Evolution of C4I Systems*

M. Harn, V. Berzins, Luqi, W. Kemple

Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

+1 831 6562615, +1 831 6562610, +1 831 6562735, +1 831 6563309

Fax: +1 831 6563225

{harn, berzins, luqi}@cs.nps.navy.mil, kemple@nps.navy.mil

Evolution of C4I Systems*

M. Harn, V. Berzins, Luqi, W. Kemple
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

{harn, berzins, luqi}@cs.nps.navy.mil, kemple@nps.navy.mil

Abstract

This paper formalizes evolution of C4I systems via a relational hypergraph model with primary-input-driven and secondary-input-driven dependencies. The evolution of C4I system is modeled by a multidimensional architecture containing successive software evolution steps and related software evolution components. We analyze a domain-specific software development architecture and give a standard software evolution process in developing a prototype system as well as a production software system. This model is applied in several real-time prototyping systems especially for Command and Control applications.

1. Introduction

The fundamental purpose of this paper is to reduce the development and evolution cost of C4I (Command, Control, Communication, Computer and Intelligence) systems (For this paper, C4I systems include C3I, C3 and C2 systems). The maintenance cost of C4I systems is hard to estimate because the user requirements are unstable [Entin *et al.*, 1998] [Kemple *et al.*, 1998]. The user requirement instability comes from many factors, such as hardware platform replacement and communication channel constraints, as well as social, political, and cultural changes. The software technology of C4I systems lags far behind hardware development. Software development failures have reached staggering proportions: an estimated $81 billion was spent on cancelled software projects in 1995 and an estimated $100 billion in 1996 [Luqi and Goguen, 1997].

It is still true that industry is largely unsuccessful when it comes to building software in general and large software projects in particular. Of the 175,000 information technology projects underway in the U.S. as we speak, 31% overall will end in failure. For larger projects characteristic of enterprise development, 40% will be cancelled and another 33% will be completed significantly late and over budget. No industry, type of business, or company is immune [Roetzheim, 1998].

These problems are also reflected in developing related DoD C4I systems. The requirements of C4I systems are extremely unstable and changeable due to tactical surprises, changes in goals and missions, etc.

---

*This research was supported in part by the U. S. Army Research Lab under grant number 7DNAVYR010 and in part by the U.S Army Research Office under contract/grant number 38690.

C4I systems are clearly complex systems that change frequently thus creating demands for flexibility on the part of supporting software and supporting flexibility in the organizational architecture [Lee and Carley, 1998]. A variety of factors influence the performance of C2 architectures - technology, environmental stressors, rules of engagement and so on.

In order to improve the performance of alternative organizational structures in a simulated joint operational environment, an A2C2 (Adaptive Architectures for Command and Control) program has been ongoing for approximately four years [Kemple *et al.*, 1998]. Systematic approaches to C4I systems evaluation, including experimental design, scenario modification, planning and developing training materials, conducting player training, managing execution, and data collection, etc., work well to elicit evolution issues with new requirements.

Our research has explored systematic management and automation of evolution processes to improve the situation, including formalizing and automating evolution processes [Badr and Luqi, 1994] [Lieberherr and Xiao, 1993] [Luqi, 1990] [Luqi and Goguen, 1997] [Madhavji, 1992]; prototyping proposed software [Luqi and Ketabchi, 1988] [Luqi, 1989] [Luqi, 1992]; seeking the relationship among software evolution objects [Badr and Luqi, 1994] [Berzins *et al.*, 1997] [Luqi, 1990] [Seiter *et al.*, 1998]; and reusing software evolution components [Goguen *et al.*, 1996] [Harn *et al.*, 1999] [Roetzheim, 1998].

2. Features of C4I systems

C4I systems include the following preliminary features [Luqi, 1992]:

- Their use in strategic, operational, and tactical defense applications makes correctness and reliability critical.
- They are influenced by many people, by organizations, and by policies, so their requirements are complex and difficult to determine.
- Their design depends on techniques to guarantee that hard real-time constraints will be met both in large distributed systems connected by long-haul networks and in local distributed systems with many hardware structures.
- Their complex, dynamic interfaces make it almost impossible to deal with changes in requirements.

Computer hardware and software enhances the feasibility and functionality of C3I systems. Computers within C4I systems not only play an interface role with external platforms, but also provide real-time embedded software and intelligent software to support commanders in decision making, routine program processing, data computing, etc. C4I computer software is too large, complex, dedicated, intractable, and mutable to meet mission needs under the development circumstances [Lee and Carley, 1998]. As with any large system, their development is costly, and the current low productivity of software development aggravates the problem [Sommerville, 1996].

Due to rapid requirement changes in the evolution environment, we use the Computer-Aided Software Prototyping System (CAPS) to help develop C4I systems. CAPS is an easy to use, visual and integrated tool that can be used to rapidly design real-time applications utilizing its prototype system description language (PSDL) editor, reusable software database, program generator, real-time scheduler, and so on [Luqi and Ketabchi, 1988].

A generic C4I system includes the following external interfaces [Luqi, 1992]:

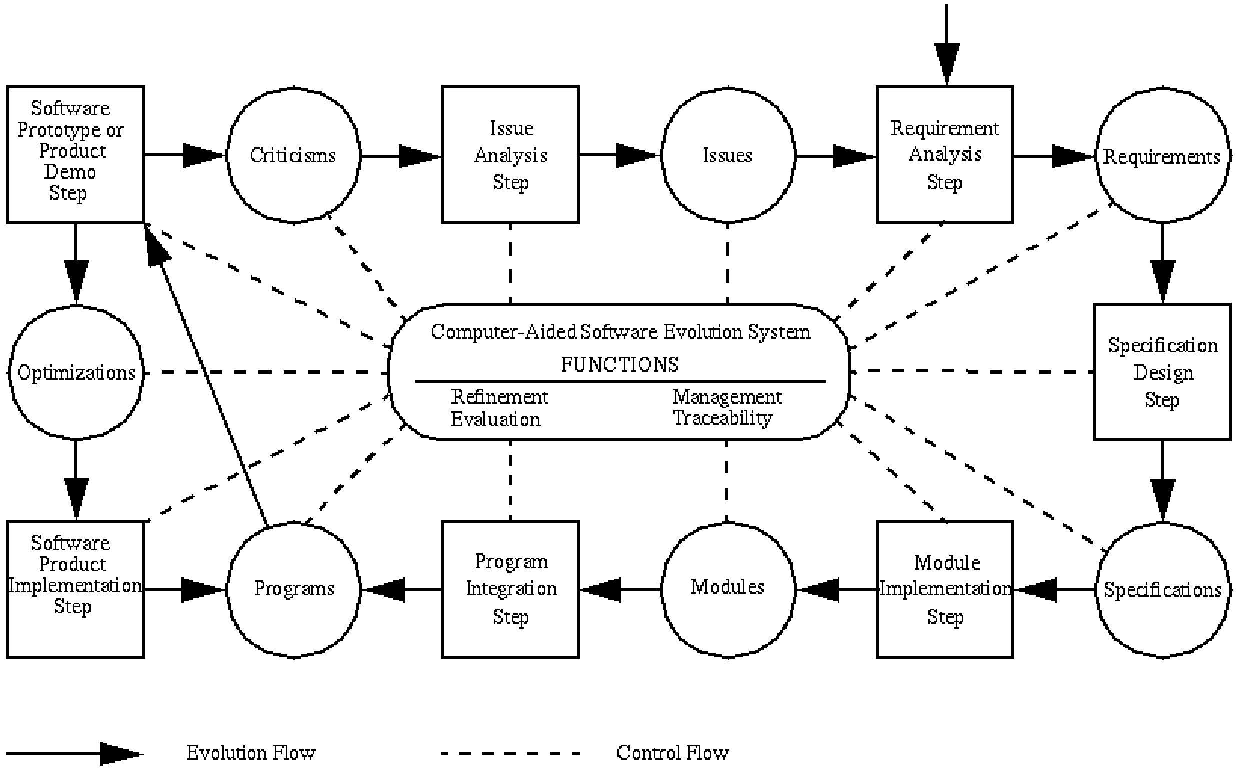
- Users: could be a composite warfare commander, officer in tactical command, warfare area commander, tactical action officer, communication officer, etc.
- Communication links: any digital communication system capable of transmitting and receiving digital messages.
- Platform sensors: any locally-mounted device capable of identifying azimuth, elevation, velocity, and/or heading if a contact or track is considered to be a platform.
- Navigation system: a system that provides a platform with own positioning, course, velocity, and time data.
- Weapons system: this interface, if exists, makes the weapons status information available to the battle manager.

The software with related data repositories of a generic C4I system is mounted in workstations. Software requirements are based on different and specific C4I systems.

