

# A System Architect's View of C2IT®

John M. Hammer, Ph.D.  
Chief Technology Officer  
Applied Systems Intelligence, Inc.  
11660 Alpharetta Highway, Suite 720  
Roswell, Georgia 30076  
March 12, 2003

## Overview

ASI's Command and Control Information Technology (**C2IT**®) is a commercial-off-the-shelf (COTS) government/military version of its **PreAct**® software toolkit. C2IT is used to military applications that solve difficult problems involving uncertainty, time pressure, many alternatives, complex interactions, information overload, expensive or grave failure consequences, and/or capable adversaries. C2IT provides decision aiding with the following functions:

- assess and analyze data from multiple sources,
- plan short and long term activities by selecting from alternatives and resolving conflicts,
- execute activities by issuing commands to its environment,
- interpret observable user actions in terms of higher level user intentions,
- configure the user interface to display the most relevant information and controls, and
- assess the consequences of a user error.

This document was written to describe C2IT characteristics:

- properties of its solutions,
- typical applications,
- appropriate problems for it,
- its architecture expressed in design patterns,
- its engineering process,
- its similarities to and differences from other technologies,
- its supported platforms and languages, and
- the variety of engagements for C2IT technology transition.

The expected audience for this document consists of architects and managers who understand the characteristics of their problem spaces and who are looking for a dramatic improvement to solving complex problems. Architects may in particular benefit from its design pattern description. Sections are independent and may be skipped.

## Properties of C2IT Applications

C2IT applications differ from traditional applications in a number of ways: competence, user orientation, adaptability, communication efficiency, architecture, and system integration.

### *Competence and Autonomy*

C2IT solutions are competent to perform complex tasks on behalf of the user. C2IT executes symbolic representations of human expertise and is capable of emulating their decision-making

---

® PreAct and C2IT are registered trademarks of Applied Systems Intelligence, Inc.

processes. Some C2IT applications are so competent that they operate systems autonomously with minimal user supervision. Since the user interacts less with each system, the user can supervise more systems.

### ***User-Centered Assured Performance***

C2IT solutions assure user performance in application domains where failure has high or grave consequences. Its assessment function helps the user understand the situation. Its planning function helps the user generate good plans. It can execute parts of plans on behalf of the user to reduce the user workload. It can interpret the user's intentions to configure displays and direct assessment and planning. It can help the user avoid the consequences of serious errors.

### ***Adaptation to Situations and User***

C2IT solutions adapt to changing situations far better than traditional software approaches. C2IT frequently assesses the current situation and either confirms that the current plan is still valid or creates a new plan. C2IT applications, although competent, follow the user's lead by interpreting the user's intent.

### ***Development Productivity Substantially Better than Traditional Software***

Our experience is that C2IT solutions can be developed much more quickly than a traditional software approach in which all behavior is implemented in code. At the same time, C2IT applications are more adaptive and more competent than traditional, all code software. For very large system of systems development, we believe that a traditional software approach, which seems well-understood and low risk, is actually high risk because it cannot scale up conceptually.

### ***Efficient Communication***

C2IT solutions communicate efficiently with other intelligent agents, including the human operator. C2IT communicates using symbolic representation of human activities, which can convey far more information with fewer symbols. As an example, C2IT has been used to control an unmanned vehicle using only 800 bytes/second using no data compression.

### ***New Assess-Plan-Execute Architectures***

Many new architectures have major functions for assessment, planning, execution, and intelligent user interface. C2IT is a technologically mature approach to these projects.

### ***Consistent Integration of Numerous Subsystems***

C2IT integrates subsystems so that they always work towards the same objective, not at cross-purposes. C2IT explicitly represents the purposes and high level activities of subsystems and uses these representations to achieve consistency. We call this *semantic integration*. Consistent purpose integration is useful when the user must manage a large number of subsystems.

Traditional software techniques can integrate subsystems so that they can communicate with each other. We call this *glue integration*. The subsystems are joined together, but the integration is less coordinated than with semantic integration.

### **Example C2IT Applications**

C2IT can be applied to a variety of problem domains. The C2IT interpreters remain unchanged, while the knowledge bases are customized to the domain. This is analogous to a database, where

the database engine is unchanged but the schema and queries are specialized to a particular domain.

