

"Generative Gateway Toolkit for Heterogeneous C3I Systems "

Greg Hupf, Principle Engineer, Command and Control Technologies

Rodney Davis, Chief Technology Officer, Command and Control Technologies

1425 Chaffee Drive, Suite 1

Titusville, FL 32780

321-264-1193, hupfg@cctcorp.com, davisrd@cctcorp.com

Abstract. CCT is conducting Air Force sponsored research to provide a model driven composable architecture for C3I gateway systems that will provide a generic platform independent software toolkit for creation of communication gateways. This new approach will simplify interoperability of new and legacy systems by leveraging transport and semantic exchange standardization. Use of product line architecture methods will be employed to ensure large-scale system reuse, enabling component-based gateways to be generated from high-level specifications. Central to the scope of the gateway architectural approach is accommodation for variation and evolutionary extension of communications and exchange services.

This dual focus on architectural strategy and standards is critical to dealing with the historical problems of stove piped architectures and N(N-1) interfaces between systems. The benefits of this approach include simplification of creation of domain specific gateways, and improved interoperability between systems.

Introduction: Identification and Significance of the Innovation

The proposed innovations are as follows:

- 1) Use generative programming techniques to automatically generate gateway software by modeling system families in a given domain. Generative programming is developed in seven stages: Domain Modeling, Common Architecture Development, Identifying Domain-Specific Notations, Specifying Configuration Knowledge, Implementing Implementation Components, Implementing Domain-Specific Notations, and Implementing Configuration Knowledge Using Generators.
- 2) Develop an Enhanced Gateway Reference Architecture that will provide simple modification, certification, and reuse of gateway functional modules based on application of product line strategy for large-scale reuse.
- 3) Leverage metadata standards and supporting middleware for transport, format, and semantics exchange between systems.

These innovations are significant because they will:

- 1) Enable legacy and future systems to exchange common data and semantic information

- 2) Prolong the life of existing systems by providing a gateway architecture that enables interoperability and transition to new technologies without requiring a major redesign or system replacement
- 3) Create more standardized architectures for gateways
- 4) Support an evolutionary architecture approach
- 5) Simplify the certification, testing, and fielding of interoperability solutions
- 6) Provide a standardized reference architecture that will allow the gateways to be built from reusable gateway building block components
- 7) Support more efficient testing and re-use of the gateway components in development
- 8) Facilitate the development, evolution, and reuse of Gateway systems software for multiple gateway families
- 9) Reduce operations and support costs of gateway systems
- 10) Foster process, system, and product improvements through comprehensive asset (documentation, processes, requirements, etc.) reuse

Background

Product Line Reference Architecture

More and more software engineering organizations and communities have come to the realization that in developing single systems they are, in essence, repeatedly developing the same capabilities (i.e. “reinventing the wheel”). This is the case with the current gateway strategy.

For instance, gateway systems typically process commands and/or data, provide a human interface, and archive data. Every time a new gateway is developed, because the existing systems are inadequate for the new purpose (e.g. not interoperable, flexible, scalable, portable, sustainable, etc.), the solutions for these capabilities are reinvented.

The problem is that the existing systems were not engineered with large-scale reuse in mind. It is universally acknowledged that there are significant benefits of achieving software reuse. Consequently, the DOD must develop the infrastructure for the strategic reuse of gateway software. How to accomplish this objective, however, has always been elusive.

Software engineering is an immature discipline. Indeed this has garnered the attention of professionals in government, industry and academic communities worldwide for many years. Yet for all of the attention, innovative technologies and tools, none has proven to be the silver bullet. In 1987 Frederick Brooks postulated, “the very nature of software makes it unlikely that there will ever be an invention that will do for software productivity, reliability, and simplicity what electronics, transistors, and large-scale integration did for computer hardware. We cannot expect ever to see two fold gains every two years”.

In light of these challenges, it’s important to embrace the over 20 year old concept of developing for system families and mature methodologies that hold hope for answering many of the problems currently faced in owning multiple systems that are functionally

very similar. The use of a software product line for gateway systems would allow for the large-scale strategic reuse of software assets that is desired.

This reuse, however, is only part of the story. The true benefits result from the creation of a common gateway system platform. This platform would provide for the consolidation of infrastructure, and allow interoperability between systems, as well as the reengineering of key business processes.

A gateway systems platform would in effect be a collection of the common elements, especially the underlying core technology, implemented across all gateway systems. This would manifest itself in system architecture capable of supporting the common components of each gateway instance within a common framework.