3. Evolution processes of C4I systems

3.1 Formalizing evolution processes

In C4I systems, the objects affected by the software evolution processes are called *software evolution objects*. We classify *software evolution objects* into *software evolution steps* and *components*. Both kinds of objects can be hierarchically refined into finer grain objects by a relational hypergraph model (RH model). The leaf nodes of the refinement hierarchy are called *atomic objects*.



Software Prototype or Product Demo Step
Criticisms
Issue Analysis Step
Issues
Requirement Analysis Step
Requirements
Optimizations
Computer-Aided Software Evolution System
FUNCTIONS
Refinement
Evaluation
Management
Traceability
Specification Design Step
Software Product Implementation Step
Programs
Program Integration Step
Modules
Module Implementation Step
Specifications
Evolution Flow
Control Flow

Figure 1: Software Evolution Processes with CASES

According to the interactive, evaluation-centered user interaction development process [Hix and Hartson, 1993] [Luqi and Ketabchi, 1988] and the schematic model of the analysis process [Berzins *et al.*, 1997] modified from the Issue-Based Information Systems (IBIS) model [Conklin and Begeman, 1988], we have identified eight types of *top-level steps* in the software evolution process: *software prototype demo, issue analysis, requirement analysis, specification design, module implementation, program integration, software product demo, and software product implementation* [Harn *et al.*, 1999]. We have also identified seven different types of *top-level components* in the software evolution process: *criticisms, issues, requirements, specifications, modules, programs, and optimizations* [Harn *et al.*, 1999]. Each top-level object (including a step and a component) can be decomposed into a set of atomic objects, either directly or indirectly. The software evolution processes is shown in Figure 1.

3.2 Automating evolution processes

Computer-Aided Software Evolution System (CASES) is a system that manages and controls all the activities that change a software system and the relationships among these activities. The whole process for software evolution is described in Figure 1.

CASES is a set of software tools that perform the following four main functions: refinement, evaluation, management, and traceability. The structure of CASES is based on a formal model and interfaces to CAPS to manage and control the software evolution process for iterative software prototyping [Luqi and Ketabchi, 1988]. The evolution processes of C4I systems are illustrated as follows.

- At the beginning, users or system designers design the first version of a prototype of their proposed system using CAPS according to user requirements managed by CASES.

- In a demo step, after running the current version of the software prototype or product demo, customers record and send their criticisms to CASES via the network which can directly connect to the CASES hypertext database. This can be done using the Front Loaded Accurate Requirements Engineering (FLARE) system [Leonard *et al.*, 1997].
- In a criticism analysis step, system analysts collect and classify the criticisms and associate them with the related issues with the help of browsing and search capabilities of the hypertext database and communication with stakeholders as needed to clarify the intent of recorded criticisms.
- In a requirement analysis step, system analysts collect and classify the issues and refine the related requirements with database and communication support. This is a creative process that involves proposing and assessing plausible alternatives for responding to open issues.
- In a specification analysis step, system designers modify the PSDL corresponding to new requirements via the graphic user interface and editor of CAPS.
- In a module implementation step, system designers modify or implement the modules corresponding to the new PSDL via the graphic user interface and editor of CAPS.
- In a program integration step, system designers modify or implement the programs corresponding to the new modules, and integrate them using the CAPS.
- After the program integration step, the system prototype has evolved to the next version and can be demonstrated again to customers to assess its acceptability.
- The evolution process of prototyping systems can continue for many iterations until the final version of the prototype matches the customerÕs real requirements. After the demo step for the final prototyping system, software engineers evaluate the target operating environment and obtain the optimizations for the proposed software products.
- In the software product implementation step, system designers carry out proposed optimizations to design the deliverable software products.
- After the software product implementation step, the software products can be delivered to customers and be used by users. During the lifecycle of the C4I system, users may develop more criticisms, which are submitted according to appropriate policies. If the criticisms exceed some threshold, the project office may consider a maintenance upgrade. If a maintenance project is authorized, the process of software evolution will be started from the demo step again.

4. Hypergraphs preliminary

The RH model is based on the hypergraph model and the evolutionary hypergraph model [Harn *et al.*, 1999] [Luqi and Goguen, 1997].

4.1 Hypergraph model

Definition 1. (Hypergraph) A *(directed) hypergraph* is a tuple $H = (N, E, I, O)$ where

1. N is a set of *nodes,*
2. E is a set of *hyperedges* (briefly called *edges*),
3. $I: E \rightarrow 2^N$ is a function giving the set of *inputs* of each hyperedge, and
4. $O: E \rightarrow 2^N$ is a function giving the set of *outputs* of each hyperedge.

This definition describes a bare structure of a hypergraph. The traceability of the software evolution can be presented via the path of a hypergraph.

A *path* in the hypergraph represents an evolution history whose components, including nodes and hyperedges, can be traced.

4.2 Evolutionary hypergraph model

In the evolutionary hypergraph model [Luqi and Goguen, 1997], the software evolution components and steps have been identified by nodes and edges respectively. The attributes of the components and the steps must be recorded and labeled.

Definition 2. (Evolutionary Hypergraph) An *evolutionary hypergraph* is a labeled, directed, and acyclic hypergraph $H = (N, E, I, O)$ together with label functions $L_N : N \to C$ and $L_E : E \to A$ such that the following assumptions are satisfied:

1. The elements of N represent unique identifiers for software evolution components;
2. The elements of E represent unique identifiers for software evolution steps;
3. The functions I and O give the inputs and outputs of each software evolution step, such that $O(e) \cap O(e') \neq \emptyset$ implies $e = e'$;
4. The function L_N labels each node with *component attributes* from the set C, including the corresponding version of the software evolution component;
5. The function L_E labels each edge with *step attributes* from the set A, including the current status of the software evolution step, such that $A = \{s, d\} \cdot A'$ (that is, each element of A has the form (s, a') or (d, a'), where $a' \in A'$.) and an edge labeled *"s"* is called a *step* and one labeled *"d"* is called a *decomposition*.

According to this definition, both components and steps can be refined into finer components and steps. In particular, the minimal hypergraph whose edge set has only one edge can also be refined into a finer hypergraph [Luqi, 1992].

5. RH model

A *relational hypergraph* is an evolutionary hypergraph in which the *dependency* relationships between components and steps can have a hierarchy of specialized interpretations. These can be used to refine the software evolution process model.