***Mission Computing for Autonomous Unmanned Vehicles***

The current generation of unmanned air vehicles (e.g., Predator, Global Hawk) can best be described as remotely controlled. The next generation of vehicles (e.g., DARPA/USAF X-45 Unmanned Combat Air Vehicle) is more accurately described as autonomous. Differences are shown in Table 1.

<b>Problem</b>	<b>Current Generation</b>	<b>Future Generation</b>
Threat reaction	Remotely controlled by human operator	Autonomously assessed and reaction planned by onboard intelligent agent technology. Operator may approve agent-generated reaction or input his/her own reaction.
Operators per vehicle	3 or 4	1 operator can control 4 to 10 vehicles
In flight replanning	Impossible	Happens autonomously whenever needed
Cooperation between vehicles	Impossible	Vehicles can negotiate tasks autonomously and present recommendation to operator

**Table 1. Autonomous control far more capable than remote control.**

***Interpretation of Uncertain Data***

C2IT has been used to assess and analyze medical data to make complex diagnoses. The C2IT application had competence that was surprising to the physicians who reviewed its performance. C2IT has also been used to analyze the actions of potential terrorists to identify means, motive, and opportunity to conduct a specific act of terrorism.

***Supply Chain Management***

C2IT is used in eValue, a supply chain management application. It uses customer, manufacturing, and purchase orders across the supply chain to detect production and delivery problems. eValue is a distributed, collaborative, peer-to-peer application that is currently in beta test.

***Potential Applications***

C2IT can be used for a variety of applications such as decision aiding for general aviation, fraud detection, and computer network and computer security operations management.

**Feasible Problems for C2IT**

Every technology has a particular set of problems to which it is well suited. This section describes the types of problems to which C2IT is applicable.

***C2IT Solves Difficult Problems***

C2IT was intended to solve those difficult problems that humans can solve only after they have had 1 to 10 years of education and/or training. These problems typically involve conditions such as uncertainty, time pressure, expensive or grave failure consequences, complex interactions, numerous alternatives, information overload, capable adversaries, and/or Murphy’s Law.

Humans typically manage systems that deal with these problems. Decision aiding for these human users is usually seen as beneficial.

Another indication that a problem is well suited for C2IT is that there is a considerable performance difference between the expert user and the average user. These differences can be explained in terms of the expert's superior knowledge and problem solving skills. A C2IT application can capture the expert user's knowledge and skills to make them available to all users as a decision aid.

C2IT can also be used to solve problems that a human could solve except for two issues:

- labor costs would be too high to solve every instance of the problem, and
- conventional software approaches are inadequate.

### ***C2IT Preserves and Operationalizes Expertise***

C2IT can preserve and operationalize the expertise that a company may lose due to retirement or would prefer to make more widely available.

### ***C2IT Performs Cognitive Reasoning but not Perception or Motor Control***

C2IT is highly suited for emulating the reasoning that experts can verbalize and explain. C2IT itself has no capabilities for perceptual tasks such as finding objects in visual scenes or controlling a robotic arm. However, C2IT could be integrated with technology for perception and/or robotic control. For example, C2IT could suggest possible objects of interest to a visual system. C2IT could assess and react to (plan about) an object detected by a visual system. C2IT could also provide a robotic arm with desired end points or maximum velocities. C2IT uses symbolic representations of the world, not waveforms or vast arrays of sensor data.

### ***C2IT Can Be Modified to Learn***

Changes to C2IT knowledge bases are made by knowledge engineers relatively quickly, compared to software changes. The C2IT architecture could support a learning mechanism; its internal representations were explicitly chosen to be general enough to support the addition of a learning mechanism.

### ***C2IT does High Level Natural Language Processing***

C2IT has been used to interpret a structured sentence input into higher level plans and goals. C2IT has not been used to parse natural language or assign meaning to words.

### ***C2IT does Symbolic Computation***

C2IT performs symbolic computation to assess, plan, execute, interpret, and configure. These computations involve searching for solutions, weighing evidence, and selecting from alternatives. Numerical calculations, such as optimization, probability calculations, or route planning, are integrated into C2IT through library calls. C2IT can provide parameters and goals for these library calls and use the outputs returned from these calculations. Applied Systems Intelligence frequently uses these mechanisms in our application engineering.

### ***C2IT Interacts with GUIs***

Even though C2IT can control and format displays, it is not a GUI framework. C2IT can interact with existing GUIs, or new GUIs can be created explicitly for C2IT inputs and outputs.

