2002 Command and Control Research and Technology Symposium

Topic: Network Centric Applications and Space

Paper Title: A System Design Archetype for C4ISR systems of the 21st century

Point of Contact: Darryn Reid

Authors: Darryn Reid, Land Operations Division, Defence Science and Technology Organisation, P.O.Box 1500 Edinburgh South Australia 5111, ph 61-8-8259 7156, fax 61-8-8259 5624, email darryn.reid@dsto.defence.gov.au

Wayne Johnson, Land Operations Division, Defence Science and Technology Organisation, P.O .Box 1500 Edinburgh South Australia 5111, ph 61-8-8259-7141, fax 61-8-8259-5624, email wayne.johnson@dsto.defence.gov.au

A System Design Archetype for C4ISR systems of the 21st century

Darryn Reid and Wayne Johnson

Land Operations Division, Defence Science and Technology Organisation,
P.O.Box 1500 Edinburgh South Australia 5111,
ph 61-8-8259 7156, fax 61-8-8259 5624,

Abstract

In this paper we discuss the difficult issue of how to reliably design and build systems that are fundamentally suited to interoperate with one another. Ideally, all these systems would be constructed using the proposed design principles, but the requirement to interoperate with legacy systems is also recognized. It is not a traditional systems engineering or technology based approach. Rather, a set of key design principles is enumerated. They are, (i) *interoperability and system integration*, (ii) *decoupling of architecture layers*, (iii) *decoupling of system components*. There is also an emphasis placed on a design methodology embodying these design principles, which allows a flexible system design that changes infrequently, and a system implementation that changes constantly.

Together with other invariants, these principles are embodied in a highly abstract and widely applicable system design archetype. Populating this archetype further with application frameworks and specifications of components and their behaviors progressively produces more detailed designs. In particular, the archetype identifies core components for storing and conveying arbitrary information between application components, as well as components for managing the dissemination of information around the wide-area network.

Using this approach, a system concept demonstrator has been developed that embodies a new approach to building C4ISR support systems, in particular with the aim of addressing the challenges of tactical land environment. This concept demonstrator is an aspect of a broader integrated programme in which advanced technology solutions are developed in conjunction with new tactics, training, procedures, and organizational structures, within particular environmental and operational contexts.

1. Introduction and Overview

Efforts invested in developing large-scale Command, Control, Communications, Intelligence, Surveillance and Reconnaissance (C4ISR) Systems [3,5,11,13,16,17,19] have highlighted inadequacies in current design and development approaches. In particular, object-oriented and component-oriented design approaches [6,15,20,22] are incomplete in the sense that they do not in themselves provide a mechanism for achieving the desired levels of flexibility, robustness, and maintainability that C4ISR environments demand. Recent work has focused on categorizing groups of related objects (and their interactions) into 'design patterns' [6,8,9,10,20], to encapsulate and categorize useful design fragments at a higher level of abstraction. However, these patterns are themselves typically without a broader context to relate them directly to the goals of the entire system. That is, most design patterns address relatively small sub-problems frequently encountered within larger designs, but do not directly describe an entire system design.

A central problem with existing design and development practice is that the resulting systems are not able to withstand change. Object-oriented approaches are a firm foundation for system design, but do not automatically capture a number of vital system goals and principles; a methodology is required that decouples parts of the system that change at different rates, ideally leaving an infrastructure that can remain inviolate. Neither object-orientated analysis and design nor design patterns inherently provide a way to achieve this.

The new approach outlined in this paper extends the design pattern concept to addressing entire designs, by relating a highly abstract archetype design directly to enduring system goals and parameters, and then instantiating these patterns with greater detail to produce prototype designs that identify groups of objects and summarize the nature their interactions. The fundamental idea is that archetype design development is a process of localizing the design around fundamental principles of system design as well as important enduring constraints and conditions that should be recognized at the outset. Note that the archetype design, and therefore the entire approach, is not localized around user requirements, rather describing how each application-specific object will be decoupled from other parts of the system.

