
AFRL-RI-RS-TR-2022-144

# INTENT-DEFINED ADAPTIVE SOFTWARE (IDAS)

SIERRA NEVADA CORPORATION

*OCTOBER 2022*

FINAL TECHNICAL REPORT

***APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED***

STINFO COPY

**AIR FORCE RESEARCH LABORATORY**
**INFORMATION DIRECTORATE**

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

# NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2022-144 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

**/ S /**
WILLIAM E. MCKEEVER
Work Unit Manager

**/ S /**
GREGORY J. HADYNSKI
Assistant Technical Advisor
Computing & Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

# REPORT DOCUMENTATION PAGE

| 1. REPORT DATE | 2. REPORT TYPE | 3. DATES COVERED | |
|---|---|---|---|
| | | START DATE | END DATE |
| OCTOBER 2022 | FINAL TECHNICAL REPORT | APRIL 2020 | APRIL 2022 |

**4. TITLE AND SUBTITLE**
INTENT-DEFINED ADAPTIVE SOFTWARE (IDAS)

| 5a. CONTRACT NUMBER | 5b. GRANT NUMBER | 5c. PROGRAM ELEMENT NUMBER |
|---|---|---|
| FA8750-20-C-0518 | N/A | 62303E |

| 5d. PROJECT NUMBER | 5e. TASK NUMBER | 5f. WORK UNIT NUMBER |
|---|---|---|
| | | R2ZA |

**6. AUTHOR(S)**
Jeff Smith and Mitch Kokar

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Sierra Nevada Corporation
444 Salomon Circle
Sparks, NV 89434-9651

**8. PERFORMING ORGANIZATION REPORT NUMBER**

| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | | 10. SPONSOR/MONITOR'S ACRONYM(S) | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |
|---|---|---|---|
| Air Force Research Laboratory/RITA<br>525 Brooks Road<br>Rome NY 13441-450 | DARPA<br>675 North Randolph St<br>Arlington, VA 22203-2114 | AFRL/RI | AFRL-RI-RS-TR-2022-144 |

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
Approved for Public Release; Distribution Unlimited. PA#: AFRL-2022-5152
Date Cleared: 25 October 2022

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**
The overall Intent-Defined Adaptive Software (IDAS) goal was to automate code generation derived from software intent, and associated constraints, for rapid adaptation to late changes in requirements and operating environments. IDAS was divided into Automated Software Genera-tion, Problem Set Generation, Integrated Test & Evaluation and Experimental Control and Transition Technical Areas (TAs). During the research and initial prototype Phase 1, Sierra Ne-vada Corporation (SNC) performed on the latter TA, working with problem sets/changes, exer-cises, prototype toolchains and execution guidance from other TAs and providing, 1) abstrac-tion layers and software framework (consisting of dashboard, control, and abstraction APIs along with exemplary use cases) used for evaluation and 2) a Cloud Agility Baseline (CAB), with associated Agile process improvements, to compare prototype workflows and software. This framework was intended to test and evaluate the effectiveness of technologies supporting the continual adaptation of DoD software-enabled systems.

**15. SUBJECT TERMS**
Intent-Defined Adaptive Software (IDAS), DevSecOps; Agile Software Development, SADL (Semantic Application Design Language), AADL (Architecture Analysis and Design Language)

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES |
|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | C. THIS PAGE | | |
| U | U | U | SAR | 20 |

| 19a. NAME OF RESPONSIBLE PERSON | 19b. PHONE NUMBER *(Include area code)* |
|---|---|
| WILLIAM E. MCKEEVER | N/A |



PREVIOUS EDITION IS OBSOLETE.

**STANDARD FORM 298 (REV. 5/2020)**
*Prescribed by ANSI Std. Z39.18*

## TABLE OF CONTENTS

## LIST OF FIGURES

## 1.0 SUMMARY

The overall Intent-Defined Adaptive Software (IDAS) goal was to automate code generation derived from software intent, and associated constraints, for rapid adaptation to late changes in requirements and operating environments. IDAS was divided into four technical areas (TA): Automated Software Generation, Problem Set Generation, Integrated Test & Evaluation and Experimental Control and Transition. Sierra Nevada Corporation (SNC) was a TA 4 performer; working with problem sets/changes, exercises, prototype toolchains and execution guidance from other TAs and providing, 1) abstraction layers and software framework (consisting of dashboard, control, and abstraction APIs along with exemplary use cases) used for evaluation and 2) a Cloud Agility Baseline (CAB), with associated Agile process improvements, to compare prototype workflows and software. This framework was intended to test and evaluate the effectiveness of technologies supporting the continual adaptation of DoD software-enabled systems.