### ***Scalability and Performance***

C2IT was designed to work in real-time environments with response times in the .1 second to 1 second time frame. Several related C2IT applications, each working on a separate part of the problem, may be distributed to cooperatively solve problems.

### ***Partial Use of C2IT***

C2IT can be used in part or in whole. For example, A C2IT application could perform only data interpretation (assessment) without doing planning, intent interpretation, or any other C2IT function. C2IT functions such as assessment can be replaced with an alternative assessor. The C2IT architecture is modular to support this interchangeability.

### **Design Pattern Architecture of C2IT**

Activities are the fundamental unit of C2IT computation. All C2IT functions – assessment, planning, execution, interpretation, and configuration – are driven by models of activities. All other computations are driven by the needs of these functions with respect to activity models. Consequently, programming with C2IT is essentially programming with activities.

This section describes C2IT in terms of design patterns [Gamma, Helm, Johnson, & Vlissides 1995]. It will be of interest primarily to system architects and is intended to give a high level overview only.

### ***C2IT is a Set of Interpreters***

C2IT is a set of interpreters, one for each of the major functions of C2IT. Each interpreter interprets a model, which is a hierarchical graph structure that represents activities or state. Interpretation requires traversal of the graph that is unique to each interpreter. The interpreters are analogous to a database engine, in that neither has any domain knowledge. The database domain model is in the schema and query language. The C2IT interpreters' domain models are the hierarchical graphs.

The interpreters read their models from a file at runtime. Thus, C2IT application behavior can be changed without any re-compilation. Each model has design patterns that are unlike software design patterns. These design patterns are described in the C2IT training classes. C2IT interpreters store models in a manager that is responsible for managing the lifecycle of the model entities. The interpreters are stateless and conform to a service-based architecture. C2IT uses the prototype pattern to describe the models.

### ***C2IT is Modular and Loosely Coupled***

Each C2IT interpreter has an abstract interface that allows it to be replaced by an equivalent object. This would allow, for example, the assessment interpreter to be replaced by an equivalent assessor that is not part of C2IT.

C2IT interpreters read from input queues and write changes to output queues. This architecture allows the interpreters to be distributed to different processors. Interpreter inputs are either subsystem state changes or user commands to subsystems in the environment. Interpreter outputs are either model changes or commands to subsystems in the environment. The managers implement the model changes by processing change requests taken from the output queues.

### ***C2IT Integrates with External Software***

C2IT can invoke external algorithms (strategy design pattern) that perform calculations. These calculations are written in C++ and must conform to a C2IT specified interface, or, more typically, use an adapter that conforms to this interface. ASI engineers frequently use this strategy mechanism to integrate calculations that are better done in a traditional programming language than by an interpreter.

### ***C2IT is a Dynamic Object Model***

C2IT is a *dynamic object model* [Riehle, Tilman, & Johnson 2000]. This architectural pattern describes a family of object-oriented architectures that provide runtime object flexibility beyond that provided in the implementation language (e.g., C++). For example, an object might change its type, methods, or attributes at runtime. Or, an object might change how virtual method calls work (e.g., allow parent classes to provide *before* and *after* methods that run before and after sub-class method execution). The advantages of dynamic object model are its flexibility and compactness. Source code reductions on the order of 100X have been reported, which is consistent with our experience.

### ***Conclusion***

C2IT is an interpretive, dynamic object model for activity assessment, planning, execution, interpretation, and configuration.

### **Engineering with C2IT**

Engineering with C2IT involves two major activities: adapters and knowledge engineering. Adapters are required to connect C2IT input/output to the external environment. A very significant part of this engineering is understanding the semantics of the external environment, which is essential to knowledge engineering.

### ***Engineering Personnel***

The personnel who build C2IT applications must be willing to learn a new way to do computation. From 3 to 9 months of experience are required to achieve competence. This transition is roughly the same as the following:

- from C programmer to competent object-oriented programmer
- from object-oriented programmer to competent database analyst
- from database analyst to competent Java Swing programmer

Key attributes of success are motivation to learn a new way of development, abstract reasoning, and willingness to understand a domain (e.g., aviation, medicine, transportation).