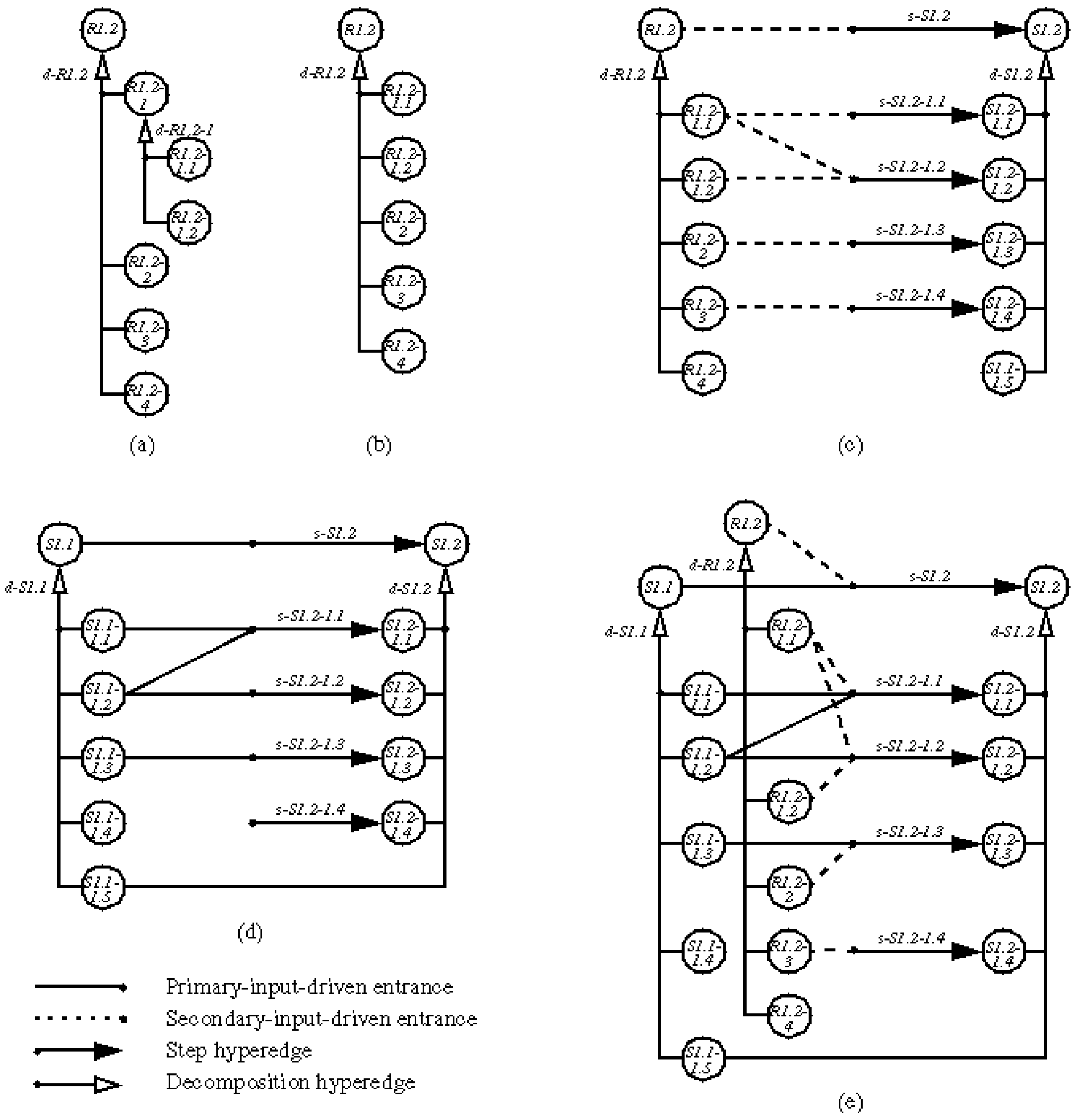


Figure 2: Relational hypergraphs

For example, we can distinguish between primary-input-driven paths and secondary-input-driven paths. The input part to each hyperedge in a path could be a set of multiple input nodes containing many kinds of software evolution components. If there exist an input node and an output node to an evolutionary hyperedge that are different versions of the same component then the path from the input node via the hyperedge to the output node is called a *primary-input-driven path*, the hyperedge is called a *primary hyperedge*, and the relationship between the input node and the step is called a *primary_input dependency*. If there exist an input node and an output node to an evolutionary hyperedge that are different components then the path from the input node via the hyperedge to the output node is called a *secondary-input-driven path*, the hyperedge is called a *secondary hyperedge*, and the relationship between the input node and the step is called a *secondary_input dependency*.

Because input nodes to a step come from different entrances, a step in a relational hypergraph can be represented by an arrow with multiple tails that are primary inputs or secondary inputs. The structure of relational hypergraphs is an unlimited-depth hierarchy. In order to formalize the relationship among software evolution objects and software evolution tasks represented by steps, we focus on top-level and atomic-level relational hypergraphs. Some atomic-level objects are too big to be realized as monolithic components; therefore, these objects must be decomposed. After they are decomposed, middle-level relational hypergraphs can be ignored because the software evolution objects in these levels are clusters of lower-level objects that only represent a concept of sets instead of interpreting their dependencies.

In Figure 2 (a) and (b), node *R1.2* is a top-level requirement that includes five atomic-level requirements *R1.2-1.1, R1.2-1.2, R1.2-2*, *R1.2-3*, and *R1.2-4*. Figure 2 (a) shows a three layer relational hypergraph whose node *R1.2-1* is a middle-level requirement. Figure 2 (b) is an atomic level representation of (a).

Each input to a step *s* is an individual input and can be connected together with others. For example, inputs *R1.2* and *S1.1* to the step *s-S1.2* in Figure 2 (c) and (d), respectively, are individual inputs and they can be connected together with each other shown as Figure 2 (e). Figure 2 (c) and (d) are partial views or projects of (e). This feature can be applied to both top-level and atomic level steps.

6. C4I/MD system evolution

A C4I system called Missile Defense (MD) system has been developed by CAPS. The MD system provides defense functions to a specified area or a nation such that it can be extended to a TMD (Theater Missile Defense) system and a NMD (National Missile Defense) system. TMD and NMD systems are extremely complicated; however, we need to know how system requirements are obtained. The first prototype MD system is very simple and immature, but it gradually gets complicated and feasible as the MD system goes through the evolution process several times.

Initially, the assumptions of a MD system are as follows:

- There are ten bases where certain kind of land-to-air missiles are deployed.
- Radar systems can accurately detect target objects.
- The radar coordinate system is 360 degrees increasing counterclockwise.
- The coordinate system is two-dimensional.
- The path of target objects is straight to an attack destination.
- All the attack destinations of target objects are on the center of a map which is the origin in the coordinate system.
- In one execution process, the MD system simulates only one target object and one defending missile.
- The speed of target objects and defending missiles is constant.
- The safety point is specified on a safety ring.
- The intersection point of target objects and defending missiles is on the safety point.
- Defending missiles can accurately destroy target objects at the safety point.
- There is no contingency plan in the case the missile fails to destroy the target object.
- Commanders take time making decisions with the computer.
- The position of target objects, the speed of target objects, the position of missile bases, and decision delay time are manually manipulated by users.

Assumptions can be generalized and specialized according to requirements from users. Users provide criticisms by their own view with different generalization and specialization ideas. The issue analysts collect criticisms into different generation and specialization issues. The requirement analysts provide some information about requirements, such as cost-benefit analysis and resource constraints, to managers. The decisions made by managers to new requirements of next generation prototype decides whether assumptions are generalized or specialized.

In the first prototype of MD systems, some of assumptions are transferred to requirements. In the second prototype of MD systems, some other assumptions will be released into new requirements.

In evolution processes of MD systems, criticism, issue, and requirement components can be described as hypermedia types, such as text, pictures, video, etc.; specification components can be described as data flow diagrams in PSDL editor and software programs; module and program components can be described as software programs.

6.1 The initial prototype of MD systems

- Requirement analysis step:

The initial prototype requirements can be generated by the requirement analysis step from the above assumptions. The top-level requirement component *R1.1* is a set of atomic requirement components that are categorized into three groups: *R1.1-1*, *R1.1-2*, and *R1.1-3*. Requirement components are presented as the following text descriptions stored by individual files:

R1.1-1: The MD system must provide an output interface for users to monitor the data concerning base position, target position, target situation, missile direction, missile speed, target and missile intersection point, current time, and missile reach time.
    R1.1-1.1: The output interface must provide the function of monitoring the base position data.
    R1.1-1.2: The output interface must provide the function of monitoring the target position data.
    R1.1-1.3: The output interface must provide the function of monitoring the target situation data.
    R1.1-1.4: The output interface must provide the function of monitoring the missile direction data.
    R1.1-1.5: The output interface must provide the function of monitoring the missile speed data.
    R1.1-1.6: The output interface must provide the function of monitoring the target and missile intersection point data.
    R1.1-1.7: The output interface must provide the function of monitoring the current time data.
    R1.1-1.8: The output interface must provide the function of monitoring the missile reach time data.
R1.1-2: The MD system must provide an input interface for users to enter the data concerning base location, target location, target speed, and delay time of decision making.
    R1.1-2.1: The input interface must provide the function of entering the base location data.
    R1.1-2.2: The input interface must provide the function of entering the target location data.
    R1.1-2.3: The input interface must provide the function of entering the target speed data.
    R1.1-2.4: The input interface must provide the function of entering the delay time of decision making data.
R1.1-3: The MD system must provide a control system capable of efficiently transferring, generating, receiving, and computing information in real time.
    R1.1-3.1: The control system must provide the function of transferring base locations to base coordinates.
    R1.1-3.2: The control system must provide the function of transferring target locations to target coordinates and computing coordinates of safety point.
    R1.1-3.3: The control system must provide the function of receiving data of base coordinates, target coordinates, coordinates of safety point, target speed, and delay time of decision making; and computing data of base position, target position, missile direction, missile speed, target and missile intersection point, and missile reach time.
    R1.1-3.4: The control system must provide the function of generating system time.
    R1.1-3.5: The control system must provide the function of receiving data of missile reach time and system time; and computing data of missile reach time, current time, and target situation.