## 2.0 INTRODUCTION

The objective of the Intent-Defined Adaptive Software (IDAS) program is to develop technologies that can capture the intentions of software engineers, to enable rapid code generation to support the continual adaptation of Department of Defense (DoD) software-enabled systems. In practice, changes in requirements and resources are a common occurrence. The program will develop new methods for representing the intent of software and its abstract constraints separately from its concrete instantiation and will leverage automated methods to adjust to a particular instance.

Technologies developed on the IDAS program will enable rapid adaptation of software to changes in requirements and/or operating environments. There are three underlying beliefs of IDAS:

1. Creating separate representations of the problem to be addressed by the software (e.g., the constraints on a viable solution), and the actual solution (e.g., a specific software architecture and program that addresses the problem), is essential for scalability. If the problem constraints include aspects of the intended solution (e.g., opting for the use of a particular algorithm that now implicitly fixes certain data types and concurrency requirements when there are viable alternatives that do not add such constraints), the resulting constraint satisfaction problem is drastically more difficult to solve, because such aspects ramify the dimensions of the problem, creating a more complex search space.

2. Representing the program as a set of higher-level programmer intentions rather than just as specific, concrete source code that addresses a current set of problem constraints is essential for enabling rapid future changes.

3. Application Programming Interfaces (APIs) and pre-defined interfaces are concretizations that hamper software flexibility.

This third point requires some unpacking. Conventional APIs and interfaces hide the underlying implementation of a software module from its users, creating an abstraction boundary that enables the module users and module developers to conduct their development activities independently. For this to work in practice, however, the APIs must not change in ways that invalidate the assumptions of the API users. The specification of an API therefore requires extensive commitments to design choices, in other words, many concretizations and conventions (typically described in human-readable documentation).

The IDAS program will enable adaptation of software to radical changes in requirements or its computational environment with an order-of-magnitude reduction in the effort required. The key idea of IDAS is the separation of problem description (in terms of intentions and constraints) from any particular, concrete instantiation. This intent and constraint model must be semantically accessible to an IDAS toolchain, yet expressive enough to capture the relationships between the problem and the method by which generated software can solve and validate a solution. For IDAS to transition, this capture process should be done to the greatest extent possible within the familiar process of writing software, and impose minimal additional tasks on developers who may not understand formal methods. Through additional automation of specific implementation generation,

software sustainment effort should be drastically reduced, freeing engineers to focus on the design of the software and adding new functionality.

The IDAS program had 4 technical areas (TA). TA1 is Automated Software Generation. The goal of TA1 is to create technologies that enable software engineers to develop and verify adaptive software through a deferred-concretization methodology. The core challenge will be to enable traditional developers to work at a higher level of abstraction than currently possible with minimal additional effort.

TA2 is Problem Set Generation. TA2 will develop sets of requirements and environments that are comparable in complexity to, but lack the security sensitivities of, actual DoD systems. These surrogate problem sets should map to DoD use cases, but will be temporally compressed into an evaluation period to test the ability of TA1 and TA4 performers to keep pace with changing requirements and environments.

TA3 is Integrated Test and Evaluation. TA3 will ensure the proper execution of experimentation by deeply understanding the nuances of the TA4 and each TA1 approach, and adjusting the specifics of timing for the release of changed requirements during a 1-4 month evaluation exercise engagement.

TA4 is Experimental Control and Transition. In order to properly evaluate and measure TA1 performer approaches, the TA4 experimental control and transition team will establish a baseline of performance, against which TA1- developed software and workflows will be compared. The TA4 performer should apply current software engineering best practices to develop software that addresses the same requirements and environmental constraints as TA1 during each evaluation exercise, and will respond to all changes in requirements or computational resources.