Once developed, these common components would form the basis of each gateway system. The application specific capabilities for each gateway would then be added to complete the full set of functionality. In this way, the pitfall of trying to design a system that is all things to all people would be avoided. The acknowledged trade is that the gateway user does not get to exactly specify a whole new system, but quickly gets most of the capability with high quality and reliability. Only those attributes that are common to all gateways in the domain would be incorporated into the common assets (toolkit). The resulting systems would then be aligned with the goal of meeting the challenges currently faced.

Software product line development allows the reuse of basic family requirements, models and analysis, architecture, design, code, test cases and procedures, and documentation. The process of producing a new system, called Application Engineering, becomes more of an integration effort than a development effort with only system unique requirements being newly implemented. The domain model, reference (generic) architecture and documentation are used as the basis for adapting the product family assets to produce the new system. The benefits include the following:

- New systems can be fielded much more quickly than the typical 3-to-5 year development life cycles experienced with current systems.
- New systems can be developed at a much lower cost and with fewer resources.
- New system developments will entail much less risk as the needed cost, schedule, and resources will be considerably more predictable.
- New system developers will be able to focus on the system specific problems at hand instead of re-implementing the basic functionality previously implemented for other systems.
- Personnel can move seamlessly from project to project because the architecture, components, and development processes will be familiar to them.
- O&M personnel will be familiar with all systems in the product line family allowing them to seamlessly support any.
- New systems will be much more reliable as defects will have been identified and fixed in their ancestors.
- New systems will have come with an established pedigree allowing for the elimination or minimizing of design certification activities.

- Processes, procedures and data can be shared across all of the systems of the product family.
- Better training material and documentation, which can be shared.
- Sustaining engineering costs can be shared across all projects.
- Logistics costs can be shared because of the common architecture.
- Upgrades, enhancements and improvements to the common assets can be realized across all systems of the product line.
- System specific applications and products with applicability to the product family can be added to the common set of assets for other systems to reuse.
- Configuration management, requirements management and CASE tools can be shared across projects.
- Money and resources spent on the development of checkout systems can now be diverted to more strategic goals
- The development of an applications framework to greatly reduce applications development costs now becomes worthwhile.

Model Driven Architecture

Achieving coherent reuse depends on a support of a model based gateway lifecycle that takes advantage of asset reuse and process automation as shown in Figure 1.