In other words, the archetype design describes at a very high level of abstraction how the entire system will work. This contrasts sharply with the usual accounts of design patterns, which promote the idea that the skeleton of a system design is obtained by bringing together many different patterns. Specifying some parameters and providing greater detail produces a prototype design, and the detailed system design is simply the prototype pattern fully instantiated with an operational architecture. The final system itself is this detailed design implemented in hardware, software, people, and processes.

This discussion of the new system development approach centers on the new system design archetype. Here we focus not on engineering standards but rather on the semantic understanding of information. A system infrastructure based on a given conceptual schema [2,4,7,16,17,21] should be relatively slow to change if the conceptual schema remains relevant as the organization evolves. The system infrastructure is thus the essential integrator on the ground both at any instant in time and across subsequent versions.

Adoption of standard conceptual schemata also provides management benefits in the form of a clear assignment of responsibility for system integration to the organizations accountable for the development of individual component systems. This approach does provide an effective mechanism for controlling the complexity of an overall system of systems, by reducing the diversity otherwise present if considering a multitude of dissimilar interactions between subsets of component systems.

However, standardizing on one or more common conceptual schemata simply does not represent a total solution to interoperability problems. External systems, such as those of coalition partners, and new systems, like COTS products, are a fact of life. Furthermore, such standards are not immutable and must evolve with the organization they reflect.

Within the system design archetype, we also discuss a model for containing the information, the *information space*, and a mathematical representation of the information. This representation of information assumes the form of logic predicates and subsequently a grammar that allows operations on the information. The archetype also describes the interconnection of information space objects and identifies the need for polices to control the dissemination of information.

Some key principles for system design are outlined in section 2. The most important of these principles are methods for achieving semantic interoperability between components, decoupling of architecture layers, and decoupling of components. Semantic interoperability is not directly concerned with the exchange of data between components, but is rather concerned with ensuring that components exchanging information agree on its meaning.

The enterprise and operational architectures with their user requirements, business processes, and organizational structures are separated from the system architecture. That is, the system architecture can support any enterprise and operational architectures, because the system design is built on invariant system design principles and goals rather than changing organizational roles and user requirements. In addition, the system architecture is also independent of the technologies used to implement it, in the sense that the archetype pattern can be built using any technology base. This decoupling of the architecture layers is important in achieving a design that can outlive changing user requirements.

Section 3 presents a system archetype pattern embracing these principles that is applicable to a broad range of systems (in fact, this archetype has application beyond just the C4ISR domain). This archetype describes a system core built around a set of communicating information space objects, through which individual applications satisfying user requirements exchange information. An intelligent network management layer is capable of interpreting instructions passed from applications through the space objects concerning priority and quality of service for different kinds of information.

These concepts have been used in the development of experimental C4ISR systems. Section 4 outlines some lessons learnt from our laboratory systems synthesis experience, and provides an overview of how our work will now progress.

2. Key Principles for System Development

The overall philosophy of system development, within which this system design exists, identifies an immutable set of principles of system development. A design methodology embodying those principles is proposed, leading to a flexible system design which should change only infrequently, and system implementation in people, procedures, management structures, hardware and software which changes constantly. These principles ultimately tie the system together, not only at any given instant but also as the system evolves over time.

That is, a central thesis of this document is that change in the design parameters is inevitable, and that system design and development methodologies themselves must be able to cope with such change. Inevitably, the way to achieve this goal is to isolate that which remains inviolate in the face of such change. The approach to system development advocated here is multi-layered, with immutable principles at the top, various levels of design detail in decreasing order of permanence in the middle, and rapidly changing implementation in hardware and software at the bottom.

Of the stationary principles upon which a design and development methodology might be founded, the cornerstone is methodologies for dealing with semantic interoperability problems and thus achieving subsystem integration. Other important principles include the independence of system designs from both operational architectures and from the technologies used in their implementations, the separation of subsystems from the core system infrastructure, and attention to the way in which the core infrastructure depends upon underlying transport mechanisms.