The Sierra Nevada Corporation IDAS contract filled the role of a TA 4 performer, to support the Defense Advanced Research Projects Agency (DARPA) in achieving their objectives of reducing the cost of software operations and management caused by the complexity of updating software to meet changes in requirements and computing resources. Their solutions have been deployed into a wide range of environments, at all classification levels to include tactical disconnected resources, private clouds and commercial clouds. Throughout their programs they have implemented processes, tools and technologies to enable a flexible open architecture that can be reused and evolved. This flexible architecture is the core to enabling rapid development and minimizing the impact of changes. All of SNC's software programs are managed using Agile development with Scrum and implement the best of breed Agile development techniques. The SNC Team's industry best practice experience ensures delivery of a quality TA4 solution, on time, and within budget.

## 3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

The IDAS TA4 key challenge is establishing a reliable baseline to measure the effectiveness of the TA1 solutions. To be effective, this baseline must be free of measurements that are not directly applicable to the problem, e.g. learning entirely new skills or incorporating entirely new technologies. The TA4 team must develop the abstraction layers and platforms necessary to perform the exercises provided by TA2, so the baseline measurement reflects only the fundamental difficulty of changing requirements of the exercises. This will enable effective identification of areas that complicate software maintenance.

The SNC Team implements the Agile process across all phases (See Figure 1). SNC is a leading practitioner of the Agile process and has used it for every major project for over a decade. SNC's Agile process mitigates the risk inherent in project planning, ensures good communications among all team members, and pro-vides rapid adaptation to changing circumstances and conditions. Velocity measurement, a key Agile concept, will be directly applicable to creating a reliable IDAS baseline measurement. Velocity will be established for the team when composing the CAB. It will create a baseline for the volume of work that can be competed for a set period of time. This will provide a relative measurement of the im-pact of changing requirements. Additionally, implementation time per story will be captured enabling TA3 to compare the cost of implementation and reaction to change. SNC's Agile process will also help ensure a successful Phase 3 transition by establishing an effective and frequent communication.

