ODP-based Improvements of C4ISR-AF*

Jan Øyvind Aagedal[①], Arne-Jørgen Berre[①], Bjørn W. Bjanger[①], Tor Neple[①], Camilla Bårnes Roark[②]

[①] SINTEF Telecom and Informatics, P.O.Box 124 Blindern, N-0314 Oslo, Norway. Tel: +47 22 06 73 00, fax: +47 22 06 73 50,
{joea | bjornb | ajb | tne}@informatics.sintef.no

[②] CCIS House, P.O. Box 452, N-1371 Asker, Norway. Tel: +47 66 76 58 00, fax: +47 66 90 36 60, cbr@ccis.no

Abstract

This paper introduces a reference architecture based on C4ISR-AF extended with concepts from ISO RM-ODP. The work is a result of experiences using C4ISR-AF and certain shortcomings identified (namely more focus on distribution aspects, security issues and an information model). RM-ODP was chosen because of its influence in the computer science society, and because it provides a consistent overall framework and foundation for describing distributed systems. An example using the UML notation is presented to obtain more insight in using the different models.

1. Introduction

Military operations often require the joint efforts of a mix of forces. The ability for the supporting information systems to interoperate is essential to succeed [Department of Defense, 1998].

During the last decades many military computer systems have been built, each with a specific purpose in mind. The results are mainly stand-alone systems that satisfy a set of requirements, but that are not able to communicate with other systems and are difficult to maintain.

As the computer industry has evolved and more high-level features are readily available, more focus has been put on interoperable systems that still provide the same basic functionality. It has become clear that most military units need the same subset of functions to support planning, report status, give orders, etc., and these functions should be coordinated. Specific functionality such as fire direction systems, logistics support, etc. can then be added as needed. However, the systems are large and complex and involve many intricacies. To succeed in building such systems, it is useful to build an architecture that contains the essential models of the systems and that can be used as a basis for adding functionality to each specific system. In addition, an architecture enforce that the systems are developed using the same concepts. From this, reusable elements can be discovered, requirements on interoperability imposed and potential areas of conflicts identified.

This paper is structured as follows: Section 1 defines the important terms, gives a brief introduction to RM-ODP and describes the importance of interoperability. Section 2 introduces

* The work reported herein is sponsored by the Norwegian Army Material Command.

the experience gained when building the first iteration of the Norwegian C2IS Architecture. Section 3 suggests improvements to C4ISR-AF and introduces a reference architecture with elements from C4ISR-AF and ISO RM-ODP. Finally, Section 4 gives examples of some of the models presented in Section 3 and how they correspond.

1.1 Terminology

In order to avoid confusion of the exact meaning of some important terms used, we define the semantics of the terms *system*, *architecture*, *architecture framework* and *reference architecture*.

A *system* is any part of the world we choose to regard as a whole. A system can consist of subsystems, each of which is a system in its own right.

In the context of command and control, the term system describes the collection of resources necessary to perform a function with defined goals. The term encompasses

- qualified personnel,
- infrastructure and hardware,
- specified routines and methods, and
- their support in information systems and proper equipment.

This wide use of the term can be a source to confusion, even more so when the definition is used recursively. Nevertheless, a system describes a collection of resources, routines, facilities and actors put together for a specific purpose in a specific context.

All systems have an *architecture* that can be viewed from a number of perspectives. For instance, security architecture, operational architecture and physical architecture are all perspectives of the architecture, all with different foci. An <X> architecture (where <X> denotes any one perspective on the architecture) contains a description of essential elements and relationships that systems should include with respect to <X>. Different systems that have the same <X> architecture are conformant with respect to <X> in that they follow the same guidelines and principles on this; hence, they contain the elements and relationships prescribed by the <X> architecture. For instance, two systems can both use third-party authentication as their authentication architecture; they then have the same authentication architecture even if they use two different third-parties for the actual authentication. As an architecture defines a set of perspectives (<X> architectures), it describes a coherent set of guidelines and principles a conformant system should follow. When creating a system according to this, the resultant system should meet all the properties of the constituent <X> architectures. The term architecture is in our context mostly used when discussing information systems, but with the broad definition of system above, systems like an organization will also have an architecture. The organizational hierarchy is one perspective on this architecture, a work process description is an other.

An *architecture framework* is a collection of generic <X> architectures (perspectives). While an architecture also consists of a set of <X> architectures (perspectives), they differ in level of abstraction. A system has an architecture that represents the essential aspects of that system, while an architecture has an architecture framework that represents the essential aspects of that architecture. Many architectures can follow the same architecture framework just like many systems can have the same architecture. An architecture framework can for instance prescribe that architectures should include a security and a physical architecture, that the security architecture should include the specification of security levels, that the physical architecture

should include a specification of nodes, and that the specification of communication between nodes should include security levels.

A *reference architecture* defines a set of useful terms and concepts to use when specifying architecture frameworks, architectures or parts thereof, together with some rules on how to structure the architectures or architecture frameworks. Based on definitions of important terms, a reference architecture could for instance include definitions of different <X> architectures and how they relate, e.g. what comprises an security architecture and a physical architecture, and how these relate.

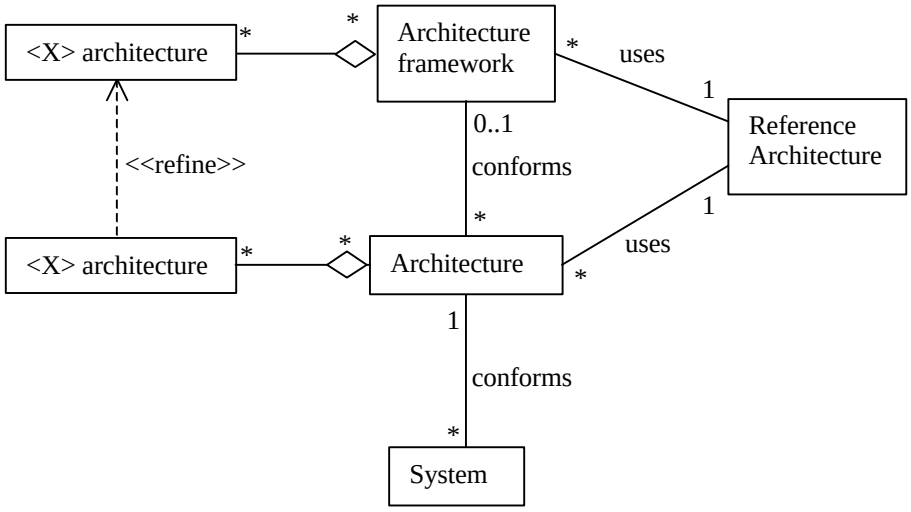

Figure 1. Relationships between architecture definition frameworks, reference architecture, architectures and systems

Using the UML notation, Figure 1 summarizes the relationship between systems, architectures, architecture frameworks and reference architecture. The architecture framework defines properties of some architectures that in turn define the properties of some systems. An architecture consists of a number of <X> architectures. Some of the <X> architectures are refinements of the corresponding <X> architectures of the architecture framework. Both the architecture frameworks and the architectures are defined using terms and concepts from the reference architecture, and may also be structured according to rules in the reference architecture.