Unfortunately, the specification of user requirements is the central focus of much system development work. User requirements are invariably incomplete in all but trivial cases, and are particularly prone to alteration as the organization evolves (perhaps in response to the development of the system itself). Worst still current system engineering design is based on purely on meeting these user requirements, resulting in systems that are effectively hard-wired or stove-piped to a particular snapshot of the user requirements. Not surprisingly then, when the user requirements do change, an event that is at best politely discouraged by systems engineers, it becomes a process of forcing a square-peg fit in a round hole. Such inordinate attention to finding the '*immutable user requirements*' as the basis of concrete system development is fatally flawed, as testified by the partial or complete failure of so many system projects. Thus user requirements remain a vital aspect of system development, but are subjugated to the kinds of fundamental system design principles outlined above. That is, user requirements are constantly evolving at varying rates, and therefore should be isolated from the core system design process.

2.1 Interoperability and System Integration

Key to constructing any large system is the problem of integrating a set of separately developed component subsystems typically displaying a range of interfaces discordant in terms of hardware and operating system platforms, data transport mechanisms, syntactic structure, and semantic structure. Of these issues, it is differences in semantic structure that provides the greatest barrier to achieving interoperability between components, and it is arguably this fundamental issue that is the least understood.

While technological and syntactic differences are relatively concrete, semantic relationships between different components must be considered at an abstract level. The semantic structure of information presented at the interfaces of subsystem components is not often given the attention it demands, with the result that system integration is never fully achieved and the overall system is thus unable to meet expectations.

At the abstract level, the semantic structure of a subsystem interface can be described in terms of a set of relationships between domains of data values, and the process of achieving interoperability between two such components involves interpreting these structures [2,4,7,16,17,21]. Conceptual modeling in database theory and formal ontology are concerned with examining semantic structures independently of any particular choices of message syntax or transport mechanism, and are therefore advocated as fundamental tools in system design and development. Using such methodologies produces highly abstract and formal representations amenable to rigorous analysis, and these will be referred to here as conceptual models or conceptual schemata, regardless of the particular modeling approach adopted.

Identifying that a single common conceptual schema provides a framework for managing complexity, assists the separation of core system infrastructure from component subsystems, and clearly assigns responsibility for integration of application components. However, this does not in practice provide a complete solution for system integration problems, as the difficult problem of actually integrating nonconforming component subsystems still needs to be addressed. This is just the same problem of achieving semantic interoperability, this time between the subsystem and the core system infrastructure which instantiates the common conceptual schema.

Once two conceptual schemata are defined, corresponding to either to two subsystems to be integrated (note that a system core embodying a common model is also a subsystem), they must be integrated to produce a new model that encompasses the relationships between the structures of the two originals. The schema integration problem is the essence of interoperability, and thus forms the focus of much ongoing research [2,4,7,16,17,21]. Broadly speaking, the process involves identifying common data sets, resolving conflicts caused by incompatible relationships, and making decisions about how other structures should be connected. Thus the integrated schema typically contains whole new data sets and relations that do not appear in either of the original schemata.

Given the integrated schema, the behavior of the translation mechanism can be derived automatically. This process amounts to examining the possible sequences of operations that may be performed on each pattern of relations between data values that may be received from one subsystem. The aim is to find a sequence that successfully produces a pattern of relations between data values that may be delivered to the other subsystem.

2.2 Decoupling of architecture layers

An approach that encourages consideration of the whole system is the Australian Defence-Architecture-Framework (DAF). The DAF essentially seeks to enhance the defence capability development for the Defence Information Environment (DIE), through the definition of business processes. The DAF is based on the US C4ISR Architecture Framework [5]. Within the DAF, there are various architecture views, which show different perspectives of the system or system of system (SoS). At an operational level, the essential views are OV-1 *High-level Operational Concept Graphic*, OV-2 *Operational Node Connectivity Description*, OV-3 *Operational Information Exchange Matrix*, and OV-5 *Activity Model*. At a system level the single essential view is, not surprisingly, SV-1 *Systems Interfaces Description*, while for the technical level it is TV-1 *Technical Architecture Profile*. Supporting views SV 8 *System Evolution Description*, SV-9 *System Technology Forecast* and TV-2 *Standards Technology Forecast* are also important.