- Specification design step:

The initial prototype specifications can be generated by the specification design step from above atomic requirement components. Top-level specification component *S1.1* is a set of atomic specification components that are categorized into two groups: *S1.1-1* and *S1.1-2*. Specification components are presented as following specification files with IMPLEMENTATION and SPECIFICAION PSDL code:

- *S1.1-1: c4i.gui_3.imp.psdl & c4i.gui_3.spec.psdl*
  - *S1.1-1.1: c4i.b_p_68.imp.psdl & c4i.b_p_68.spec.psdl*
  - *S1.1-1.2: c4i.t_p_71.imp.psdl & c4i.t_p_71.spec.psdl*
  - *S1.1-1.3: c4i.t_a_47.imp.psdl & c4i.t_a_47.spec.psdl*
  - *S1.1-1.4: c4i.m_d_38.imp.psdl & c4i.m_d_38.spec.psdl*
  - *S1.1-1.5: c4i.m_s_41.imp.psdl & c4i.m_s_41.spec.psdl*
  - *S1.1-1.6: c4i.int_35.imp.psdl & c4i.int_35.spec.psdl*
  - *S1.1-1.7: c4i.c_t_50.imp.psdl & c4i.c_t_50.spec.psdl*
  - *S1.1-1.8: c4i.m_r_t_44.imp.psdl & c4i.m_r_t_44.spec.psdl*
  - *S1.1-1.9: c4i.b_l_56.imp.psdl & c4i.b_l_56.spec.psdl*
  - *S1.1-1.10: c4i.t_l_59.imp.psdl & c4i.t_l_59.spec.psdl*
  - *S1.1-1.11: c4i.t_s_62.imp.psdl & c4i.t_s_62.spec.psdl*
  - *S1.1-1.12: c4i.d_t_65.imp.psdl & c4i.d_t_65.spec.psdl*
  - *S1.1-1.13: c4i.gui_event_monitor_53.imp.psdl & c4i.gui_event_monitor_53.spec.psdl*
- *S1.1-2: c4i.ctrl_6.imp.psdl & c4i.ctrl_6.spec.psdl*
  - *S1.1-2.1: c4i.trans1_114.imp.psdl & c4i.trans1_114.spec.psdl*
  - *S1.1-2.2: c4i.trans2_117.imp.psdl & c4i.trans2_117.spec.psdl*
  - *S1.1-2.3: c4i.ctrller_120.imp.psdl & c4i.ctrller_120.spec.psdl*

  - *S1.1-2.4: c4i.time_gen_126.imp.psdl & c4i.time_gen_126.spec.psdl*
  - *S1.1-2.5: c4i.target_123.imp.psdl & c4i.target_123.spec.psdl*

The IMPLEMENTATION and SPECIFICAION PSDL code is automatically generated by CAPS through data flow diagrams in the PSDL editor. In Figure 3 a top-level data flow diagram *c4i* includes *gui* and *ctrl* operators with 4 data streams from operator *gui* to operator *ctrl* and 8 data streams from operator *ctrl* to operator *gui*. Furthermore, Figure 4 shows a decomposed data flow diagram - *gui* including 13 operators and their data streams, and Figure 5 shows a decomposed data flow diagram *ctrl* including 5 operators and their data streams.

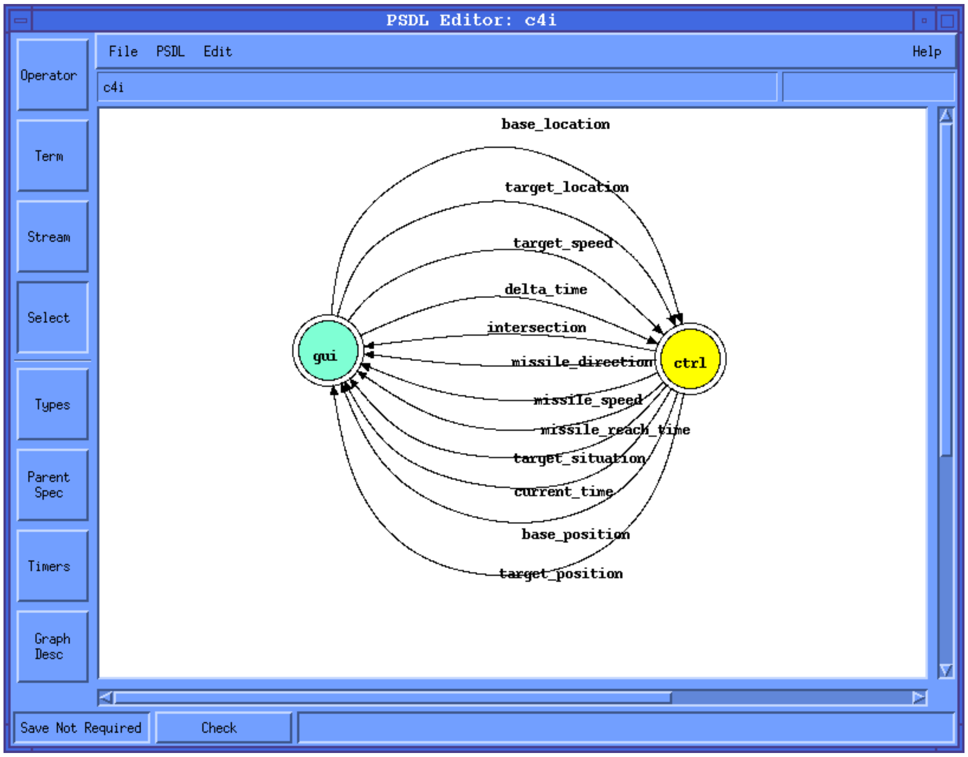


Figure 3: A top-level data flow diagram - *c4i*

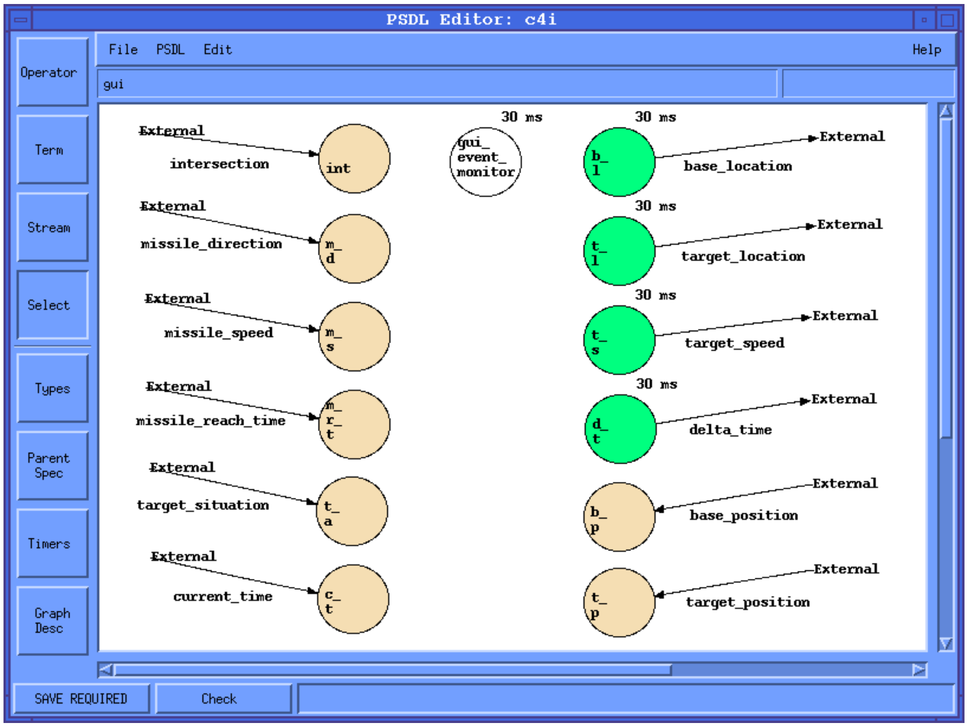


Figure 4: A decomposed data flow diagram - *gui*

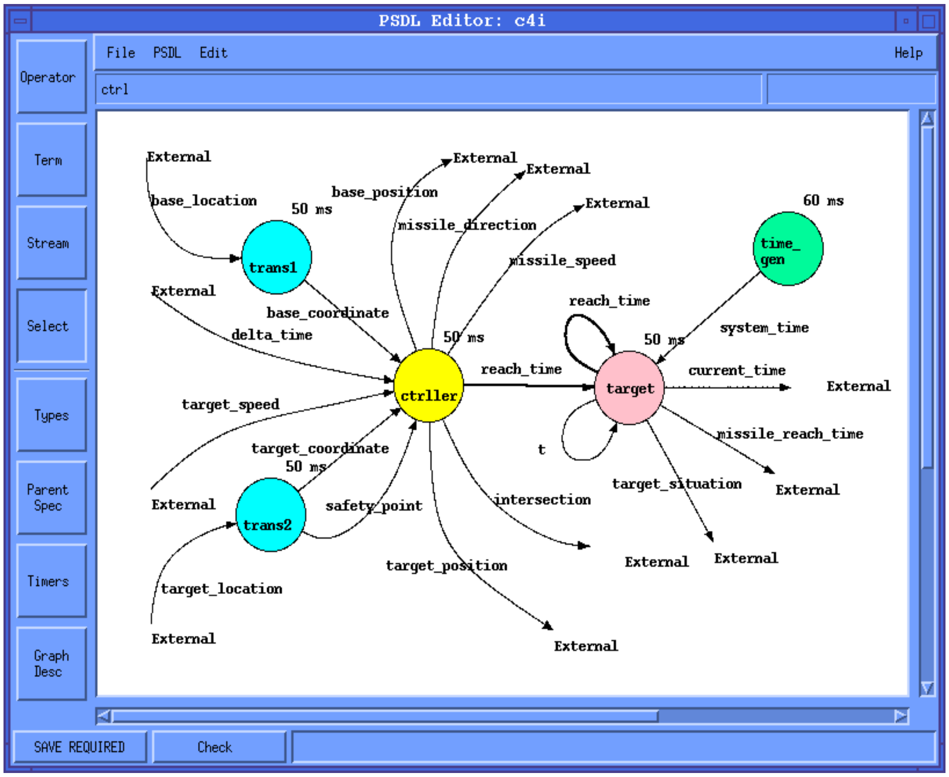


Figure 5: A decomposed data flow diagram - *ctrl*

- Module implementation step:

The initial prototype modules can be generated by the module implementation step from the above atomic specification components. The top-level module component *M1.1* is a set of atomic module components that are categorized into two groups: *M1.1-1* and *M1.1-2*. Files in *M1.1-1* are automatically generated by CAPS through TAE Plus (Transportable Applications Environment Plus). Files with extension file name "*.at*" are automatically generated by CAPS from related specification components. Files with extension file name "*.a*" are implemented by designers. Therefore, atomic module components are presented as the following files with ada code:

M1.1-1: Interface
  M1.1-1.1: c4i.b_p_68.a
  M1.1-1.2: c4i.t_p_71.a
  M1.1-1.3: c4i.t_a_47.a
  M1.1-1.4: c4i.m_d_38.a
  M1.1-1.5: c4i.m_s_41.a
  M1.1-1.6: c4i.int_35.a
  M1.1-1.7: c4i.c_t_50.a
  M1.1-1.8: c4i.m_r_t_44.a
  M1.1-1.9: c4i.b_l_56.a
  M1.1-1.10: c4i.t_l_59.a

M1.1-1.11: c4i.t_s_62.a
M1.1-1.12: c4i.d_t_65.a
M1.1-1.13: c4i.gui_event_monitor_53.a & c4i.gui_event_monitor_53.at
M1.1-2: Controller
M1.1-2.1: c4i.trans1_114.a & c4i.trans1_114.at
M1.1-2.2: c4i.trans2_117.a & c4i.trans2_117.at
M1.1-2.3: c4i.ctrller_120.a & c4i.ctrller_120.at
M1.1-2.4: c4i.time_gen_126.a & c4i.time_gen_126.at
M1.1-2.5: c4i.target_123.a & c4i.target_123.at

- Program integration step:

The initial prototype programs can be generated by the program integration step from the above atomic module components. The top-level program component *P1.1* is a set of atomic program components that are categorized into four groups: *P1.1-1*, *P1.1-2*, *P1.1-3*, and *P1.1-4*. Files in *P1.1* are automatically integrated, scheduled, compiled, and executed by CAPS. Atomic program components are presented as the following files with Ada code:

P1.1-1: Input
P1.1-1.1: c4i.b_p_68.a
P1.1-1.2: c4i.t_p_71.a
P1.1-1.3: c4i.t_a_47.a
P1.1-1.4: c4i.m_d_38.a
P1.1-1.5: c4i.m_s_41.a
P1.1-1.6: c4i.int_35.a
P1.1-1.7: c4i.c_t_50.a
P1.1-1.8: c4i.m_r_t_44.a
P1.1-2: Output
P1.1-2.1: c4i.b_l_56.a
P1.1-2.2: c4i.t_l_59.a
P1.1-2.3: c4i.t_s_62.a
P1.1-2.4: c4i.d_t_65.a
P1.1-3: GUI event monitor
P1.1-3.1: c4i.gui_event_monitor_53.a
P1.1-4: Controller
P1.1-4.1: c4i.trans1_114.a
P1.1-4.2: c4i.trans2_117.a
P1.1-4.3: c4i.ctrller_120.a
P1.1-4.4: c4i.time_gen_126.a
P1.1-4.5: c4i.target_123.a

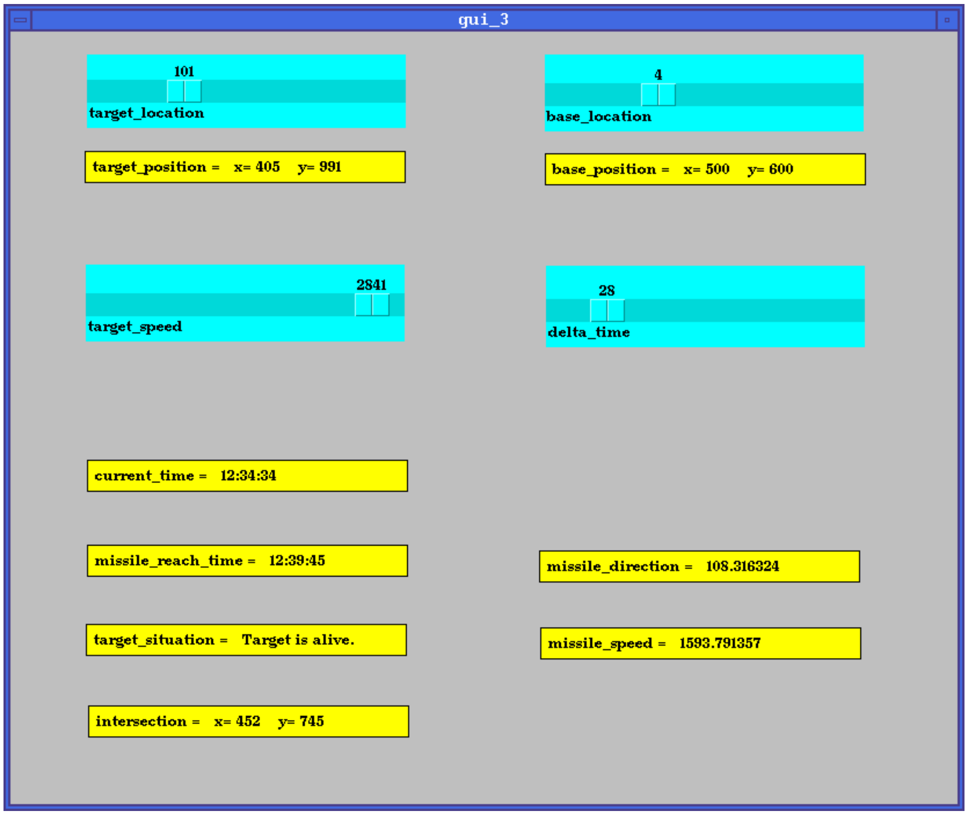
6.2 The second prototype of MD systems

- Software prototype demo step:

Criticisms to be addressed in the second prototype can be generated by the software prototype demo step from the above atomic program components. The top-level criticism component *C1.2* is a set of atomic criticism components that are categorized into six groups: *C1.2-1*, *C1.2-2*, *C1.2-3*, *C1.2-4*, *C1.2-5*, and *C1.2-6*. The criticism components are presented as the following text descriptions stored by individual files:

C1.2-1 MD system must consider the 3d situation.
C1.2-1.1 The target position must consider the 3d situation.

  - *C1.2-1.2 The missile intersection point must consider the 3d situation.*
- *C1.2-2 The coordinate system of MD system must include the height.*
  - *C1.2-2.1 There is no height at target positions.*
  - *C1.2-2.2 There is no height at missile intersection points.*
- *C1.2-3 It is hard to understand units of measurement, target and missile tracks, and degree system via the interface.*
  - *C1.2-3.1 The interface must include unit of measurement.*
  - *C1.2-3.2 The target and missile tracks must be shown graphically.*
  - *C1.2-3.3 The degree system should provide options for the location of the origin (0 degree).*
- *C1.2-4 This version of MD system is too simple.*
  - *C1.2-4.1 The is no height at missile base positions.*
  - *C1.2-4.2 The safety point must include 3d.*
  - *C1.2-4.3 The distance unit system should provide options for kilometers and nautical miles.*
  - *C1.2-4.4 MD system must consider multiple targets and missiles.*
  - *C1.2-4.5 MD system must provide the virtual reality image.*
- *C1.2-5 MD system should consider the data of missile numbers, missile types, and missile speeds in each base.*
  - *C1.2-5.1 MD system must consider the number and type of missiles in a base to help the commander make the optimal decision.*
  - *C1.2-5.2 MD system must show data about available missile speed and type in each base, after detecting the target.*
- *C1.2-6 This version of MD system can not protect the theater well.*
  - *C1.2-6.1 MD system must consider failure to hit the target object and launch another missile.*
  - *C1.2-6.2 MD system must provide the function to recognize the target object and predict the attack point of target object.*
  - *C1.2-6.3 MD system must suggest to the commander what kind of missile to launch from which base.*



101
target_location

4
base_location

target_position = x= 405 y= 991

base_position = x= 500 y= 600

2841
target_speed

28
delta_time

current_time = 12:34:34

missile_reach_time = 12:39:45

missile_direction = 108.316324

target_situation = Target is alive.

missile_speed = 1593.791357

intersection = x= 452 y= 745

Figure 6: A demo panel of the initial prototype program

Figure 6 shows a demo panel of the initial prototype program. Information from external parts linked to the MD systems, such as communication links, platform sensors, navigation system, and weapons system, is simulated. Therefore, target location, target speed, base location, and command delay delta times are manually manipulated by stakeholders.