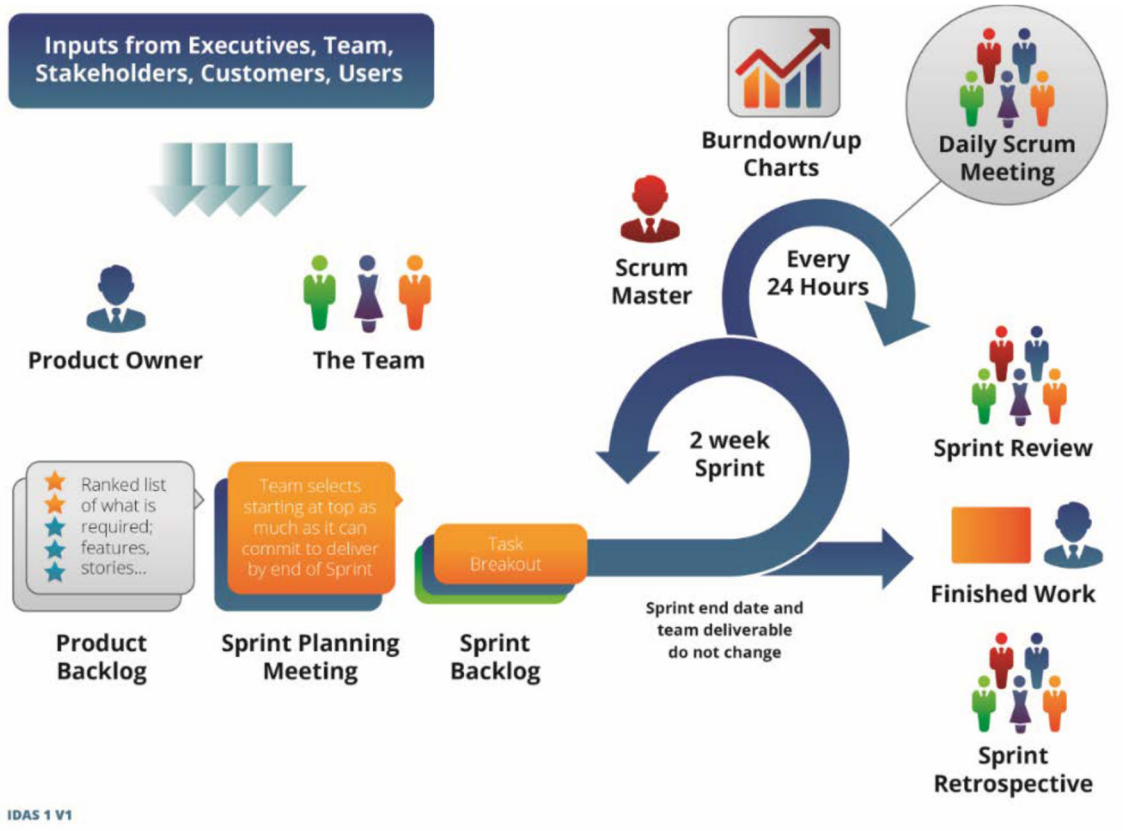


**Figure 1: Agile process management enables responsiveness to change**

During Phase 1 (18 months), the SNC Team will build a CAB, a baseline framework of abstraction layers and microservices to support exercise events. During this phase, they will collaborate with TA3 team(s) to determine the data needed to effectively compare TA1 to TA4. TA2 will develop exercise requirements and parameters and these will be provided to TA1 and TA4 teams by TA3 company. The test exercises during Phase 1 will familiarize all of the teams with the evaluation exercise process and provide an opportunity to make adjustments prior to Phase 2. In Phase 1, SNC will assemble a team that includes software professionals with comprehensive experience in software architecture, operating systems, system security, middleware frameworks, model-driven development, formal methods, distributed/cloud and web-based computing, open systems, DARPA projects, build processes, continuous integration and continuous deployment, Agile development methods, and the entire software development life cycle. The team will comprise engineers from the Enterprise Engine (E2) and Joint Cyber Intelligence Tool Suite (JCITS) Conversion programs and other similar efforts who have broad experience in these areas and who have successfully handled changing requirements.

The SNC Team will build a backlog based on the two exercise areas and any information provided on test exercises. The backlog will be used as a foundation for modeling and definition of abstraction layers. To develop the abstraction layers and framework to provide a reliable baseline for the TA2 exercises, the SNC Team will create a CAB. The CAB will be built to support a Logistics, Cloud and a third, to be defined exercise. The CAB will pro-vide flexibility for the various exercises to promote reuse of microservice components. For the Logistics exercise the micro-services will route supplies to area of responsibility (AOR) via several transport means supporting evolving political and military realities. The core features of the CAB will be to provide an easily adaptable framework which can be molded to meet mission requirements and changes to those requirements with ease.

In order to achieve this adaptability, platform independent microservices will be utilized. All cloud systems have coarse-grained services in categories e.g. storage (S3 or EFS), databases (RDS, Neptune, DynamoDB), migration and transfer (e.g. Snowball and DataSync), network content and delivery (e.g. VPC or CloudFront), security (e.g. Resource Access Manager), and machine learning (e.g. DeepLens, SageMaker or Tensorflow). The problem is the cloud vendors macro-services have vendor-specific Application Programming Interfaces (APIs). By creating an abstraction layer which is service independent which can broker the various macro-services you create flexibility and overall reliability reducing the overall integration friction. This is what has driven the industry to microservice architecture where services are small, independent, and loosely coupled with strong cohesion. For instance, Azure defines a Domain-Driven Design (DDD) that “provides a set of design patterns that you can use to create the domain model”. This approach is more agile with small/independent

microservice/infra-structure development and is resilient and scalable (See Figure 2). The CAB Architecture provides a valid baseline representing the state-of-the-art in software development.