The DAF represents an attempt to standardise the way in which we think about, portray and communicate about systems and SoS, to give a blue print for the future C4ISR systems and how they interoperate. While it is a step forward, it still encompasses the traditional systems engineering approach of demanding an operational level architecture that drives the systems architecture, which in turn drives the technical architecture. In contrast, our approach seeks to decouple these architecture layers as far as possible, or in other words to architect a loosely coupled system or SoS. The prime driver to create loosely coupled systems is to overcome the brittleness of particular concrete-system designs to changes in user requirements, ie. changes to the operational architecture level. Said another way, we search for systems archetypes[1] and technologies with which to develop and implement them.

Out of these we can construct a particular concrete system to meet the needs of the overall system specified in the operational architecture. Subsequently, as changes occur in operational architecture, we can adapt the system to meet these, without having to build the new system from the ground up.

[1] The word ‘pattern’ is often associated with relatively low-level programming design patterns, as opposed to system-level design-patterns. We use the word *archetype* to indicate a system level pattern, perhaps applied somewhat inexactly.

2.3 Decoupling of system components

It is important that system components be as independent as possible from each other to make it easy to interchange systems components. This is not only applied at an application level, say changing one battle-map application with another, but also to the interaction between the application and the core system infrastructure itself.

There is an analogy between decoupling system-components and decoupling object-components, albeit the system components are at a far higher level. Decoupling object-components comes from the well-known method of Object-Orientated Programming (OOP). In OOP, programmers decouple data-representation and data-access methods. This loose coupling creates the ability to change the data representation within the object, without requiring users of the object to be aware of this change, since the access method remains unchanged. Similarly, we seek to change system components, without having to make changes to other parts of the system to which it happens to be connected.

Interestingly, we have found that it is possible to treat the system infrastructure itself in exactly the same way. That is to say, components of the systems infrastructure itself can also change without having to make changes elsewhere in all the applications. By abstracting out the interaction between applications and the core system infrastructure, changes of implementation within the core system infrastructure do not affect the application. In general, it is loose coupling that is the key to system flexibility.

3. A System Design Archetype

The underlying principles of system development are unchanging, and it is this set of principles that ultimately ties the system together and provides continuity across even radical technological and organizational change. A system infrastructure based on a given conceptual schema should be relatively slow to change if the conceptual schema remains relevant as the organization evolves. The system infrastructure is thus the essential integrator on the ground both at any instant in time and across subsequent versions.

The archetype pattern outlined in Figure 1 defines the overall structure of the system by specifying this core system infrastructure and its relationship with application-specific code. The central components are the ‘information space’ server objects, which provide a buffer capable of accepting structured information arbitrary in content. In the case of our prototype design, this structured information conforms to a logic-style language, although other models could also be used. Clients harboring application-specific functionality communicate anonymously and asynchronously with one another. That is, any client to a space object communicates indirectly with other clients in the absence of knowledge of their location, and two clients do not have to be connected at the same moment to communicate.