1.2 RM-ODP

The Reference Model for Open Distributed Processing (RM-ODP) [ISO/IEC JTC1/SC21, 1995] is an ISO standard focusing on open distributed processing systems, and it is, in our terms, a reference architecture. It creates a framework within which support for distribution, inter-working, and portability can be integrated. RM-ODP defines a set of basic concepts and an analytical framework for normalized description of ODP-systems. It also contains the specification of the required characteristics that qualify distributed processing as open.

RM-ODP divides the specification of ODP systems into five different, but related, viewpoints. Each viewpoint focuses on some aspect of the systems, disregarding others. However, since each viewpoint specification is a specification of the same underlying system, concepts defined in one

are often related to concepts in other viewpoint specifications. The viewpoints in RM-ODP are: enterprise (focuses on purpose, scope and policies), information (focuses on information processing and relationships between information objects), computational (focuses on functional specification and decomposition), engineering (focuses on how to solve distribution issues), and technology (focuses on specific technology and solutions). In the first three viewpoints, a set of distribution transparencies are chosen (access, location, replication, migration, re-location, transaction, persistence, and failure), meaning that some of these issues can be regarded as transparent. How to solve each of the chosen transparencies is addressed in the engineering viewpoint.

1.3 Interoperability

Literally, interoperability means the ability for two or more systems to co-operate, i.e. work together in a meaningful sense. As such, interoperability is about being able to exchange information between the co-operating parties and to ensure that this information keeps the intended interpretation when passed on to the peer system and further to the end user. The information passed can be both a message containing information that holds some importance and meaning to someone, or it can be a procedure call between two parts of two systems, for instance two components exchanging information.

Posing a requirement of interoperability on two or more systems introduces a series of properties on the systems that need co-operate. This means that many aspects of a system need to be able to be compared with similar aspects in other systems and discrepancies identified. On one level, the systems must be physically and electronically connected. Second, they must be able to understand the messages being sent over this connection, for instance by using a common communication protocol. Third, they must be able to interpret the meaning of the data sent over the connection, that is, the number of bits sent must have a common interpretation. For instance may some data represent a procedure call on the remote system while other data may represent conveyed information. Finally, the data must be used in a common context so that the end users of each system have a shared view of what the systems do.

Interoperability is therefore an issue that must be treated on different abstraction levels and from different viewpoints, and we believe the five viewpoints of RM-ODP provide a good basis for discussion of various aspects of interoperability between systems. Interoperability aspects have different implications at the different viewpoints:

- **Technology viewpoint**: Describes the technologies chosen in the two systems. One needs to discuss how these technologies can be bridged.
- **Engineering viewpoint**: Is it possible to make the mechanisms chosen for communication and distribution play together? Can objects in the two systems communicate with each other?
- **Information viewpoint**: How is information modeled, stored and interpreted in the two systems? Can the data be converted between the formats without losing their meaning? For this, the two systems need to have a common understanding of at least parts of the world.
- **Computational viewpoint**: The computational viewpoint describes how the systems are decomposed, and which interfaces the components export. Do the systems already have a

notion of each other and have the knowledge necessary to call procedures in the other, or do there exist mechanisms for them to discover and interpret these interfaces at run-time?

- **Enterprise viewpoint**: How do the two organizations work, and in what contexts? Is it possible for a worker in one organization to put a message from the other side into its context, and give it the interpretation intended? Will the message then mean the same as it did when it was sent?

2. Experiences

The Norwegian Defense Research Establishment (NDRE) has strongly recommended that C2IS systems should be developed using an architecture-driven approach. The motivation for this is that the architecture will contribute to coherence between the different subsystems. This is true because the same models and terminology will be used to describe them. As a result of this, reuse will be encouraged mainly because it is easy to discover similarities in the system design. Furthermore, the life cycle costs will be reduced owing to the fact that parts of the system easily can be replaced. When the Norwegian Industry received the contract to design the Norwegian Army Tactical Command, Control and Information System there was an unspoken requirement that an Architecture Framework should be used. After an evaluation of the architectural frameworks C4ISR-AF and ISO RM-ODP, it was decided to use C4ISR-AF to build the Norwegian C2IS Architecture. There were several reasons for choosing C4ISR-AF:

- It is tailored for military systems
- It gives a good description of the products
- It emphasizes which products that are essential
- It gives good examples.

However we soon discovered that using the framework raised some questions:

- What process is used to develop a product?
- What are the dependencies between the products?
- How can we get from the Operational View to the Systems View?
- How do we maintain consistency in the Architecture?
- Which tools can we use?

Developing the Operational Architecture did not invite many problems. We chose to focus on OV-5: Activity Model to describe the functionality the system should support. The main discussion was on how to make the system as independent of the organization as possible. It was not clear how to achieve this using the concepts defined in C4ISR-AF. We introduced role-modeling [Reenskaug et al., 1996] to address this problem.

Using the products from the Operational Architecture to develop the System Architecture was a challenge. The problem was to achieve the correct level of detail and to maintain consistency between the products. A conceptual model was developed. This model explains how the different elements of the architecture are connected.

Today no tools are made that fully support architecture-driven development. As we had experience using RDD-100, which is a requirement driven system engineering tool, we used this to build the first iteration of the Norwegian C2IS Architecture. However, in time we will focus on using UML-notation and change to an UML-based tool. Using RDD-100 was not trivial, and it was an effort to make the tool support what we wanted the architecture to express.

In retrospect, after using C4ISR-AF to build the first iteration of the Norwegian C2IS Architecture, we believe we need to focus more strongly on distribution and security aspects, and on the information model. These aspects were problem areas, and by emphasizing them we hope to improve our architecture. We also believe it is important to build an architecture where the requirements are reflected in the system in a manner that makes them traceable.

3. Suggested improvements

C4ISR-AF divides the specification of a system into three different architecture views. It is our view that the approach of RM-ODP of refining it into five viewpoints is useful, and that the more object-oriented approach of RM-ODP is beneficial to the development of complex systems. Specifically, RM-ODP makes distribution an important concern, it makes the information viewpoint explicit, and it introduces the concept of roles.

Based on these observations, we created MACCIS (Minimal Architecture for CCIS in the Norwegian Army) [Neple et al., 1999]. MACCIS consists of a reference architecture that prescribes how systems following this architecture should be described and built, and a process of maintaining this reference architecture that includes procedures of how to handle changes in the reference architecture. This is shown in Figure 2.

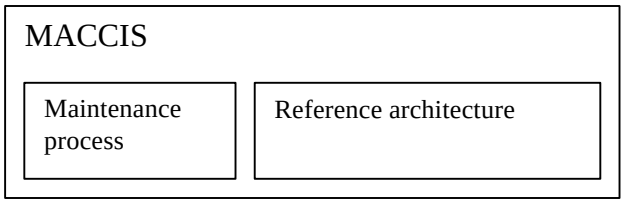

Figure 2. MACCIS

The reference architecture consists of a description of five sets of models and a process to develop these models. The models represent, when created, the essential parts of the final system. However, in the reference architecture itself only the models to create and their properties are described. The system developers then use these model descriptions when modeling and creating the real system. The reference architecture also contains a discussion of possible tool chains that can be used to develop both the models and the system itself. Finally, the reference architecture contains a number of standard elements in some of the models. In this version of the reference architecture there are no such standard elements, hence the name minimal architecture. Future versions will have an increasing number of standard elements included. These standard elements and their relationships will make the architecture framework for CCIS in the Norwegian Army, and represent one important tool to facilitate interoperability. The inclusion of these new elements is the most important activity that is regulated by the maintenance process.