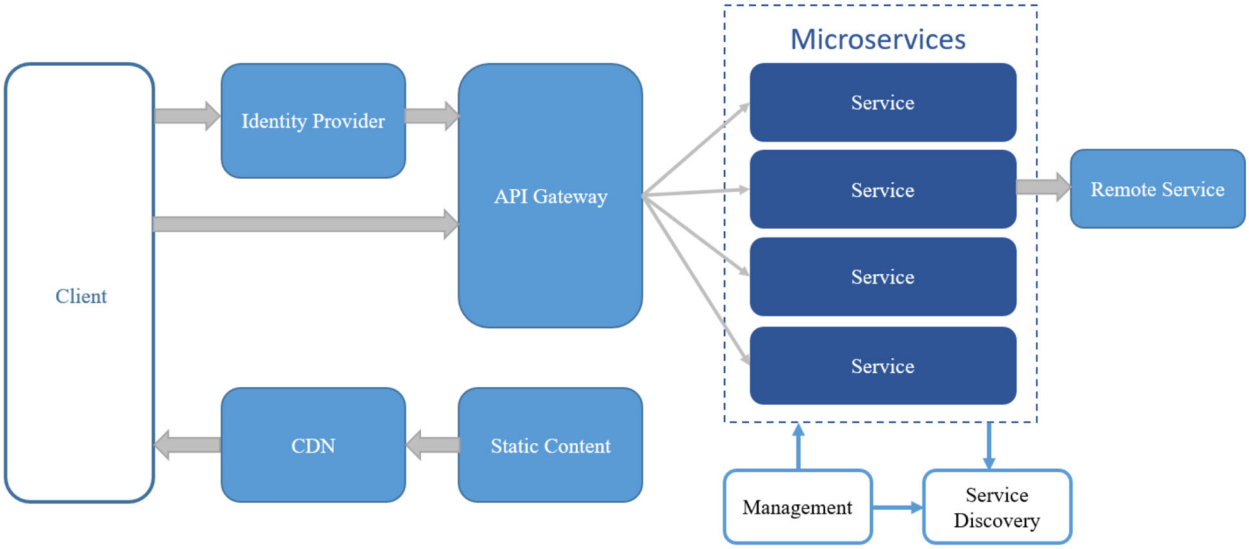


**Figure 2: CAB Architecture**

SNC will extend the concept of this DDD framework to serve as a gateway to all cloud providers, provide required custom microservices and build microservices with enhancements to a standard Model Based Systems Engineering (MBSE) framework as well as subscribe to key standards for new service definitions. For instance, cloud platforms do not provide standard lightweight logging services and they anticipate such services will be needed to support the metrics they want to collect. SNC will develop these microservices, in this case, conforming to the Object Management Group (OMG) Lightweight Logging Specification (LLS).

SNC's MBSE approach provides a formalized application of modeling to support system requirements, design, analysis, verification and validation activities. Capturing requirements as part of a model allows quick reaction to changing requirements. Their implementation is novel in two ways. First, The Enterprise Unified Process [Ambler] shows one approach to the iterative MBSE process. They will practice the Agile form of MBSE by creating extensive models before writing source code for Agile models good enough for each sprint. Second, SNC will use mainstream plugins to the MagicDraw MBSE to support microservice design expressed as Elemental Design Patterns (EDP). EDPs are written in the style of traditional design patterns, complete with applications they are suited for, example implementation, usage consequences and related patterns, but at a more granular level, with solutions to common problems found in existing software systems. In fact, all object-oriented programs have EDPs, but without a strategy and standard, they are an infinite set.

EDPs are granular forms of coding patterns in common usage. A given EDP can be implemented in many ways but still embody the same concept. An isotope of a design pattern defines how external interfaces remain constant despite pattern variances, much like an atom is the same pattern

despite the number of electrons. The idea of separating the implementation from external interfaces is not new to object programming, but isotopes, separating implementation and interface of a concept, is a novel part of EDPs that help insulate against requirements and constraints changes. For example, suppose an *object f* has a method *foo* calling method *bar* of *object b*. They say that *f.foo* relies on *b.bar*. The object, type and method similarities between these two endpoints is well defined and defines the EDP between them. If the implementation is changed by injecting a new method-calling chain, the construct is altered but the relationship between the endpoints remains unchanged. The strategy for achieving their goal is to invest in the front-end activities of the software engineering process life cycle by modeling standard EDPs in SysML, using an EDP composable, graphical abstractions supported by large-scale modeling tools.

EDPs correlate to components that have mechanisms provided at the beginning of a software project (in architecture design and development), rather than putting out fires whenever they occur. They are nestable (EDPs within EDPs), as in Figure 3 below described in Pattern Instance Notation (PIN) and composable (forming larger patterns from micro-versions).