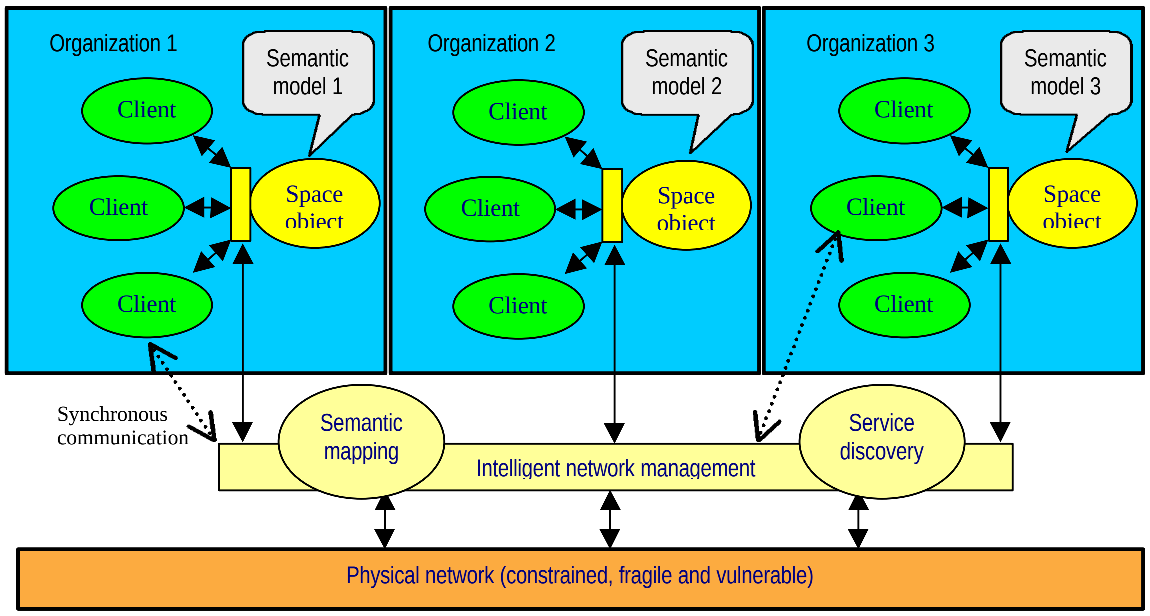

Figure 1: The Information Space Archetype.

The intelligent network management controls the dissemination of information between space objects across a wide-area network featuring constrained and transient resources. As far as the space object is concerned, the network management layer is present in the form of just another client. Clients aware of the business processes of the organization may provide instructions to the network management client by specifying a dissemination policy for information matching given criteria. The network management clients may also provide other clients with information about the current network state, by asserting facts to the space object. The intelligent network management layer provides simultaneous management for asynchronous exchanges between space objects and for synchronous voice and video communications directly between certain types of clients.

Distributed lookup and service discovery is also present; this network of objects provides clients and space objects with the ability to dynamically reconnect to one another upon reconnection, or when first connecting to the rest of the system. Typically, each organizational element having its own local area network will have a lookup service object that communicates periodically with others in the system.

An advanced system design will provide a more stable core infrastructure if it can be independent not only of changing component subsystems, but also of the conceptual schema itself. The system design reported here encompasses this very ideal, and does not even demand that the system represent only one conceptual schema; potentially many conceptual schemata are instantiated dynamically by component subsystems themselves. When connecting to the core infrastructure, subsystem components declare their semantics to the system core, which may then independently generate new code to achieve integration.

3.1 Information Space operations

The prototype design is generated from the archetype by making particular choices concerning the elements identified in the archetype. One such choice is that of a general representation of information for exchanges between space objects and their clients (recall that the intelligent management layer is represented by yet other clients as far as space objects are concerned), and the kinds of operations that the space objects support to facilitate this exchange.
In our current prototype design, client applications and information space servers exchange information in a formal structured language that defines the set of all predicates as the fundamental construct. Each predicate is simply a labeled set of attributes, each of which is an attribute name and corresponding value. Each attribute name occurring within a given predicate is unique. Values are typed, and can in principle belong to any well-defined named domain. One such domain is the set of all predicates, so that a predicate can contain a nested predicate in any of its attributes. Another important domain is simply a standardized set of variable names, which is distinguished from other domains in that values chosen from this domain have particular meaning to information space servers and their clients.

Information space servers can be instructed by clients to perform essentially two types of operations, namely assertions and queries; assertions explicitly alter the state of the server, while queries do not. In principle, assertions and queries could involve arbitrarily complex logical constructions, using either standard two-valued logic with a closed-world assumption or even three-valued logic (‘true’, ‘false’, and ‘unknown’) with an open-world assumption. Currently the implementation only supports the standard two-valued logic, principally because this is more readily supported by currently available underlying storage technologies.