The model descriptions that are part of the reference architecture are grouped into five different, but related, sets (herein called viewpoints). Each viewpoint focuses on specific aspects of the system, leaving others out. However, the models in each viewpoint represent aspects of the same underlying system, they only focus differently. The consistency between the models is therefore of particular importance; they all model the same system. Having overlap in the elements used in the models ensures consistency, and from this conflicts between models can be detected.

Consistency is best enforced by tools, hence requirements of a possible tool chain is discussed in the MACCIS-report. This is left out in this paper for brevity. Figure 3 shows the structure of the reference architecture, an important part of the minimal architecture.

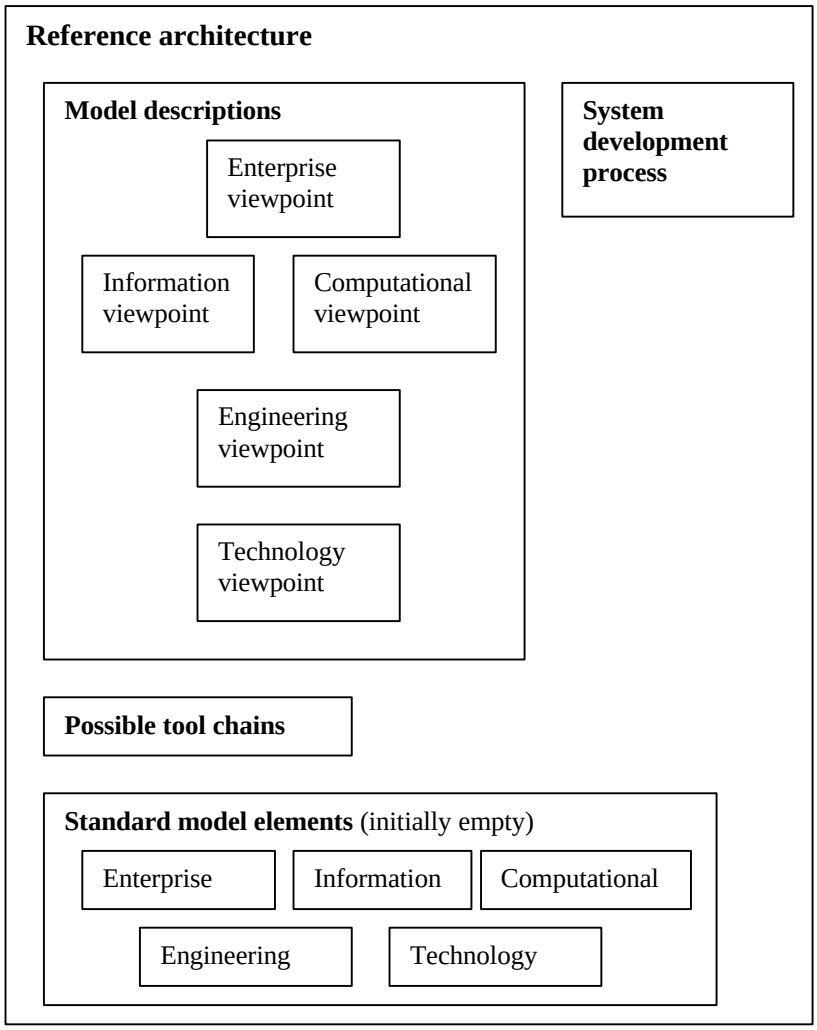

Figure 3. Reference architecture

System developers wanting to conform to the prescribed architecture are the primary users of this reference architecture. First, they must develop their system according to the system development process prescribed. Second, they get advice on which tools to use to develop their system. Third, they get a description of which models they must create when designing their system. Finally, they get a set of standard elements they must use in their system, and a description of the models in which these elements are represented.

However, system purchasers are also important users of MACCIS. First, by forcing the use of MACCIS, they ensure that the system developers follow a well-defined process in developing the system. Developing systems is an error prone and high-risk activity, by following a well-funded process it is more lightly that the developers will deliver the system in time, to the right price and with the right quality. Second, by forcing a set of standard elements to be used, interoperability between different systems using the same elements can be achieved. Third, by forcing the developers to model the system using the same set of models, reusable elements can

be discovered that others that use the same set of models can use at a later stage. Putting these elements into the standard set of elements can even enforce this.

3.1 Viewpoints

C4ISR-AF consists of three architecture views in which a number of deliverables (artifacts) is defined. We suggest specification of five viewpoints corresponding to the viewpoints of RM-ODP. Based on the artifacts from C4ISR-AF and the concepts in the five viewpoints of RM-ODP, we suggest specification of specific models in each viewpoint. The models identified form the basic set of models. In addition, a number of derived models can be created. These would not contain any new information, but rather combine information in new and potentially useful ways.

In [Neple et al., 1999], both process of how to develop the models and which UML-diagrams to use for expressing each model is indicated, this is left out in this paper for brevity.

Figure 4 shows all models in each viewpoint, and two cross-viewpoint documents.

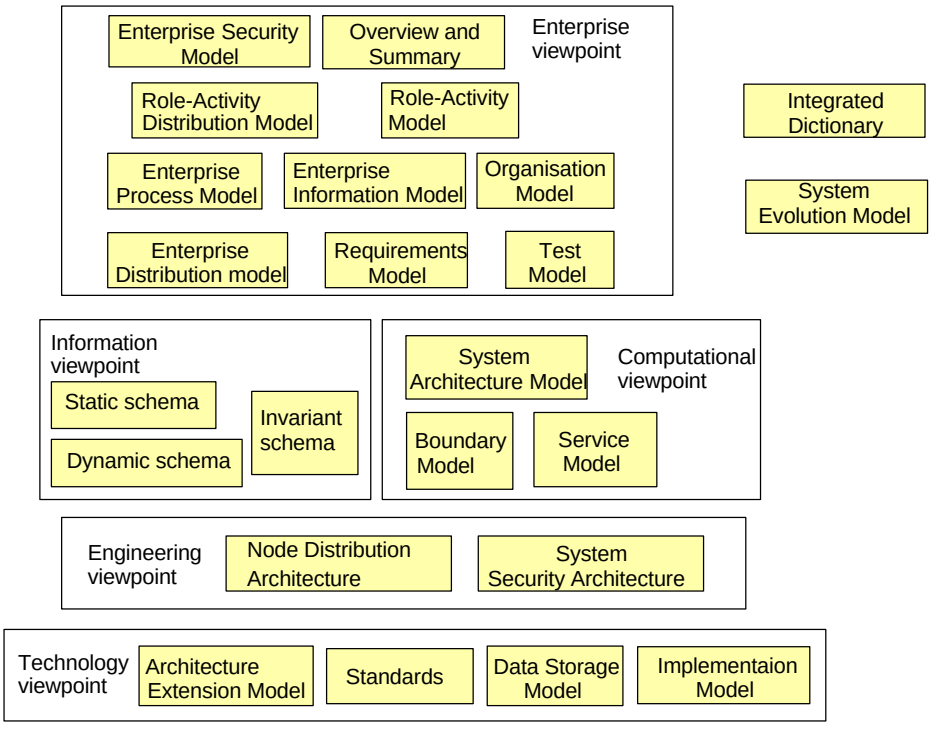

Figure 4. Models overview

Much of the information expressed in these models is also present in C4ISR-AF; the correspondence between our models and C4ISR-AF artifacts is discussed in the presentation of each viewpoint.