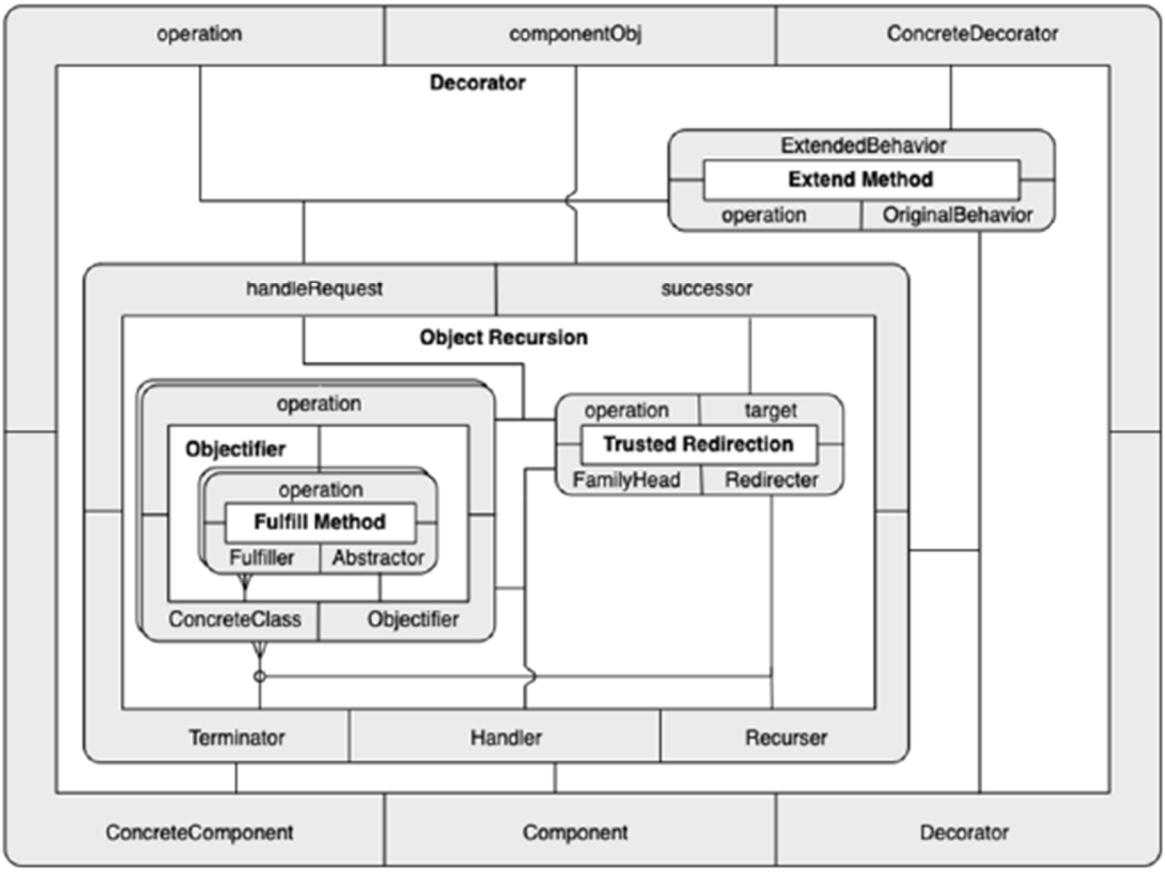


**Figure 3: Nestable EDPs described in PIN**

## 4.0 RESULTS AND DISCUSSION

Between the IDAS kickoff meeting and first cadence meeting, in July 2020, SNC had settled into their agile process, derived candidate hybrid model-based code generation approaches, applied them to their Cloud Agility Baseline (CAB), and assembled the beginnings of their Cloud Native Development Tools to support their initial Continuous Integration and Continuous Deployment (CI/CD) pipeline. SNC had refined their method to decompose complex problems into smaller parts and to make those parts as general as possible to promote reuse and resilience to requirements changes between July 2020 and their second Cadence meeting in September 2020. The Domain-Driven Design method gave developers the understanding they needed to safely change or extend a solution in response to new requirements. The method distilled a complex problem domain into a common modeling language to help keep the evolving design and the model-based portion of SNC's code base synchronized with each other.

Before the first PI Meeting in October 2020, SNC had 1) defined the CAB requirements in SysML, using a IEEE 15288 compliant framework, 2) augmented the Cloud Native Toolbox to help respond to requirement changes (including tools and technologies for databases, middleware, data processing, and user interface development), 3) used this toolbox to deploy a containerized form of the initial CAB to the Amazon Web Services (AWS) cloud, 4) investigated several possible mechanisms for automatically generating code and configuration for parts of the project and used those mechanisms to respond to IDAS requirement changes and 5) cataloged and classified types of requirement changes (with an ontology for modeling requirement changes and implementation guidance when changes occur).