- Issue analysis step:

The second prototype issues can be generated by the issue analysis step from above atomic criticism components. Top-level issue component *I1.2* is a set of atomic issue components that are categorized into seven groups: *I1.2-1*, *I1.2-2*, *I1.2-3*, *I1.2-4*, *I1.2-5*, *I1.2-6*, and *I1.2-7*. Issue components are presented as the following text descriptions stored by individual files:

I1.2-1 MD system must consider the 3d situation.
  I1.2-1.1 The target position must consider the height.
  I1.2-1.2 The missile intersection point must consider the height.
  I1.2-1.3 The missile base position must consider the height.
I1.2-2 MD system must employ uniform units of measurement.
  I1.2-2.1 The distance unit system must allow options for kilometers and nautical miles.
  I1.2-2.2 The distance unit system must be kilometers or nautical miles.
  I1.2-2.3 The degree system should provide options for the location of the origin (0 degree).
I1.2-3 MD system must have a friendly user interface.
  I1.2-3.1 The user interface must provide degree marks.
  I1.2-3.2 The target and missile tracks must be shown graphically.
I1.2-4 MD system must include detailed data on target objects and missile.
  I1.2-4.1 MD system must consider multiple targets and missiles.
  I1.2-4.2 MD system must consider the number and type of missiles in a base to help the commander make the optimal decision.
  I1.2-4.3 MD system must show data about the speed and type of available missiles in each base, after detecting the target.
I1.2-5 MD system must consider failure to hit target object.
  I1.2-5.1 MD system must consider failure to hit the target object and launch another missile.
  I1.2-5.2 MD system must have the capacity to know whether the missile hit the target object or not.
  I1.2-5.3 MD system must consider how many missiles are left in each base.
I1.2-6 MD system must consider missile launch automation and optimization.
  I1.2-6.1 MD system must provide the function to recognize the target object and predict the attack point of the target object.
  I1.2-6.2 MD system must suggest to the commander what kind of missile to launch from which base.
  I1.2-6.3 MD system must provide the function to automatically launch missiles to attack target objects according to intelligence from sensors.
  I1.2-6.4 MD system must provide the function to simulate the missile defense process and consider the missile launch optimization.
I1.2-7 It is hard to design a virtual reality image in CAPS.
  I1.2-7.1 TAE does not provide a virtual reality design environment.
  I1.2-7.2 Virtual reality design can be considered in different software development platform.

- Requirement analysis step:

The second prototype requirements can be generated by the requirement analysis step from the above issues. Some issues are able to generalize or specialize new requirements but some issues are not included in new requirements after validating the requirements. Validating requirements is the process through which the customersÕ real needs are checked against the formalized requirements to make sure that the formalized requirements accurately meet those needs. Top-level requirement component *R1.2* is a set of atomic requirement components that are categorized into two groups: *R1.2-1*, and *R1.2-2*. Requirement components are presented as the following text descriptions stored by individual files:

R1.2-1: The MD system must employ uniform units of measurement.
  R1.2-1.1: The distance unit system must be nautical miles.
  R1.2-1.2: The degree system must locate 0 degrees (the origin) at the north, 90 degrees at the west, 180 degrees at the south, and 270 degrees at the east.
R1.2-2: The MD system must provide dynamic output graphic interface for users to monitor the base position, target position, missile direction, missile speed, target and missile intersection point.
  R1.2-2.1: The dynamic output graphic interface must provide the function of locating the base positions.
  R1.2-2.2: The dynamic output graphic interface must provide the function of monitoring the target position.
  R1.2-2.3: The dynamic output graphic interface must provide the function of monitoring the missile direction.
  R1.2-2.4: The dynamic output graphic interface must provide the function of monitoring the missile speed.
  R1.2-2.5: The dynamic output graphic interface must provide the function of monitoring the target and missile intersection point.
  R1.2-2.6: The dynamic output graphic interface must provide two movers: the target object and the missile object, that can be moved in the 2d coordinate system.
  R1.2-2.7: The dynamic output graphic interface must provide two rings: the target object approaching ring and the safety ring (also called the target and missile intersection ring).
  R1.2-2.8: The dynamic output graphic interface must provide grid coordinates, distance marks and degree marks.

- Specification design step:

The second prototype specifications can be generated by the specification design step from the above atomic requirement components. The top-level specification component *S1.2* is a set of atomic specification components that are categorized into two groups: *S1.2-1* and *S1.2-2*. The specification components are presented as the following specification files with IMPLEMENTATION and SPECIFICAION PSDL code:

S1.2-1: c4i.gui_3.imp.psdl & c4i.gui_3.spec.psdl
  S1.2-1.1: c4i.b_p_68.imp.psdl & c4i.b_p_68.spec.psdl
  S1.2-1.2: c4i.t_p_71.imp.psdl & c4i.t_p_71.spec.psdl
  S1.2-1.3: c4i.t_a_47.imp.psdl & c4i.t_a_47.spec.psdl
  S1.2-1.4: c4i.m_d_38.imp.psdl & c4i.m_d_38.spec.psdl
  S1.2-1.5: c4i.m_s_41.imp.psdl & c4i.m_s_41.spec.psdl
  S1.2-1.6: c4i.int_35.imp.psdl & c4i.int_35.spec.psdl
  S1.2-1.7: c4i.c_t_50.imp.psdl & c4i.c_t_50.spec.psdl
  S1.2-1.8: c4i.m_r_t_44.imp.psdl & c4i.m_r_t_44.spec.psdl
  S1.2-1.9: c4i.b_l_56.imp.psdl & c4i.b_l_56.spec.psdl
  S1.2-1.10: c4i.t_l_59.imp.psdl & c4i.t_l_59.spec.psdl
  S1.2-1.11: c4i.t_s_62.imp.psdl & c4i.t_s_62.spec.psdl
  S1.2-1.12: c4i.d_t_65.imp.psdl & c4i.d_t_65.spec.psdl
  S1.2-1.13: c4i.gui_event_monitor_53.imp.psdl & c4i.gui_event_monitor_53.spec.psdl
  S1.2-1.14: c4i.merge_233.imp.psdl & c4i.merge_233.spec.psdl
  S1.2-1.15: c4i.t_m_p_236.imp.psdl & c4i.t_m_p_236.spec.psdl
  S1.2-1.16: c4i.m_p_239.imp.psdl & c4i.m_p_239.spec.psdl
S1.2-2: c4i.ctrl_6.imp.psdl & c4i.ctrl_6.spec.psdl

S1.2-2.1: *c4i.trans1_114.imp.psdl & c4i.trans1_114.spec.psdl*
S1.2-2.2: *c4i.trans2_117.imp.psdl & c4i.trans2_117.spec.psdl*
S1.2-2.3: *c4i.ctrller_120.imp.psdl & c4i.ctrller_120.spec.psdl*
S1.2-2.4: *c4i.time_gen_126.imp.psdl & c4i.time_gen_126.spec.psdl*
S1.2-2.5: *c4i.target_123.imp.psdl & c4i.target_123.spec.psdl*
S1.2-2.5: *c4i.movers_250.imp.psdl & c4i.movers_250.spec.psdl*