3.2 Cross-Viewpoint Documents

In addition to the models in the five viewpoints, two cross-viewpoint (C) documents exist. C1 is a dictionary that defines all terms used in all models. C2 is a system evolution model that contains descriptions of how versioning, configuration management, etc. are to be handled. This

includes both evolution of the system with its constituent models, and evolution of the enterprise the system is part of.

Document reference	Document name	Essential or supporting	General nature
C1	Integrated dictionary	Essential	Definitions of all terms used in all models
C2	System evolution model	Essential	System evolution policies and tools

Table 1: Cross-viewpoint documents

The integrated dictionary in C1 corresponds to AV-2 in C4ISR-AF. C2 has no corresponding artifact in C4ISR-AF.

3.3 Enterprise Viewpoint

Model reference	Model name	Essential or supporting	General nature
EV1	Overview and summary	Essential	Purpose, scope, intended users, environment depicted, high-level graphical representation of operational concept
EV2	Enterprise process	Essential	Activities, relationships between activities, constraints and other policies on the activities, information flow between activities.
EV3	Enterprise information	Essential	Enterprise information objects and their relationships
EV4	Organization	Essential	Organizational roles, their responsibilities, their interactions, and related policies
EV5	Distribution	Essential	Units of distribution within the enterprise
EV6	Role-activity	Essential	Responsibilities of organizational roles in terms of activities that must be performed. It combines activities with roles from the organizational model to show which roles have responsibility of executing which activities.
EV7	Role-activity-distribution	Essential	Organizational roles with their activities mapped to units of distribution. It combines the roles already combined with the activities they are responsible for in EV6, with the units of distribution.
EV8	Security	Essential	High-level security policies
EV9	Requirements	Supporting	Based on the previous models, this model summarizes the system requirements, hereunder QoS requirements.
EV10	Test	Supporting	Test model of system requirements, including use cases. It – outlines how the identified requirements can be tested in the system

Table 2: Enterprise Viewpoint Models

Specification of the enterprise viewpoint of a system is concerned with describing it from an enterprise (or business) point of view, and as such focuses on the purpose, scope and policies of the system.

This viewpoint (i.e. the models in this viewpoint) forms the basis for the other viewpoints in that the system requirements are identified here; the other viewpoints focus on how to meet these requirements.

3.3.1 The Enterprise Viewpoint in C4ISR-AF

C4ISR-AF has a number of architecture products that defines elements in the enterprise viewpoint. The following table summarizes where in C4ISR-AF elements from the Enterprise Viewpoint are represented:

Model reference	**C4ISR-AF Architecture Products**	**Comment**
EV1	AV-1, OV-1	One to one between EV1 and the union of AV-1 and OV-1
EV2	OV-5	One to one between these models
EV3	OV-7	EV3 contains the subset of OV-7 that represents enterprise objects
EV4	OV-1	High-level organization contained in OV-1, detailed role descriptions not in C4ISR-AF
EV5	OV-2	Units of distribution contained in OV-2
EV6	OV-2	Activities performed at each node contained in OV-2
EV7	OV-1	High-level mapping of organization to geographic configuration in OV-1, detailed mappings not in C4ISR-AF
EV8		Not in C4ISR-AF
EV9		Not explicit in C4ISR-AF
EV10		Not in C4ISR-AF

Table 3: Correspondence between Enterprise Viewpoint and C4ISR-AF

3.4 Computational Viewpoint

Model reference	**Model name**	**Essential or supporting**	**General nature**
CV1	Architecture	Essential	System components and their collaboration
CV2	Boundary	Essential	User interfaces
CV3	Service	Essential	Design models of service components

Table 4: Computational Viewpoint Models

Specification of the computational viewpoint is concerned with describing the computational objects and their interactions. The computational viewpoint is based on entities (objects) and activities (processes) identified in the enterprise viewpoint.

The computational viewpoint defines the distribution-independent design of the system. It specifies how the functionality of the system is divided into components, how these components interact and how each such component offering services are designed. It also provides user interface design.

3.4.1 The Computational Viewpoint in C4ISR-AF

C4ISR-AF has a number of architecture products that defines elements in the computational viewpoint. The following table summarizes where in C4ISR-AF elements from the computational viewpoint are represented:

Model reference	**C4ISR-AF Architecture Products**	**Comment**
CV1	SV-1, SV-3, SV-4	CV1 contains SV-1 with the communication between components as represented in SV-3 and the information flow from SV-4
CV2		Not represented in C4ISR-AF
CV3	SV-7, SV-10a-c	CV3 contains the design of components as represented in SV-10a-c, along with QoS constraints from SV-7

Table 5: Correspondence between Computational Viewpoint and C4ISR-AF

3.5 Information Viewpoint

Model reference	**Model name**	**Essential or supporting**	**General nature**
IV1	Static schema	Essential	Facts about information true at a given point in time
IV2	Invariant schema	Essential	Statements always true about the information
IV3	Dynamic schema	Essential	Information processing

Table 6: Information Viewpoint Models

The information viewpoint of a system is concerned with describing the information objects and information processing in the system. The information viewpoint is based on the entities identified in the Enterprise Viewpoint, for instance EV3, and information flow identified as types for parameters etc in the systems viewpoint.

The invariant schema describes facts about the information content of a system that must be true at any time. For instance "the completion date of a project must not be set to a date before the start date of the project". It is important to note that the invariant schema should be created first, as it naturally sets constraints for the other parts of the information schema.

The static schema describes facts about the information in a system at a given point in time. For instance "upon creation the balance of an account is zero".

The dynamic schema describes how the information in the system changes as the system is used. This in essence means a description of how the state of information objects change as operations in the component or object interfaces are called.

To illustrate the state changes, state diagrams should be created for the operations that make essential changes to the information. Pre- and post-conditions for the operations are also a part of this work product, and can be depicted as formal statements or plain text. The pre- and post-conditions can be attached to the interface description in the computational viewpoint, but are in essence a part of the information viewpoint.

3.5.1 Information Viewpoint in C4ISR-AF

C4ISR-AF has a number of architecture products that are related to the information aspects of the architecture. The following table summarizes where in C4ISR-AF elements from the Information Viewpoint are represented:

Model reference	**C4ISR-AF Architecture Products**	**Comment**
IV1	OV-7	OV-7 defines the logical information model of C4ISR-AF. Parts of the information in IV1-3 are not represented by C4ISR-AF.
IV2	OV-7	
IV3	OV-7	

Table 7: Correspondence between Information Viewpoint and C4ISR-AF

3.6 Engineering Viewpoint

The engineering viewpoint consists of two architectures: Security Architecture and Node Distribution Architecture.