After this first PI Meeting, SNC had 1) refined the CAB to more closely resemble an actual cloud-based logistics system, 2) derived a representative set of CAB requirement changes to measure their velocity and evaluate approach refinements, 3) completed their preliminary classification and ontology of software requirement changes to be able to choose which requirement change processes to use and to measure their effectiveness of responding to changes, 4) automated various the DevOps aspects of the CAB (including continuous integration with unit, integration, and regression tests; reproducible development, test, and continuous deployed environments) and 5) selected the model-to-code mechanisms that they planned augment manually, including the use of fundamental design patterns.

By November 2020, SNC had 1) built a proof-of-concept prototype for how model and code development can cooperate, 2) built baseline approaches solving the traveling salesman problem (prototypical SNC planned Routing Planner DoD-relevant application), 3) evaluated TLA+ for model-checking behavioral properties, and implemented a version of a CAB-related algorithm, to show the interaction between SysML/formal models and code interoperate, 4) built a system test harness to evaluate run-time characteristics of deployed system under test, 5) increased the confidence in their UML/SysML models with mappings to OWL, 6) verified the satisfaction of modeled requirements through OWL/SPARQL and 8) upgraded CI/CD pipeline for delivery in test evaluation format.

For the second IDAS PI meeting in January 2021, SNC had 1) designed and developed a full lifecycle system simulator, 2) presented results of the research of possible targets (identifying criteria for target selection) for new program shift, 3) continued refining the CAB to more closely resemble an actual cloud-based logistics system, 4) analyzed the use of category theory for system, 5) continued investigating mapping of SysML models to OWL and 6) continued developing a method for ensuring requirement satisfaction using SPARQL and reasoning.

Since the January 2021 pivot request, SNC had settled on their first selection of a DoD-relevant application of IDAS technology that could be shared with the other TAs. This was the JADC2/Narwhal Route Planner as it met the candidate deployment criteria as expressed in the table below. They had also built a shareable form of JADC2/Narwhal Route Planner. Towards this end SNC had a) built open version of CUI Route Planner, b) completed Data and Software Release form, and sent code, for the open version of Route Planner with AFRL, c) assembled an unclassified dataset to stimulate the Route Planner and d) stood up a portable Docker image of an NGTS simulator to dynamically stimulate the Route Planner. SNC stood up and demonstrated UxAS Framework on AWS Cloud. They had also researched and stood up UxAS-related AADL and SADL tools/data from "Safe and Secure Systems and Software Symposium", Summer of Innovation (SoI) Group GE/Rockwell to gain improved UxAS understanding and to avoid research duplication. The SNC team wrote a specification of a SysML to OWL extension, to perform deeper OCL constraint checking and implement a requirements change ontology, read XMI and ontology, extract block operations and OCL elements from XMI, generate ontological representations for each operation and OWL axioms for each OCL element, and insert these ontological representation into the ontology.

Consistency checking of UML/OCL models is a challenging issue in software development. The SNC team developed OWL/ontology-based method to detect the inconsistencies in the UML/OCL models as the first step of requirement change management. Specifically, they map the UML/OCL models to OWL, so that the consistency of the corresponding ontology can be checked by OWL reasoners automatically. They propose a set of mapping rules to interpret the components of UML state machine diagrams, along with OCL constraints, to OWL DL.

| Target Property | Narwhal |
|---|---|
| 1.Composable, reusable components for multiple systems (macro-patterns) | JADC2 is large-scale C-C5ISR&T SoS made up of all-domain set of prototypical apps |
| 2. Deployable on "embeddable" part of a system of systems with adaptability focus | Narwhal targeting DWP platforms |
| 3. Includes security-first design yet can be researched in unclassified environment and extended to secure systems | Narwhal has worked with aspects of system in university setting with methods to move in classified setting |
| 4. Utility to DARPA on overlapping program(s) | Additional applicability to SoSITE/Stitches |
| 5. Utility and ability to affect chosen DoD program(s) | Narwhal also targeting DWP deployment in 2021. Stitches also involved with JADC2 |
| 6. Rapid handling of "stressing" cases exemplary of major requirements change – discuss examples | Driven by JADC2 requirements, spread across JADC2 participants, driven/exercise by real scenarios |
| 7. Legacy code incorporation with novel code (inside-out and top-down design/code) | Narwhal has combination of advancing legacy and novel software |
| 8. Operate/test with legacy simulators, with monitoring and trace, HLA/DIS compatible | Uses same mission simulator (NGTS) as JADC2, used many, will work with Mitre to establish UML/simulator configurator |