The assertions and queries essentially follow a *tuple-space* style [9,18], with its basic operations of write(…), readIfExists(…), waitToRead(…), takeIfExists(…), and waitToTake(…). Other convenience operations of writeAll, readAllIfExists etc, along with some other register-query/callback mechanisms are provided. This provides a more restricted set of operators than those of a database ilk are used to. However, it provides an easy to use synchronisation and coordination mechanism, where all of the hard work is done within the core system infrastructure of the Info-Space itself, not in the application.

In practice, the attribute names are not passed between clients and information space servers, rather being implied by order. That is, the set of attribute name and value pairs is reduced to a set of values ordered by implicit attribute names; in the implementation framework, a predicate is represented by a labeled list of values. These values can be of any of a number of standard types, including signed and unsigned integer and real number types, Boolean, character, string, raw binary data of fixed or variable length, variable or predicate. Predicates that are not nested, meaning that they do not appear as values within another predicate, are said to be ‘top-level’ predicates.

The current implementation does not permit arbitrarily complex expressions for assertion operations; arguments to assertion operations are conjunctions of one or more predicates. To simplify the traversal of queries, they are always expressed in disjunctive normal form. The language for information exchange thus defines an expression as consisting of a disjunction of clauses, and a clause is a conjunction of terms. Each term can consist either of a predicate or of a negated predicate. A restricted part of this language, prohibiting negations in terms, serves to represent the conjunction of predicates used in an assertion. The grammar commonly used in development and debugging to represent this structured language textually is outlined in Figure 2.

```
<expression> ::= <clause> | <clause> ‘;’ <expression> ;
<clause> ::= <term> | <term> ‘,’ <clause> ;
<term> ::= ‘!’ <predicate> | <predicate> ;
<predicate> ::= <name> ‘(‘ <argument-list> ‘)’ ;
<argument-list> ::= <value> | <value> | <argument-list> ;
<value> ::= <short> | <long> | <ushort> | <ulong> | <float> |  <double>
                <boolean> | <string> | <octet> | <blob> | <boolean>
               | <variable> | <predicate> ;
```

Figure 2: An example of a grammar for information exchange.

Predicates also fall into two classes, namely those defined in the system at run-time by clients, and predefined predicates that have special meaning to information space servers, when appearing at top-level. When appearing as nested predicates, these predefined predicates are treated in the same manner as any client-defined predicate. The predefined predicates provide a rich set of operations for constructing complex queries, and include range restrictions, equality and inequality, type restrictions, and directions for ordering of any results.

The results of a query are also passed using this language. Query operators return a logical value indicating whether or not any results matching the query criterion were found, as well as any results. Results themselves are not returned in full, but rather as a list of variable instantiations. Thus a query issued without any variables will produce a yes or no answer, depending on whether an exact match for the query expression exists in the server. Placing unbound variables in arguments of predicates in the query expression will also produce a yes or no answer, populated with a (possibly empty) list of possible solutions for the variables. These solutions are each expressed using the special equality predicate, with one argument containing the variable name and the other the solution value. A complete solution to the query is a clause containing one such predicate for each variable in the query expression, and a set of results is returned as an expression containing several such clauses.

3.2 Interconnecting Information Spaces

The archetype does not specify a specific dissemination method, only indicating that some kind of mechanism for moving information between space objects in accordance with a constantly shifting policy is required. In recognition of the constraints imposed by the wide-area network, the prototype pattern specifies that an optimistic (sometimes called lazy) replication scheme should be utilized, so that the system aims to control inconsistency across widely separated sites rather than attempting to enforce absolute consistency across the system. Several different optimistic replication schemes [1,12,14] are under investigation for utilization in a tactical land environment.

The intelligent network management layer actually consists of a number of objects. Dissemination engines connected one to each information space receive policy predicates through their local space objects from other clients. These policy predicates are interpreted by dissemination engines as instructions to register interest with the local space in information matching an enclosed query. By registering these queries with the space object, a dissemination engine embodies a list of predicates to be replicated. Dissemination policies also specify or imply a list of other dissemination engines with which matching information is to be coordinated.