3.6.1 Security Architecture

This architecture description should describe the different security mechanisms in use in the system. In EV8, a set of high-level constraints or requirements to the security aspects of the system at hand is described. Within the Engineering Viewpoint, the choices of security mechanisms and how these are used is described. Specifically, mechanisms that address the following issues should be discussed (see Security Service in CORBA[OMG, 1996]): identification and authentication, authorization and access control, security auditing, security of communication, non-repudiation, and administration of security information.

Some of the security functionality mentioned above uses other types of security mechanisms such as cryptography. Such underlying mechanisms should also be discussed, but the focus is on the listed mechanisms that support transparencies defined in the computational and information viewpoint. While having a stable set of security mechanisms, the underlying mechanisms such as cryptography can be changed without affecting the outside view of security.

Model reference	Model name	Essential or supporting	General nature
ES1	System security	Essential	Security mechanisms that support the security policies

Table 8: Security Architecture in Engineering Viewpoint

3.6.2 Node Distribution Architecture

The node distribution architecture uses the distribution model in the enterprise viewpoint, and refines this to include specifications of how the distribution concerns are addressed. The distribution architecture illustrates how the RM-ODP distribution transparencies used in the other viewpoints (information and system) are supported. The following transparencies should be addressed: access, location, failure, migration, relocation, transaction, persistence, and replication.

Support for each of the distribution transparencies can be specified as general patterns of solution using collaboration diagrams in UML. These patterns can then be deployed in single instances in the solution using collaborations as patterns in UML. This is specified in ED1.

In addition to specify how the distribution issues are addressed, the node distribution architecture also refines the distribution model as specified in the enterprise viewpoint (EV5) to include physical nodes, that is, computational units where the components reside. This is reflected in ED2. After the nodes have been identified, one needs to map components from the systems viewpoint down to these physical nodes. A mapping from the system architecture as specified in SV1 to the physical distribution nodes is done in ED3.

Model reference	Model name	Essential or supporting	General nature
ED1	Distribution patterns	Supporting	General solutions to distribution transparencies
ED2	Physical nodes	Essential	Distribution units refined into physical nodes
ED3	Component node	Essential	Components from system view mapped to physical nodes

Table 9: Distribution Architecture in Engineering Viewpoint

Note the differences between the table and Figure 4 in that the node distribution architecture in the engineering viewpoint in the figure comprises ED2-3 and the system security architecture in the figure is defined in ES1. ED1 is not represented in Figure 4.

3.6.3 Engineering Viewpoint in C4ISR-AF

Model reference	C4ISR-AF Architecture Products	Comment
ES1	OV-7	Security policies related to information only.
ED1		Not addressed in C4ISR-AF
ED2	SV-2	
ED3	OV-2, SV-2	

Table 10: Models in the Engineering Viewpoint

3.7 Technology Viewpoint

Model reference	Model name	Essential or supporting	General nature
TV1	Standards	Essential	Use of standards and infrastructures
TV2	Architecture extension	Essential	Extensions to architecture (CV1) for chosen technology
TV3	Data storage	Essential	Physical data model on chosen platform
TV4	Implementation	Essential	Hardware, code and documentation

Table 11: Models in the Technology Viewpoint

The Technology viewpoint, as the name suggests, is concerned with describing the technological viewpoints of the system. This includes a wide variety of aspects.

Note that the actual system is part of this viewpoint, TV4 represents the running code and infrastructure for the system.

3.7.1 Technology Viewpoint in C4ISR-AF

Model reference	C4ISR-AF Architecture Products	Comment
TV1	TV-1, TV-2	
TV2		Not represented in C4ISR-AF
TV3	SV-11	
TV4		Not represented in C4ISR-AF

Table 12: Correspondence between Technology Viewpoint and C4ISR-AF

4. Example

The intention of this example is to show some of the different models presented in this paper and how they correspond. The area of concern is to model how an Artillery fire mission is accomplished and to provide a Fire Direction System for the user.

The first step is to understand the domain and how the system to be developed will help the users. The figure below shows the actors involved in a fire mission, and then a brief description of them follows.

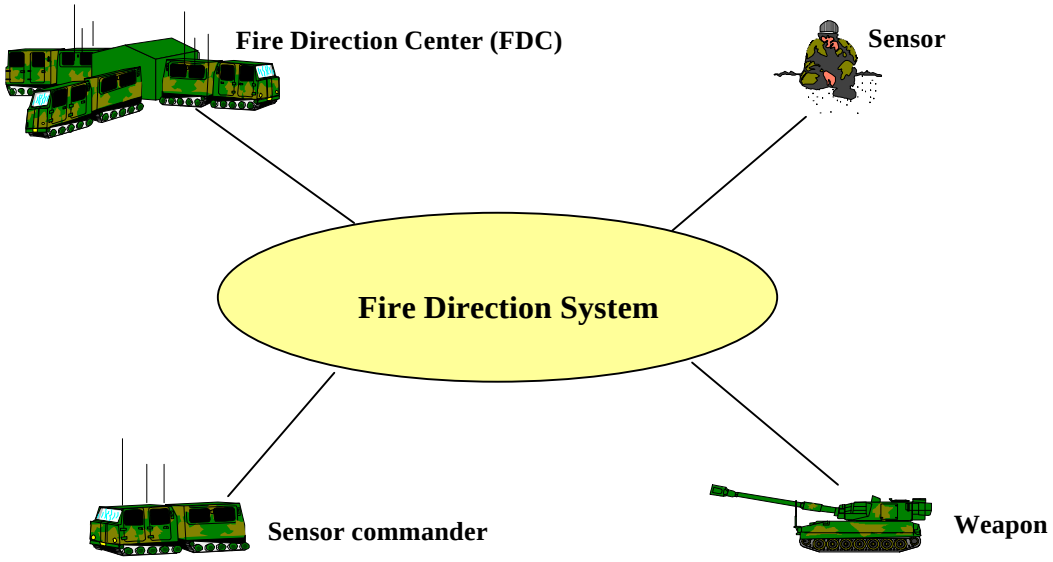

Figure 5. EV1 - Overview and Summary

- **Sensor:** a sensor can be a forward observer, artillery hunting radar or a special forward observer (artillery hunter). Their job is to describe a target as specifically as possible and then send a call for fire to their Sensor commander.
- **Sensor Commander:** When a call for fire is received, the sensor commander will control it against the own troops' positions, fire co-ordination lines and areas, and relevant orders and directives, in order to determine whether the call for fire should be acted upon or not. If not denied, the call for fire is evaluated to decide which Fire Direction Center (FDC) it should be sent to.
- **FDC:** The artillery has several FDCs; one for each battery, battalion and one in the artillery regiment. A FDC is selected to be the lead FDC for the specific fire mission, depending upon the fire power necessary to fight a target.
- **Weapon:** Guns and MLRS-launchers receive fire missions from their respective FDCs.

An organization model often helps the understanding of the domain. Figure 6 indicates the different organization elements and the role they play. The lines indicate communication between the different elements. The figure does not say how many elements of each type that exist in the organization. Note that domain-specific symbols are used. Domain experts are used to expressing their domain models using these symbols, and these symbols are used here to facilitate communication with them.