**Figure 4: Possible targets Properties for Narwhal**

Two months after the January 2021 pivot, the DARPA IDAS Program was cancelled, with the exception of finishing out the summer with universities. Working with Northeastern University, the work consisted of an OWL/ontology-based method to detect the inconsistencies in the UML/OCL models and documenting it in a paper called "An Ontology-based Method on Detection of Inconsistencies in UML/OCL Models" [1].

## 5.0 CONCLUSIONS

The overall IDAS goal was to automate code generation derived from software intent, and associated constraints, for rapid adaptation to late changes in requirements and operating environments. IDAS was divided into Automated Software Generation, Problem Set Generation, Integrated Test & Evaluation and Experimental Control and Transition Technical Areas (TAs). During the research and initial prototype Phase 1, SNC performed on the latter TA, working with problem sets/changes, exercises, prototype toolchains and execution guidance from other TAs and providing, 1) abstraction layers and software framework (consisting of dashboard, control, and abstraction APIs along with exemplary use cases) used for evaluation and 2) a Cloud Agility Baseline (CAB), with associated Agile process improvements, to compare prototype workflows and software. This framework was intended to test and evaluate the effectiveness of technologies supporting the continual adaptation of DoD software-enabled systems.

## 6.0 REFERENCES

[1] Lu, Shan, et al. "Ontology-based Detection of Inconsistencies in UML/OCL Models." MODELSWARD. 2022.

## APPENDIX A – PUBLICATIONS AND PRESENTATIONS

Lu, Shan, et al. "Ontology-based Detection of Inconsistencies in UML/OCL Models." MODELSWARD. 2022.
Chen, Y., Kokar, M.M. & Moskal, J.J. “SPARQL Query Generator (SQG).” Journal on Data Semantics 10, 291–307 (2021).

## LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

| | |
|---|---|
| AADL | Architecture Analysis and Design Language |
| AFRL | Air Force Research Laboratory |
| AOR | area of responsibility |
| API | Application Programming Interfaces |
| AWS | Amazon Web Services |
| C5ISR&T | Command, Control, Communications, Computers, Cyber, Intelligence, Surveillance, Reconnaissance, and Targeting |
| CAB | Cloud Agility Baseline |
| CI/CD | Continuous Integration (CI) and Continuous Deployment (CD) |
| DARPA | Defense Advanced Research Projects Agency |
| DDD | Domain-Driven Design |
| DL | Description Logic |
| E2 | Enterprise Engine |
| EDP | Elemental Design Patterns |
| EFS | Elastic File System |
| HLA/DIS | High-Level Architecture/Distributed Interactive Simulation |
| IDAS | Intent-Defined Adaptive Software |
| IEEE | Institute of Electrical & Electronics Engineers |
| JADC2 | Joint All-Domain Command and Control |
| JCITS | Joint Cyber Intelligence Tool Suite |
| LLS | Lightweight Logging Specification |
| MBSE | Model Based Systems Engineering |
| OCL | Object Constraint Language |
| OMG | Object Management Group |
| OWL | Web Ontology Language |
| PI | Principal Investigator |
| PIN | Pattern Instance Notation |
| RDS | Relational Database Service |
| S3 | Simple Storage Service |
| SADL | Semantic Application Design Language |

| | |
|---|---|
| SNC | Sierra Nevada Corporation |
| SoS | System of Systems |
| SPARQL | SPARQL Protocol and RDF Query Language (Recursive Acronym) |
| SysML | Systems Modeling Language |
| TA | Technical Area |
| TLA+ | Temporal Logic of Actions |
| UML | Unified Modeling Language |
| UxAS | Unmanned Systems Autonomy Services |
| VPC | Virtual Private Cloud |
| XMI | XML Metadata Interchange |
| XML | Extensible Markup Language |