### ***C2IT Model Engineering***

This engineering process will be compared and contrasted with hierarchical use case modeling of business processes [Jacobson, Ericsson, & Jacobson 1995]. Jacobson et al. in a second book on use cases described how a hierarchy of use cases could model business processes. Although use case models are traditionally used only for analysis purposes, they are widely understood and thus will serve our expository purpose. That purpose is to differentiate traditional software development from C2IT development.

A hierarchical use case model begins life in much the same way as C2IT models, but C2IT models require much more detail because they are interpreted at runtime. C2IT models are one

form of activity models; other examples of activity models include use case models, business process models, UML activity diagrams, Petri nets, and workflow models.

### *Retention of Design Intent*

In traditional software engineering, use cases are primarily an analysis artifact. Use cases are an activity model that is created to identify classes, their methods, and the flow of control among methods on classes (which is often represented in sequence diagrams for each use case). Use cases themselves are not directly represented in the software. The use case is in effect the flow of control among methods. Typically, the flow of control has no access to the name of the use case, no access to the previous method or the next method, no access to error cases, etc. The use case, which is roughly like a class, has no corresponding object at runtime.

In C2IT, the designer's intentions are retained as hierarchical activity models for the C2IT interpreters. These models are the primary focus of C2IT execution. All C2IT computational needs are derived from these models and the interpreter functions.

As a consequence of this fundamental change in the executable concept, C2IT excels at several functions that are difficult for traditional software architectures. First, C2IT excels at modeling the user's current activity and is easily able to track changes in those activities. The model of user's activities is useful for a wide variety of practical purposes: selecting information to display and implicit communication about the user's intentions. Second, C2IT excels at selecting appropriate activity from a set of alternatives (also known as planning). Activity tracking and activity selection also complement each other, in that tracking and selection are *mixed initiative*. This term simply means that either the user or C2IT may take the initiative in responding to a situation. C2IT's approach to mixed initiative behavior is elegant and benefits from over a decade of our experience with it.

One beneficial consequence of the retention of design intent is that its model is useful for indexing all kinds of additional behavior. In other words, functionality that would be appropriate in the context of a use case can be attached or activated by the C2IT models. For example, suppose that error handling, displays, workflow, and user response time are all important in an application. The C2IT models are a good place to attach such information. Of course, it would be possible to embed each of these aspects in the control flow of a traditional application. However, in a large application, the correctness of each of these aspects – spread throughout a lot of source code – is likely to be difficult to establish.

### *Details about C2IT Activity Models*

The C2IT model design uses two concepts – plans and goals – to model activity. C2IT plans are similar to use cases and represent operations that are intended to change the state of the environment. C2IT goals represent states that the user is trying to achieve. Goals have no corresponding concept in use case analysis; they are often left implicit.

Goals and plans have a hierarchical, structured relationship. A goal is decomposed into sub-plans, each of which may be sufficient to accomplish the goal. Sub-plans are in an OR relationship to each other. Plans may be decomposed into sub-goals, all of which must be achieved for the plan to reach its desired state. Sub-goals are in an AND relationship to each other.

### *Alternative and Sequential Relationships*

C2IT represents both alternative and sequential relationships. Alternative plans are different means of achieving the same goal. C2IT represents not only the alternatives but also the logic

for selecting the best from among the alternatives. In addition, this logic must be based on available environmental inputs. C2IT consequently requires access to a complete state vector. Use case models do not contain selection logic.

C2IT models and use case models both represent sequential relationships. C2IT design patterns avoid sequential relationships unless a chronological relationship is the best model. Our experience has been that sequential relationships are often used inappropriately, and that the resulting systems often become trapped in a particular state.

#### *Instances, Instance Management, and Multiple Actors*

C2IT models must handle multiple instances of the same activity being executed simultaneously. Use case models do not address this problem. To deal with this problem, C2IT must be able to distinguish the various instances with different attribute values (akin to database keys). The instance's lifecycles must be managed, which means that additional logic is required for lifecycle transitions. For example, C2IT must be able to determine when activity instance is no longer appropriate (and therefore must be revoked) so that the planner can create new activities.

C2IT also supports multiple instances of different activities being executed simultaneously. The historical roots of this feature stem from aviation in which pilots frequently multi-task. This feature may be used in environments where teams cooperate to solve problems. Activities may be transferred among the team members, and C2IT can track such changes. This feature can be extremely valuable when needed, and yet can be safely and easily ignored when not needed. Another benefit of multiple executing activities is that the activity model may be finer grained than a corresponding use case model. Multiple activities can be combined to represent what would be a single use case. This gives the planner the flexibility to select the best combination of activities to match a particular problem.