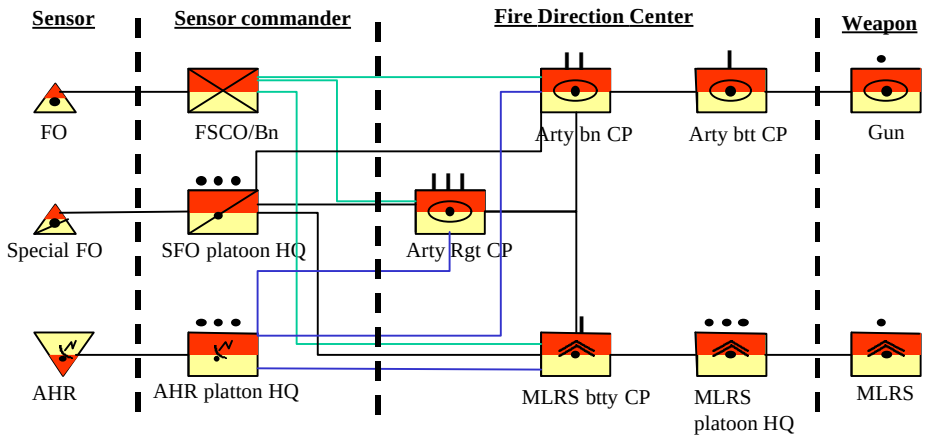

Figure 6. EV4 - Organization Model

After understanding the domain, the main activities, and the basic organization, an enterprise process model can be made. In this case a UML activity diagram is used, see Figure 7. EV2 represents the business processes that lay within the scope defined in EV1.

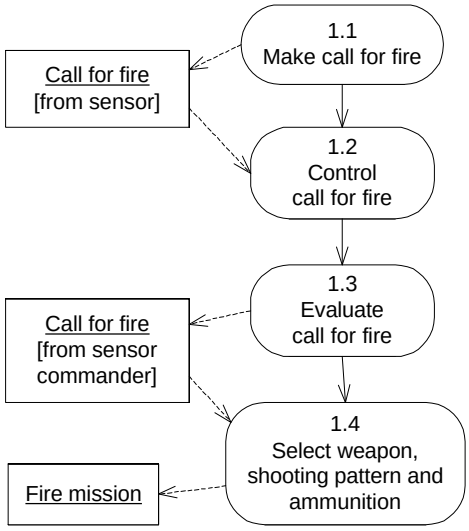

Figure 7. EV2 - Enterprise Process Model

When both EV2 and EV4 are modeled, it is fairly simple to model EV6 (Figure 8). Nevertheless, it is essential to remember the importance of these models. When more details about the activities in EV2 are revealed, these models should be used to evaluate whether the distribution of work makes sense. The user should be heavily involved in the making of these models, but an experienced analyst should suggest changes when it is obvious that the workload is unsuitable. However, it is important to avoid detailed descriptions of each activity. The focus is to get a good understanding of the high-level business processes and to identify the information flow between the different processes.

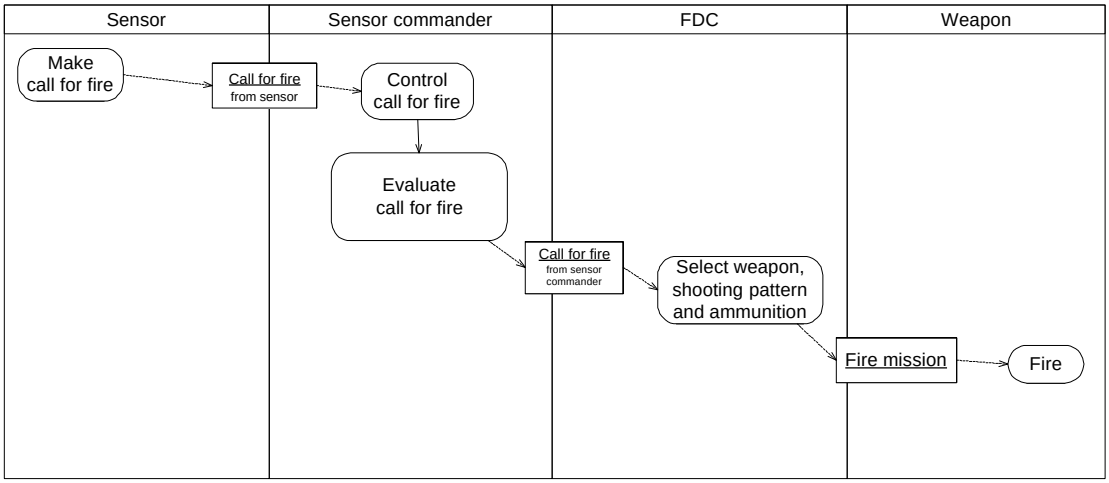

Figure 8. EV6 - Role Activity Model

The next thing to do is to combine the roles already combined with the activities (EV6), with the units of distribution. We can do this using an UML package diagram. Figure 9 demonstrates the distribution model between the sensor commanders and the FDCs. The area of concern is a fire mission, thus the dependency is from the sensor commander to the FDC since the sensor commander triggers further response in a FDC.

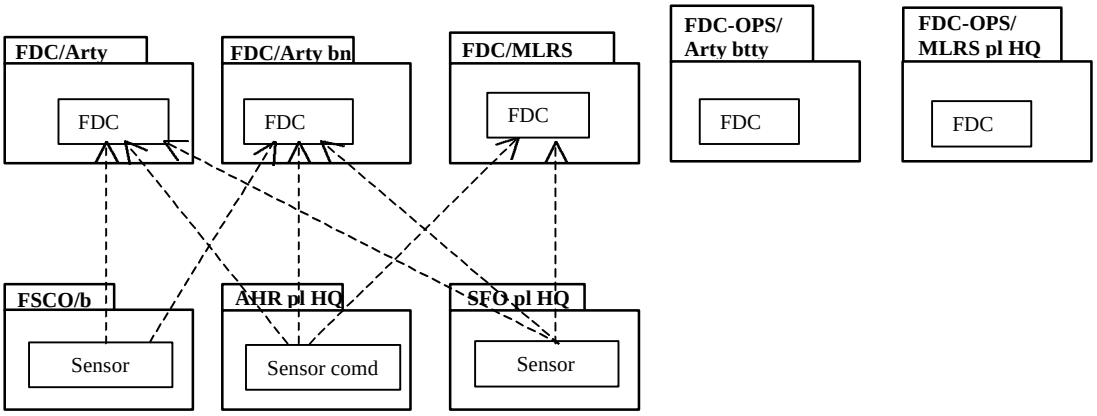

Figure 9. EV7 - Distribution Model (Sensor Commander and FDC)

Similar models should be made for distribution between the other roles in focus.

Based on the previous models and other relevant requirements given by the user, EV9 can be made. There are several ways to model requirements, but it can be useful to start with UML use case diagrams as shown in Figure 10 and Figure 11.

To narrow the example, the focus is on the role “sensor commander”, specifically FDC/bn who receives call for fires from his forward observers.