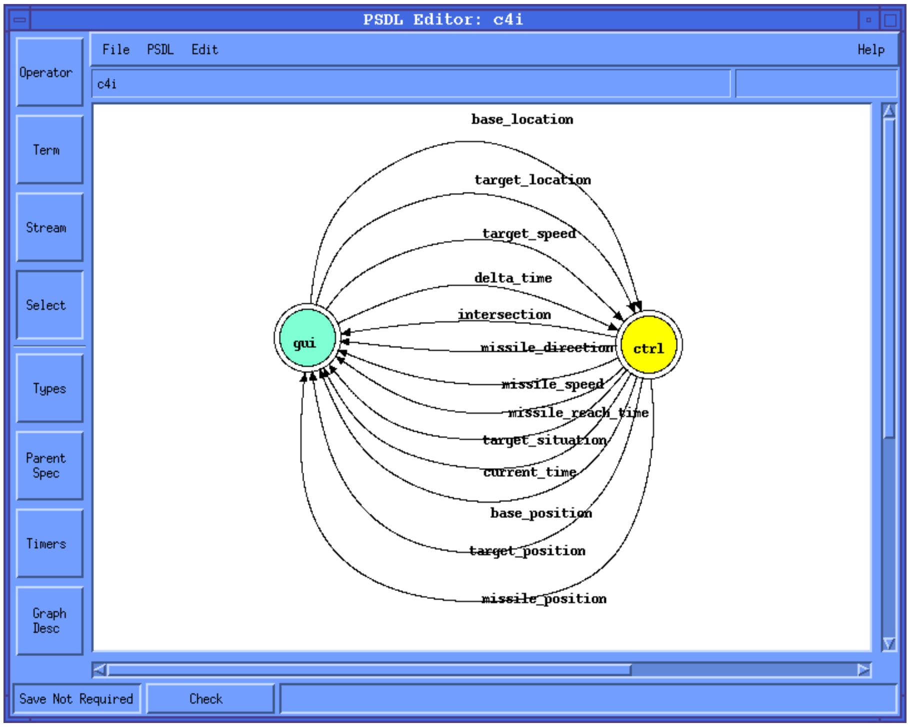


Figure 7: A top-level data flow diagram - *c4i(modified)*

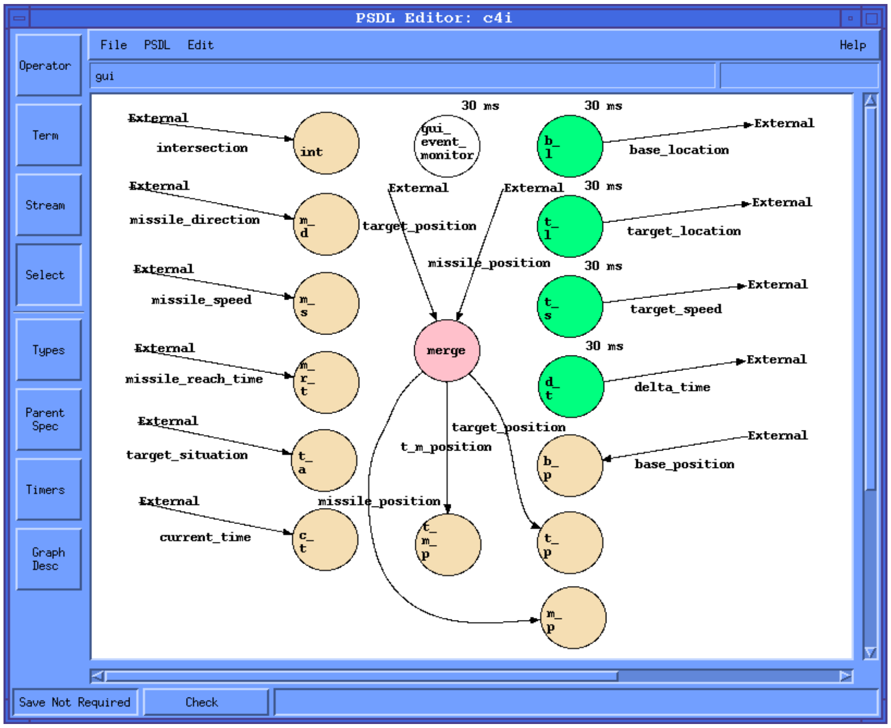


Figure 8: A decomposed data flow diagram - *gui(modified)*

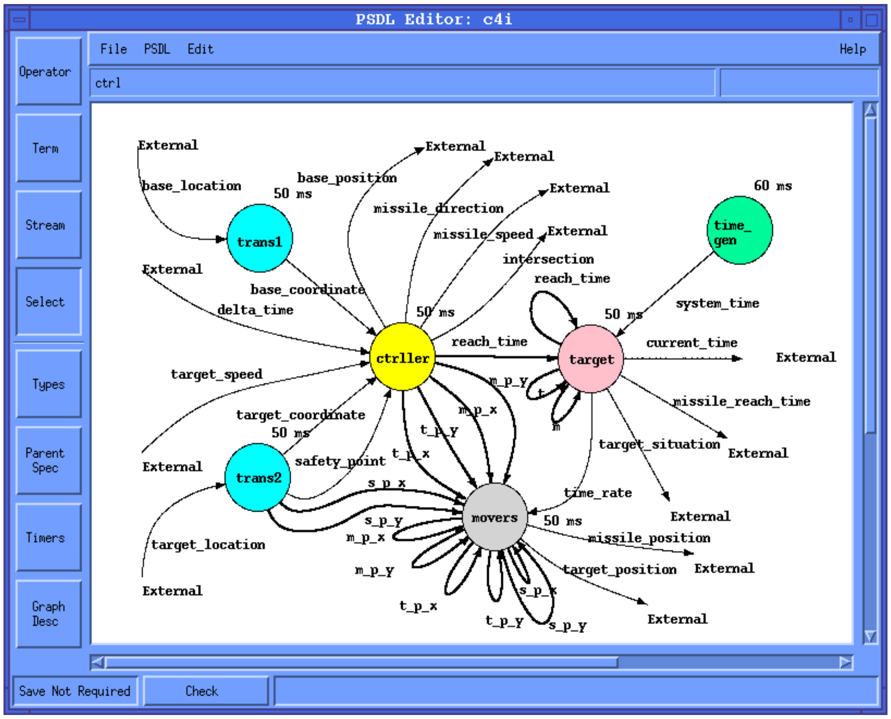


Figure 9: A decomposed data flow diagram - *ctrl(modified)*

In Figure 7, the top-level data flow diagram *c4i* is modified by one new data stream missile-position. Furthermore, Figure 8 shows a decomposed data flow diagram *gui*, modified by 3 new operators (*merge, t_n_p,* and *m_p*) and their related data streams, and Figure 9 shows a decomposed data flow diagram *ctrl* modified by one new operator (movers) and its related data streams.

- Module implementation step:

The second prototype modules can be generated by the module implementation step from the above atomic specification components. The top-level module component *M1.2* is a set of atomic module components that are categorized into two groups: *M1.2-1* and *M1.2-2*. The atomic module components are presented as the following files with ada code:

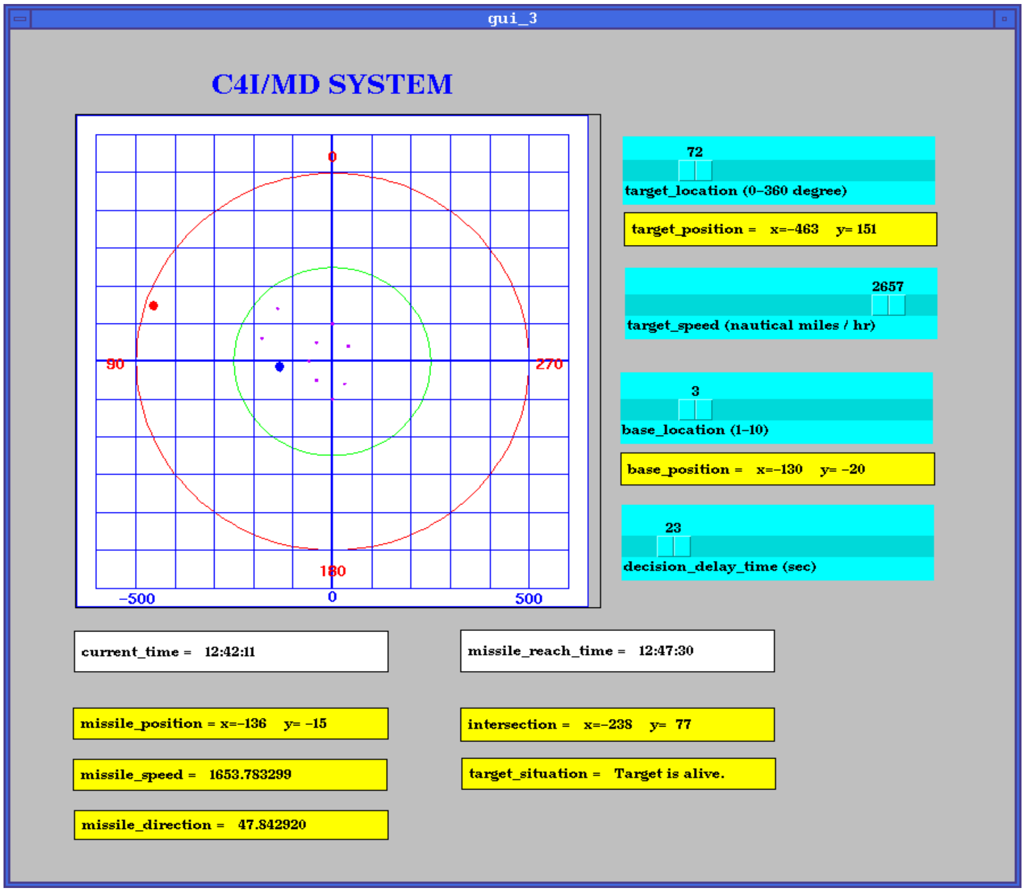
- *M1.2-1: Interface*
  - *M1.2-1.1: c4i.b_p_68.a*
  - *M1.2-1.2: c4i.t_p_71.a*
  - *M1.2-1.3: c4i.t_a_47.a*
  - *M1.2-1.4: c4i.m_d_38.a*
  - *M1.2-1.5: c4i.m_s_41.a*
  - *M1.2-1.6: c4i.int_35.a*
  - *M1.2-1.7: c4i.c_t_50.a*
  - *M1.2-1.8: c4i.m_r_t_44.a*
  - *M1.2-1.9: c4i.b_l_56.a*
  - *M1.2-1.10: c4i.t_l_59.a*
  - *M1.2-1.11: c4i.t_s_62.a*
  - *M1.2-1.12: c4i.d_t_65.a*
  - *M1.2-1.13: c4i.gui_event_monitor_53.a & c4i.gui_event_monitor_53.at*
  - *M1.2-1.14: c4i.merge_233.a*
  - *M1.2-1.15: c4i.t_m_p_236.a*
  - *M1.2-1.16: c4i.m_p_239.a*
- *M1.2-2: Controller*
  - *M1.2-2.1: c4i.trans1_114.a & c4i.trans1_114.at*
  - *M1.2-2.2: c4i.trans2_117.a & c4i.trans2_117.at*
  - *M1.2-2.3: c4i.ctrller_120.a & c4i.ctrller_120.at*
  - *M1.2-2.4: c4i.time_gen_126.a & c4i.time_gen_126.at*
  - *M1.2-2.5: c4i.target_123.a & c4i.target_123.at*
  - *M1.2-2.6: c4i.merge_233.a & c4i.merge_233.at*