#### *C2IT is an Actor*

Another dimension of cooperation is that C2IT is an actor among other things. This means that C2IT and the user can share activities or transfer control of activities back and forth. In receiving control, C2IT assumes agent-like behavior.

#### *Layered Architecture of a C2IT Model*

A C2IT activity model has three levels. The top level represents the perennial concerns of users. These concerns are often generic: preserve resources, detect problems, handle problems, be safe, and complete the assignment. The next layer down in the model decomposes the perennial concerns into specific goals and alternative plans for meeting those goals. This decomposition usually takes two or three levels of goals and plans and is highly domain specific. C2IT-specific design patterns are used in this level. The lowest level activities represent the operations that are directly executable in the environment. The lowest level is also highly domain specific but is usually concretely described in an interface document. A C2IT interpreter (through an output queue) may request the execution of these operations. Alternatively, these operations are observable by C2IT when executed by a user.

#### *Access to Subject Matter Experts*

Access to subject matter experts is required to build C2IT models. As a rule of thumb, one hour of the expert's time is required for every ten to one hundred hours of knowledge engineers time. ASI has on staff domain experts for some application areas. It is considered a best practice to

use the best expert as a source for knowledge engineers. Access to experts is required throughout the engineering lifecycle.

A subject matter expert must be able to explain such topics as:

- how inputs are interpreted,
- how choices are made,
- what the alternatives are, and
- what can invalidate a currently executing plan.

## **C2IT Comparison to Other Technologies**

C2IT's competing technologies are traditional software development, rule-based expert systems, neural networks, and workflow engines.

### ***Traditional Software Development***

Traditional software approaches can be used to solve the same problems that C2IT is intended to solve. Traditional solutions are likely to be very large applications. A framework would be a more systematic approach to difficult problems, as it would offer higher level programming abstractions that are key to improved development efficiency. However, the conventional wisdom is that frameworks are quite difficult to create and evolve. Very few developers have experience creating a framework, although most have used them. C2IT is an activity model framework that is based on over ten years of experience.

### ***Rule-Based Systems***

Rule-based systems are a programming methodology that is suitable for small scale problems. It is easy to get initial results quickly, but as the problem space become larger, progress slows dramatically. Large sets of rules are difficult to modify and extend, and the problems C2IT is intended to solve are too large to be effectively solved using rule bases alone. Rule-based systems provide programming abstractions that are above traditional programming languages but far below C2IT's activity models.

### ***Neural Networks***

Neural networks are suitable for classification problems that map a vector of data into a classification (e.g., bird, red color, 5 inches tall → cardinal). Neural networks learn how to classify using training data. Neural networks may be suitable for individual computations within C2IT, but have not been used for activity model problems of the size that C2IT can handle. For example, neural networks are not typically used to implement searches.

### ***Workflow***

Workflow models use predictable sequences of activities that involve multiple users and span long time periods (perhaps even days). In contrast, C2IT handles problem with *unpredictable* activity sequences. C2IT dynamically computes what activities are appropriate next (i.e., plans), whereas workflow models use static, fixed future activities. C2IT has been used to model activities of multiple users and can be programmed to generate activity models that span long time periods.

## **Platform Issues**

C2IT is written in C++ and runs on Windows, Linux, Solaris, Irix, and probably anything else with a GNU C++ compiler. It can be integrated with C++ programs, and XML and Java JNI integration are also available.

## **Offerings**

ASI has two offerings for building C2IT applications. First, ASI can build C2IT applications. Second, ASI can teach customers how to use the technology. In the latter, ASI begins classes and mentoring. The best way to mentor is for ASI engineers to work on real problems with customer engineers. Over time, ASI participation is reduced as customer engineers develop expertise with C2IT.

## **References**

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns*. Addison-Wesley: Reading, MA.
- Jacobson, I., Ericsson, M., and Jacobson, A. (1995). *The Object Advantage*. Addison-Wesley: Reading, MA.
- Riehle, D., Tilman, M., and Johnson, R. (2000). Dynamic object model. *Proceedings 7<sup>th</sup> Pattern Languages of Programs Conference*: Monticello, IL.