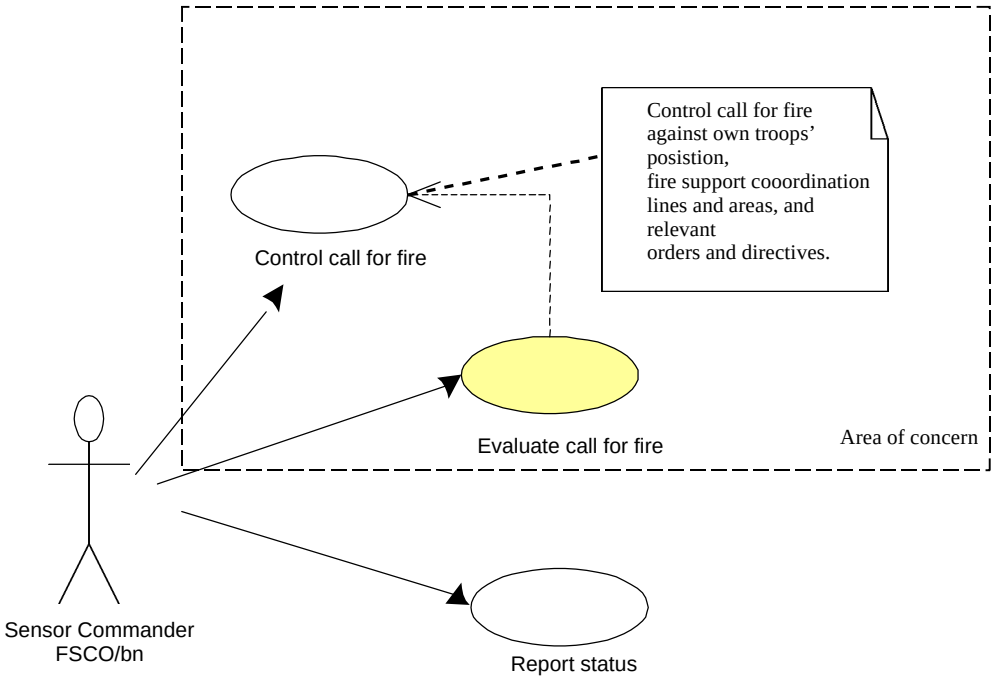

Figure 10. EV9 – Requirements Model

The use case "Evaluate call for fire" is marked because it is elaborated further in another use case diagram, see Figure 11.

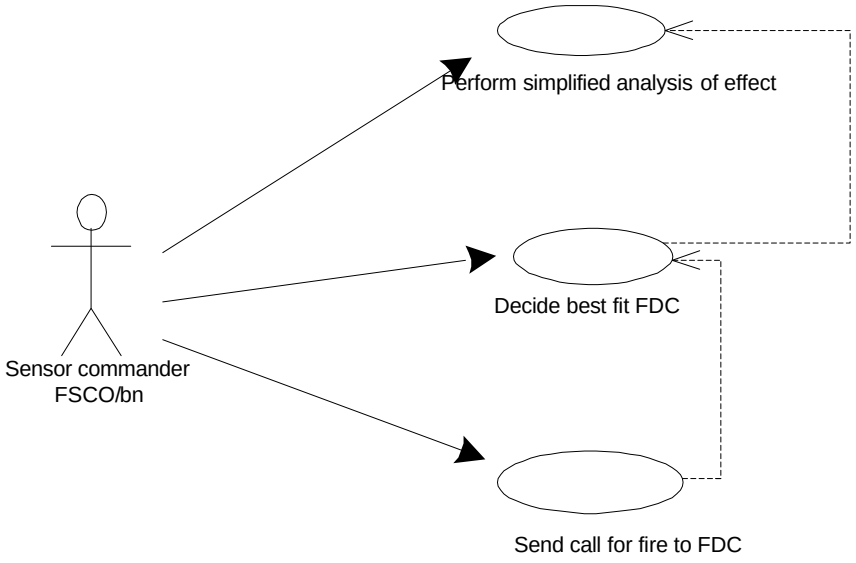

Figure 11. EV9 – Requirements Model: "Evaluate call for fire"

The models above are the essential models in the enterprise view for this specific example. Now we can begin to describe the computational objects and their interactions. The first step is to find reasonable system components that are necessary to meet the system requirements summarized in EV9. These components are displayed in an UML sequence diagram, see Figure 12. Note that

these components will support the sensor commander role in the node FSCO/bn, other system elements that may be in this node are not elaborated in this example.

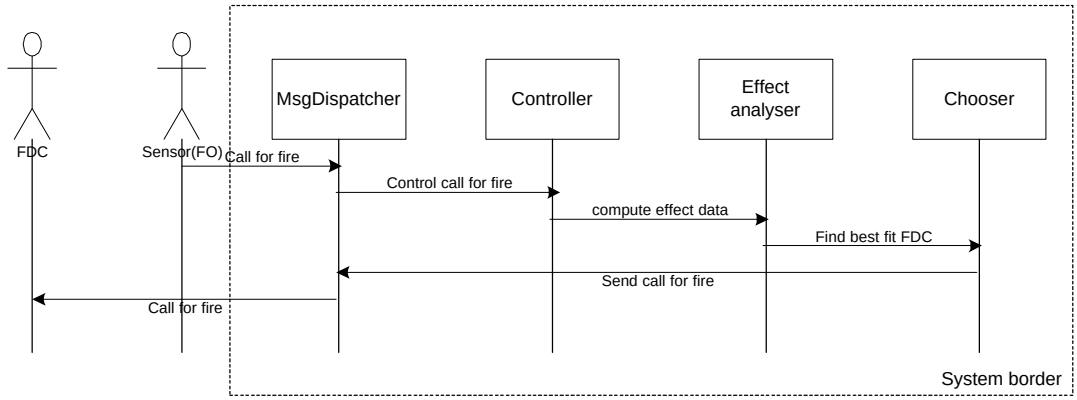

Figure 12. CV1 - Architecture Model: FSCO/bn (automatic mode)

The Computational viewpoint and the Information viewpoint are closely related. The Information viewpoint is based on the information processing and information needs of the components in the Computational viewpoint. Likewise, the Computational viewpoint is based on the concepts set by the description in the Information viewpoint. Hence, iterations between the different descriptions are necessary. Here we choose to focus on the MsgDispatcher and model what we need in order to send a message to another node. Figure 13 shows the classes and methods involved in sending a message.

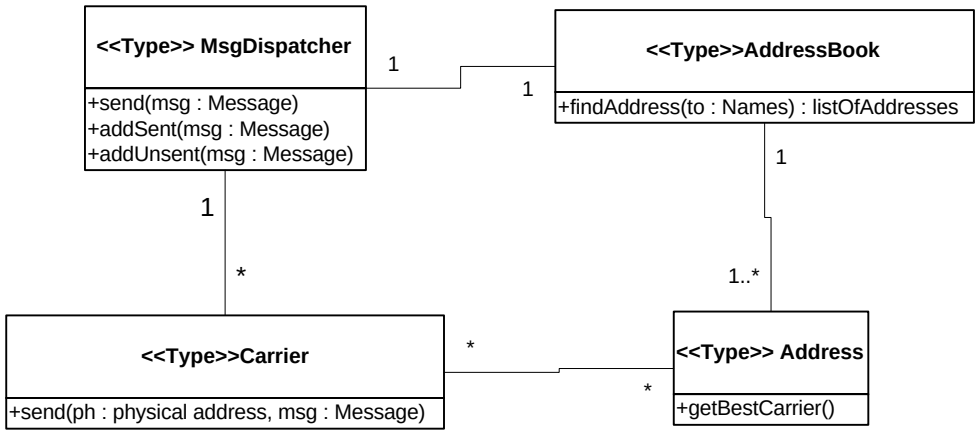

Figure 13. CV3a – Service Model, UML class diagram (send outgoing message)

The next step is to write down the pre- and post-conditions for the send-operation on the MsgDispatcher:

pre: for each msg recipient // may have multicast messages
one carrier must be the preferred one

post: for each msg recipient
if all carriers are down (not available)
add msg to the list of unsent messages
else

add msg to the list of sent messages
and send it on the preferred carrier using the physical
address of the recipient

Pre- and post-conditions add semantics to the operation specification and can be viewed upon as a contract specification between the service provider and the service user (client). When the set of operations along with their pre- and post-conditions are agreed upon on one level of abstraction, each component providing the services can be further refined without affecting the overall system.

