections between nodes (specified in the CG). The type information and unique ID tag permit the correct execution order of components. The following sequence describes the typical use of the KEE:

1. The system is started
2. Each component in the system is initialized
3. Synchronization relationships are established
4. Inputs are read and loaded into system components
5. The CG associated with the system is initialized and parsed
6. Each system component is associated with a CG element using the unique ID tag (concept, relationship, actor)
7. Each component is registered in the correct collection mechanism for each type, using the unique ID tag
8. The system begins operating, with various evolution methods. For example:
9. A user inputs information, which changes the state of the system (either event-based, process-based, or simple update loop)
10. Regular update cycles occur for various system aspects
11. The system operation mechanism executes, and an
12. Execution Cycle operation for the CG is activated

13. Each time a system operation mechanism is activated, step 4 is repeated

To complete the KEE summary, the CG Execution Cycle (EC) operation must be described. Recall that each node has a truth value – a binary variable. When the system is initialized, the truth value for each node is set to false. Each node can be queried for current truth value. Truth values can be set to true, or reset to false.

Figure 6. Sample Conceptual Graph Showing a Path (Outlined in Red) Contained Within the Graph

When a node is executed, the appropriate software or hardware function is performed. Upon completion of the node’s software or hardware function, the truth value is queried. The truth value is manipulated during the execution of the software or hardware function. If the node is true, then an activation signal is appropriately sent to each node according to the current node’s outgoing arcs.

The first time through the CG execution, an EC begins with execution of the input nodes. Input nodes typically read data values, or access databases, before execution is completed. Minimally, if a single input node executes and is true, the nodes connected by CG arcs are activated. A case in which zero input nodes execute as true, the EC is complete. The system control mechanism is notified that the CG EC is complete. This is the simplest case, but the general concept is true in the opposite case.

A CG can be viewed and analyzed in terms of graph theory, utilizing the rules of CG formation. The constraints on CG formation lead to directed graphs that allow the optimization of computation. The execution of the graph is extremely efficient and straightforward. For customization purposes, several different techniques exist – each maintains efficiency.

The exact circumstances of system integration efforts may vary from one job to the next. Hence, different execution optimization methods may be employed; and each method is
within scope of the KEE. The subsequent description of the KEE’s preferred embodiments provides several optimization possibilities.

### 4.1 First Order Predicate Calculus

The method in which the truth value is used to control graph execution allows the KEE to implement first order predicate calculus logic rules to control the system (a critical KEE point), as illustrated in Figure 7. The KEE unites numerous disciplines in realization, and derives value from the union. The KEE provides a knowledge-based representation of the system to be integrated. Furthermore, the entire toolkit of advanced system theory is employed in a trouble-free, straightforward manner, which contrasts significantly from current complicated methods. Consequently, the KEE enables useful extensions of techniques, common to expert systems. Standard systems integration theory and techniques are incorporated in the design and control system.

![Figure 7. Use of Truth Values to Control CG Execution](image)

The KEE unites numerous disciplines in realization, and derives value from the union. The KEE provides a knowledge-based representation of the system to be integrated. Furthermore, the entire toolkit of advanced system theory is employed in a trouble-free, straightforward manner, which contrasts significantly from current complicated methods. Consequently, the KEE enables useful extensions of techniques, common to expert systems. Standard systems integration theory and techniques are incorporated in the design and control system.

### 4.2 Adaptive and Dynamic Operation

Adaptive systems theory is an emerging science, central to automated, or “smart”, systems. However, a major barrier, or problem, is associated with automated devices and systems. Current systems are insufficient in providing complex and robust control mechanisms. The KEE provides a solution to the barrier in advancement and proliferation of adaptive, automated systems and it can dynamically change the structure system control logic and which components (including new ones) are utilized. As such, KEE facilitates the implementation of adaptive systems. The benefits obtained by using KEE are extended in several dimensions (for example, the ability to interactively and automatically control the system). System performance and flexibility is achieved by utilizing several CGs at once to integrate and control a particularly complex system. The ability to dynamically change the system control logic is central to the KEE’s enhanced flexibility. Typically, a system upgrade in which the control mechanism is altered requires the system to be restarted regularly (the equivalent of a “reboot” to PC users). The KEE eliminates the need to restart. KEE has the capability to:

- Execute multiple CGs
- Dynamically load a new CG simultaneous to system execution
- Pause a CG execution
- Re-start a CG execution once it has been paused
- Stop a CG execution with no re-start

Enabling a powerful set of tools to manage the complexity of system integration solution analysis, design, and implementation will have a dramatic impact on the industries where system integration is a key function.

### 5 Implementation Example: Long-Range Surveillance (Reconnaissance)

The strategies used by soldiers in reconnaissance units have been a favored asset in Eastern Europe, particularly in Estonia [16]. However, army commanders have a growing aversion to risk and an increased preference to use technology such as satellites and drones for reconnaissance rather than insert small teams of soldiers. Long-range surveillance companies in the army have been comprised of 15 six-man teams each led by a staff sergeant. The teams specialize in navigating forward positions to monitor enemy movement and gather intelligence for commanders.

In this modern age of warfare, by utilizing and integrating several systems it becomes possible to create a System of Systems (SoS) to aid and improve Situational Awareness (SA). Warfighters have new SA tools at their disposal ranging from communications to armed, or unarmed, unmanned ground systems (UGV) to unmanned air vehicles (UAV) with sensors proving real-time SA to each warfighter and unit leader. Utilizing highly maneuverable robots designed to support military forces in infantry, intelligence, reconnaissance and special operations missions improves the chance of success in intelligence gathering. Some of these ground robots assist a warfighter where both the unmanned system and the warfighter utilize the intel from the UAV sensor suite.

Consider the idea of using a wide variety of sensors that are feeding information to a redundant collection of artificial intelligence (AI) computing platforms hosted on UGVs. These UGVs will receive sensor feeds from a variety of platforms both airborne sensors carried by UAS
6 Summary

An approach has been presented for intelligent application development and systems integration within the CONDOR architecture. The approach provides a low-cost, straightforward method for developing easy to use, yet complex consumer devices and systems. CONDOR employs advanced system representation frameworks, flexible communications structures, intelligent application services (e.g. real-time control), complex synchronization relationships and constraints, data time tagging, checkpoint restart, dynamic system control, and persistence. CONDOR also provides intelligent application developers and system integrators with a parallel and distributed computing capability that supports a wide variety of hardware hosting and integration.

7 References