Also present in the network management layer are objects that control the communication of information between space objects and, in the case of synchronous voice or video communications, between applications directly. These control objects are essentially responsible for assessing the network state and interpreting the priorities in dissemination policy predicates with respect to this assessment. The control objects provide a single interface for both the asynchronous data delivered by the dissemination engines and the synchronous data passed directly between some kinds of clients. Like asynchronous exchanges between space objects, synchronous communications between clients are subject to priority controlled by dissemination policy predicates.

Finally, note that the intelligent network management layer controls possibly several separate underlying physical networks (for example, separate voice and data networks). That is, the dissemination managers acting on behalf of information space objects as well as applications that utilize synchronous communications remain unaware that the intelligent network management layer actually has several different networks for transporting data.

4. Lessons learnt

The price for flexibility is complexity. Essentially the load of complexity is within the implementation of the Info-Space itself. In practice, we have found it more difficult than expected to build information space servers to have the level of functionality and ease of use that is necessary to build system-concept-demonstrator applications. Adapting existing database products and other storage technologies is not in principle difficult, but careful design and considerable testing is needed to relate the capabilities of mature products to the new model to produce efficient information space servers. Another challenge faced is that developers are often wedded strongly to their favourite technology, programming-style, and support tools. We have found that the price of education for developers, and that associated with a reduction in efficiency due to the lack of mature development tools to support them, to be significant, to say the least. However, because we are compelled by our research and experimentation programmes to develop system –concept –demonstrators quickly, we have to make a number of strategic decisions to lower our overheads and increase our productivity. We have divided our 'spaces' effort into two parts. Firstly, a research and development stream will continue our system concept development research. The second stream is a commercial-off-the-shelf (COTS) based effort to accelerate the development and experimentation with our system-concept-demonstrators. Our decision to use these COTS products is largely governed by our need to build demonstrators in our laboratory more quickly. We reason it is more productive to go with an 80% solution, than for instance to try and build our own industrial strength space. An offshoot of the second part of this will be to possibly influence the direction of open-standards products.

Therefore, we have chosen to use:-

- GigaSpace:
  - The first commerical implementation of the open architecture JavaSpace specification [9], the GigaSpace. It has the necessary extensions to the JavaSpace specification to make it more usable; WriteAll, ReadAllIfExists, etc.
- Java [8,9]
  - OS/hardware independence, language based systems integration within the SCD
- Jini [9]
  - Service Discovery
  - Partial self administrating network
- Web based delivery, where possible
  - ease of system configuration
  - ease of deployment

Our future work will thus focus on using COTS technology for development of experimental C4ISR systems for synthetic environment experimentation and demonstration. This work will be informed by ongoing development of the system concepts outlined here, but may not directly represent the most advanced of those concepts immediately.

5. Conclusion

A set of key design principles has been enumerated, within a new overall design methodology, which allows a varying system implementation by localizing systems around a constant system design archetype. The design principles are, (i) *interoperability and system integration*, (ii) *decoupling of architecture layers*, (iii) *decoupling of system components*. These design principles are embodied in a system design archetype, which describes the entire system at the highest level.

In this system archetype, the most important concept is that of the information space objects, which provide applications with a mechanism for exchanging information asynchronously and anonymously. The archetype also defines an intelligent network management layer that assumes responsibility for implementing some appropriate replication scheme within the context of a potentially constrained wide-area network. Certain application types have authorization for controlling the behavior of the network management layer, by setting dissemination policy that controls both what information is coordinated across space objects and the priority that should be given to its transfer.

The adoption of standard conceptual schemata for a large information system is an important mechanism for controlling complexity and assigning responsibility for subsystem integration, but does not represent a panacea. A model for the representation of structured information with arbitrary meaning has been given together with a grammar for expressing and manipulating this information is proposed, as well as some ideas on the policy control of the dissemination of information between different information space objects.

Taken as a whole, this represents an approach to synthesizing systems that is strongly decoupled from both the particular user requirements and current technologies, at any snapshot in time. This will allow such systems to be adapted to the fast moving winds of change that occur as technologies evolve quickly and the fundamental mission of the system is changed is response to sudden change in political and strategic circumstances.