- Program integration step:

The second prototype programs can be generated by the program integration step from the above atomic module components. The top-level program component *P1.2* is a set of atomic program components that are categorized into four groups: *P1.2-1, P1.2-2, P1.2-3,* and *P1.2-4*. The atomic program components are presented as the following files with ada code:

- *P1.2-1: Input*
  - *P1.2-1.1: c4i.b_p_68.a*
  - *P1.2-1.2: c4i.t_p_71.a*
  - *P1.2-1.3: c4i.t_a_47.a*
  - *P1.2-1.4: c4i.m_d_38.a*
  - *P1.2-1.5: c4i.m_s_41.a*
  - *P1.2-1.6: c4i.int_35.a*
  - *P1.2-1.7: c4i.c_t_50.a*

P1.2-1.8: c4i.m_r_t_44.a
P1.2-2: Output
P1.2-2.1: c4i.b_l_56.a
P1.2-2.2: c4i.t_l_59.a
P1.2-2.3: c4i.t_s_62.a
P1.2-2.4: c4i.d_t_65.a
P1.2-2.5: c4i.merge_233.a
P1.2-2.6: c4i.t_m_p_236.a
P1.2-2.7: c4i.m_p_239.a
P1.2-3: GUI event monitor
P1.2-3.1: c4i.gui_event_monitor_53.a
P1.2-4: Controller
P1.2-4.1: c4i.trans1_114.a
P1.2-4.2: c4i.trans2_117.a
P1.2-4.3: c4i.ctrller_120.a
P1.2-4.4: c4i.time_gen_126.a
P1.2-4.5: c4i.target_123.a
P1.2-4.6: c4i.merge_233.a



target_location (0–360 degree) 72
target_position = x=–463 y= 151
target_speed (nautical miles / hr) 2657
base_location (1–10) 3
base_position = x=–130 y= –20
decision_delay_time (sec) 23
current_time = 12:42:11
missile_reach_time = 12:47:30
missile_position = x=–136 y= –15
intersection = x=–238 y= 77
missile_speed = 1653.783299
target_situation = Target is alive.
missile_direction = 47.842920

Figure 10: A demo panel of the second prototype program

Figure 10 shows a demo panel of the second prototype program. In addition to new output items and unit marks in the panel, there are two movers, a target object approaching ring, a safety ring, and ten missile base positions in the two dimension coordinate system. It has been improved from the initial prototype program. Further iteration may be required until the prototype program fully meets the users' requirements (Figure 1).

7. Conclusion

Different evolution processes of C4I systems decompose evolution activities in different ways [Luqi and Goguen, 1997]. Our formal model emphasizes a domain-specific software development architecture. There are some difficulties in formalizing domains that are subject to frequent requirements changes. The RH model with prototyping has resolved some of the problems in evolution processes of C4I systems, such as evolution path traceability, object management, process description, and requirements analysis.

References

[Badr and Luqi, 1994] S. Badr and Luqi, Automation Support for Concurrent Software Engineering, *Proceeding of the 6th International Conference on Software Engineering and Knowledge Engineering,* Jurmala, Latvia, June 20-23, 1994, pp. 46-53.

[Berzins *et al.*, 1997] V. Berzins, O. Ibrahim, and Luqi, A Requirements Evolution Model for Computer Aided Prototyping, *Proceedings of the 9th International Conference on Software Engineering and Knowledge Engineering*, Madrid, Spain, June 17-20, 1997, pp. 38-47.

[Conklin and Begeman, 1988] J. Conklin and M. Begeman, gIBIS: A Hypertext Tool for Exploratory Policy Discussion, *ACM Transactions on Office Information System*, Vol. 6, October 1988, pp. 303-331.

[Entin *et al.*, 1998] E. E. Entin, D. Serfaty, and C. Kerrigan, Choice and Performance Under Three Command and Control Architectures, *Command and Control Research and Technology Symposium*, Naval Postgraduate School, California, June 29 - July 1, 1998, pp. 66-77.

[Goguen *et al.*, 1996] J. Goguen, D. Nguyen, J. Meseguer, Luqi, D. Zhang, and V. Berzins, Software Component Search, *Journal of Systems Integration*, Vol. 6, 1996, pp. 93-134.

[Harn *et al.*, 1999] M. Harn, V. Berzins, and Luqi, Software Evolution via Reusable Architecture, *Proceedings of 1999 IEEE Conference and Workshop on Engineering of Computer-Based Systems*, Nashville, Tennessee, March 7-12, 1999, pp. 11-17.

[Hix and Hartson, 1993] D. Hix and H. R. Hartson, *Developing User Interfaces: Ensuring Usability through Product & Process*, John Wiley & Sons, 1993.

[Leonard *et al.*, 1997] T. Leonard, V. Berzins, Luqi, and M. J. Holden, Gathering Requirements from Remote Users, *Proceeding of the ninth International Conference on Tools with Artificial Intelligence*, Newport Beach, California, November 3-8, 1997, pp. 462-471.

[Lieberherr and Xiao, 1993] Karl J. Lieberherr and Cun Xiao, Object-Oriented Software Evolution, *IEEE Trans. on Software Engineering*, Vol. 19, No. 4, April 1993, pp. 313-343.

[Kemple *et al.*, 1998] W. G. Kemple, G. R. Porter, R. Benson, S. G. Hutchins, and S. Hocevar, Research in the Classroom and Simulation Laboratory: Combining C2 Research and Education, *Command and Control Research and Technology Symposium*, Naval Postgraduate School, California, June 29 - July 1, 1998, pp. 360-366.

[Lee and Carley, 1998] J. Lee and K. M. Carley, Adaptive Strategies for Improving C2 Performance, *Command and Control Research and Technology Symposium*, Naval Postgraduate School, California, June 29 - July 1, 1998, pp. 66-77.

[Luqi and Ketabchi, 1988] Luqi and M. Ketabchi, A Computer-Aided Prototyping System, *IEEE Software*, March 1988, pp. 66-72.

[Luqi, 1989] Luqi, Software Evolution Through Rapid Prototyping, *IEEE Computer*, May 1989, pp. 13-25.

[Luqi, 1990] Luqi, A Graph Model for Software Evolution, *IEEE Trans. on Software Engineering*, Vol. 16, No. 8, August 1990, pp. 917-927.

[Luqi, 1992] Luqi, Computer-Aided Prototyping for a Command-And-Control System Using CAPS, *IEEE Software*, January 1992, pp. 56-67.

[Luqi and Goguen, 1997] Luqi and J. A. Goguen, Formal Methods Promises and Problems, *IEEE Software*, January 1997, pp. 73-85.

[Madhavji, 1992] N. Madhavji, Environment Evolution: The Prism Model of Changes, *IEEE Trans. on Software Engineering*, Vol. 18, No. 5, May 1992, pp. 380-392.

[Roetzheim, 1998] W. H. Roetzheim, Formal Process Improvement - Ignore at Your Own Risk, Trends in Software Engineering Process Management, November 1998, http://www.marotz.com/journal/nov98/fpi.htm.

[Seiter *et al.*, 1998] L. Seiter, J. Palsberg, and K. Leiberherr, Evolution of Object Behavior Using Context Relations, *IEEE Trans. on Software Engineering*, Vol. 24, No. 1, January 1998, pp. 79-92.

[Sommerville, 1996] I. Sommerville, *Software Engineering*, Addison-Wesley, 1996.