It can be wise to use a sequence diagram to make sure that what we have been thinking makes sense. A UML sequence diagram for sending a message is shown in the figure below.

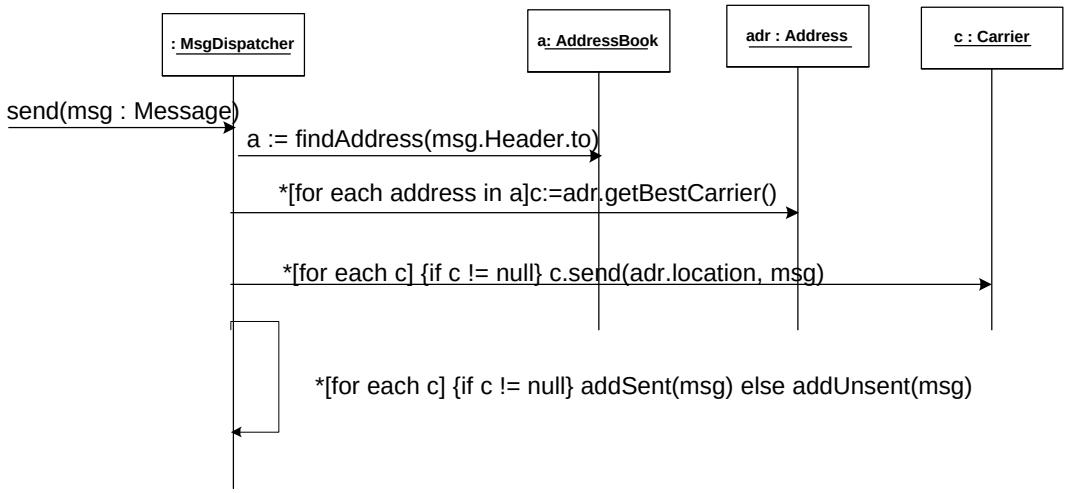

Figure 14. CV3b – Service Model, UML sequence diagram

Parallel with modeling CV3, we model IV2. The result can be viewed in Figure 15.

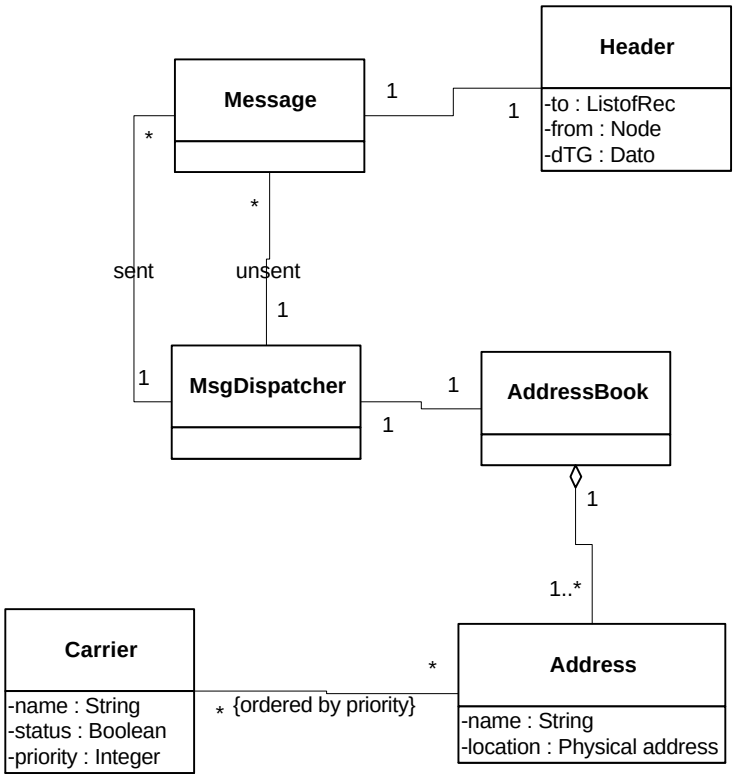

Figure 15. IV2 – Information Model, Invariant Schema

To which level of abstraction one chooses to model depends on both domain and application. Hence, it is important to decide how detailed the models represented in architecture needs to be before detailed design of each individual component can begin.

This example illustrates the use of MACCIS by modelling a small subset of a “call for fire” use case. Although presented sequentially, the work on the example was iterative and different models were developed during the work process, only the final results are presented here.

5. Conclusions and Future Work

Using C4ISR-AF to build the first iteration of the Norwegian Army Tactical Command Control and Information System Architecture revealed certain deficiencies. These shortcomings can be summarized as the lack of an information architecture, distribution architecture, security architecture, role-modeling, and requirements model. We propose to include these aspects in MACCIS as presented here, and we also suggest dividing the architecture into five viewpoints instead of three.

The basic ideas of the MACCIS framework have been discussed with representatives from the C4I domain from USA and UK within the Object Management Group (OMG) special interest group for C4I (C4I DSIG). Through these discussions it was found that work with the same goal as MACCIS is underway both in USA and in the UK. The discussions on architectures and frameworks will continue within this community and will bring valuable input to future versions

of MACCIS. It is also a goal to establish more formal cooperation with the parties from USA and UK.

Acknowledgements. Major Per Trygve Gundersen and Captain Ole Øyvind Stensli from Norwegian Army Material Command and research scientist Audun Jensvoll from Norwegian Defense Research Establishment have provided valuable input to this work.

6. References

[Department of Defense, 1998], Department of Defense. *Joint Technical Architecture, v 2.0*, DoD.

[ISO/IEC JTC1/SC21, 1995], ISO/IEC JTC1/SC21. *Basic reference model of open distributed processing, part 1: Overview*.

[Neple, T., Aagedal, J. Ø., Bjanger, B. W. and Berre, A.-J., 1999], Neple, T., Aagedal, J. Ø., Bjanger, B. W. and Berre, A.-J. *MACCIS - Minimal Architecture for CCIS in the Norwegian Army*, SINTEF Telecom and Informatics, Oslo, pp. 48.

[OMG, 1996], OMG. *CORBAServices: Common Object Services Specification*, Object Management Group.

[Reenskaug, T., et al., 1996] Reenskaug, T., Wold, P. and Lehne, O. A. (1996) *Working with Objects - The OOram Software Engineering Method*, Manning Publications.