6. References.

[1] Bestavros, A., “Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic, and Service Time in Distributed Information Systems”, Proceedings of ICDE ’96, New Orleans, Louisiana, 1996.

[2] Yang, B., and Garcia-Molina, H., “Comparing Hybrid Peer-to-Peer systems”, Proceedings of the 27th VLDB Conference, Rome, Italy, 2001.

[3] Alberts, D., Garstka, J.J., Stein, F.P., “Network Centric Warfare: Developing and Leveraging Information Superiority”, CCRP Publication Series, February 2000.

[4] Batini C., Lenzerini M., Navathe S.B., “A comparative Analysis of Methodologies for Database Schema Integration”. ACM Computing Surveys Vol 18 No 4, 1986.

[5] The C4ISR Architectural Working Group (AWG), “C4ISR Architectural Framework Version 2.0”, December 1997.

[6] Coad, P., North, D., Mayfield, M., “Object Models: Strategies, Patterns, & Applications”, Prentice-Hall, 1995.

[7] Colomb R.M., Orlowska M.E., “Interoperability in Information Systems”, Information Systems Journal Vol 5, pp37-50, 1994.

[8] Cooper, J. W., “Java Design Patterns: a Tutorial”, Addison-Wesley, Reading, Massachusetts, 2000.

[9] Freeman, E., Hupfer, S., and Arnold, K., “JavaSpaces: Principles, Patterns, and Practice”, Addison-Wesley, Reading Massachusetts, 1999.

[10] Gamma, E., Helm, R., Johnson, R., Vlissides, J., “Design Patterns: Elements of reusable object-oriented software” , Addison-Wesley, 1995.

[11] Gossink, D., Horsfall, J., Pattison, T., Zhang, L., and Scholz, J., The Application of the C4ISR Architectural Framework to the Maritime Tactical Environment (U) – DSTO-CR-0118, DSTO, May 99.

[12] Heidemann, J., Goel, A., and Popek, G., “Defining and Measuring Conflicts in Optimistic Replication”, Technical report UCLA-CSD-950033, University of California, Los Angeles, 1995.

[13] Krygiel, A.J., “Behind the Wizard’s Curtain: an Integration Environment for a System of Systems”, CCRP Publication Series, July 1999.

[14] On, G., Schmitt, J., Liepert, M., Steinmetz, R., “Replication with QoS Support for a Distributed Multimedia System”, Proceedings of the 27th EUROMICRO Conference, Warsaw Poland, 2001.

[15] Page-Jones, M., *Fundamentals of object-oriented design in UML*, New York : Dorset House Pub.; Reading, Mass.: Addison-Wesley, 2000.

[16] Reid, D. J., “Interface Components for Coalition Interoperability”, Proceedings of the 5th International Command and Control Research and Technology Symposium, Canberra, Australia, 2000.

[17] Reid D.J., Davies, M., “Towards a Gateway to Interconnect Simulations and Operational C3I Systems”, Proc 3 rd SimTecT98, pp 27-32, 1998.

[18] Rowstron, A., “Using asynchronous Tuple Space access primitives (BONITA primitives) for process coordination”, Coordination Languages and Models, ed D Garlan and D Le Metayer, pp426-429, Springer-Verlag, Lecture Notes in Computer Science 1282, 1997.

[19] Seymour, R., Kirby, B., Krieg J., Reid, D., and Uniwisse, M., “Achieving Interoperability through an Information Management Architecture”, Proceedings of 5th International Command and Control Research and Technology Symposium, Canberra, Australia, 2000.

[20] Sparks, S., Benner, K., and Faris, C., "Managing Object-Oriented Framework Reuse", IEEE Computer, pp 52 - 61, September 1996.

[21] Takizawa, M., Hasegawa, M., Deen, S., “Interoperability of Distributed Information Systems”, Proceedings 1st International Conference on Interoperability in Multidatabase Systems, pp239-242, 1991.

[22] Wang, G., Ungar, L., and Klawitter, D., "Component Assembly for OO Distributed Systems", IEEE Computer, pp 71 - 78, July 1999.