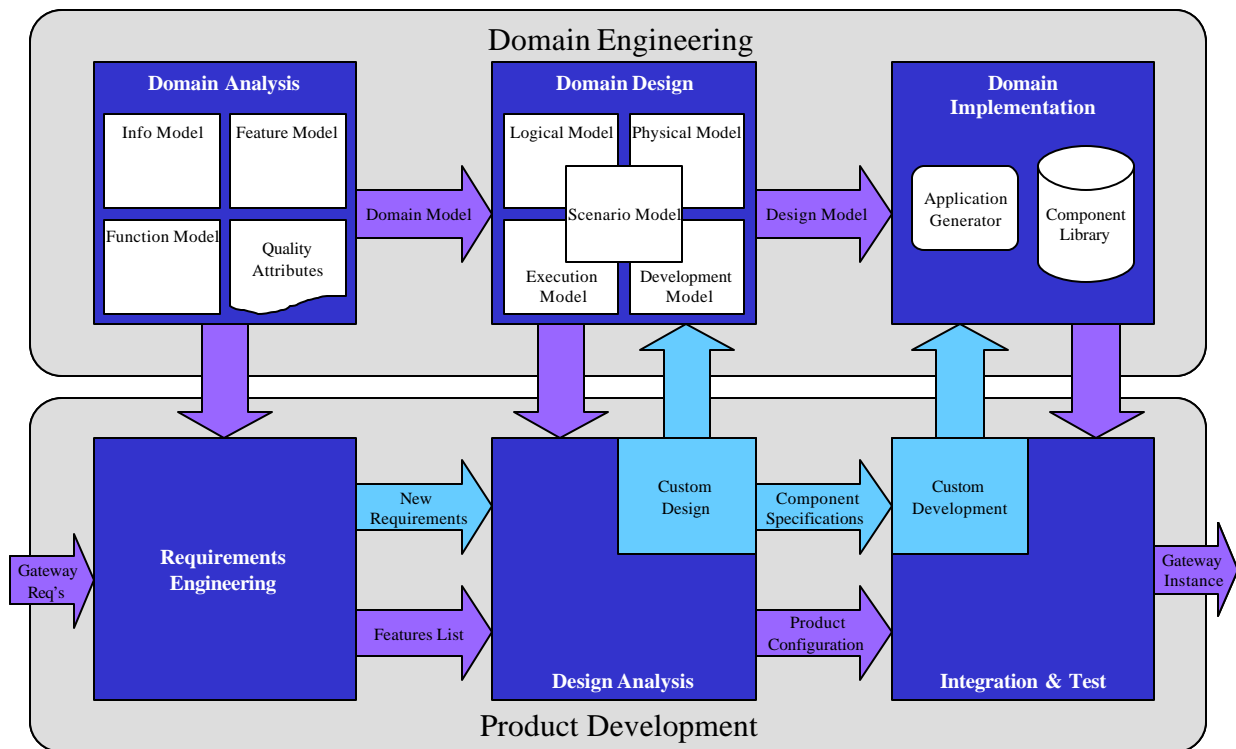


Figure 1, Model based Gateway Development

Domain Engineering

Domain engineering is a process for creating a competence in application engineering for a family of similar systems. Domain engineering covers all the activities for building software core assets. These activities include identifying one or more gateway domains, capturing the variation within a domain (domain analysis), constructing an adaptable design (domain design), and defining the mechanisms for translating gateway requirements into systems created from reusable components (domain implementation). The products (or software assets that make up the toolkit) of these activities are domain model(s), design model(s), domain-specific languages, code generators, and code components.

Product (Gateway System) Development

Domain engineering and product development are complementary, interacting, parallel processes that comprise a model-based, reuse-oriented software production process for the gateway product line. The product development process develops products from assets created by a domain engineering process.

The focus of product development is a single gateway instance whereas the focus of domain engineering is on all related systems within a target domain. Typical product development activities include using:

- A domain model to identify gateway requirements,
- A generic reference design (design model) to specify a product configuration
- A partitioning strategy and coordination model to guide custom development
- Application generators and software components to produce application code

Domain engineering is the embodiment of the principle of design-for-reuse whereas product development is the embodiment of the principle of design-with-reuse.

Asset Management

There are many barriers to software asset reuse. There is a natural resistance to change, whether organizational or process. People will continue to stay focused on day-to-day tasks without proper motivation. Some attitudes that resist implementation of software include fear of metrics, apathy, peer rivalry, and fear of failure.

Project managers are typically tasked to ensure the success in development of particular systems or applications. Managers with this type of incentive will resist committing resources to find common solutions, processes, or standards.

Legacy systems have led organizations into solutions lacking in common standards or processes. Choices are made without considering the impact on compatibility or interoperability.

It is critical to overcome these barriers. The gateway toolkit approach must have executive command support, training, and incentives for project managers. An incremental approach may be best for introducing these concepts. It is also important to develop a framework by which software assets can be identified and reused efficiently.

Data and Semantics Exchange

Future gateway systems will need to be able to operate adaptively, with more autonomy than systems do today. In the future, concepts of ubiquitous communications infrastructure, dynamic service discovery, and mobile agents will enable loosely coupled systems to collaborate to establish objectives and achieve broad mission goals. A critical stepping-stone in realizing this future is establishing a vocabulary and mechanisms for exchange of a wide range of configuration, control, and instrumentation information. This potential stepping-stone to the future is a very real challenge for our war fighter systems infrastructure today.

C3I and C4I design is performed today through the use of a number of diverse tools and techniques. Interface design for systems is manual and time consuming. Data design is performed multiple times by multiple contractors during the systems lifecycle, well before the systems are deployed for mission operations.

Similarly, mission operations require the exchange of information across multiple heterogeneous missions using a common communications infrastructure. Information must be exchanged among all of the operational phases, systems and organizations. This is made difficult and costly because there is no standard method for exchanging this data definition information. The lack of standardization currently requires custom ingestion of the semantic information. This customization is inherently error-prone, resulting in the need to compare and revalidate at each step in the lifecycle.

A typical example of this process is between operating organization. Each operational organization defines data in a format that is unique to their C3I architecture. This creates the need for database translation, increased testing, software customization, and increases probability of error. Standardization of the data definition format will streamline the process allowing dissimilar systems to communicate without the need for the development of mission specific database import/export software.

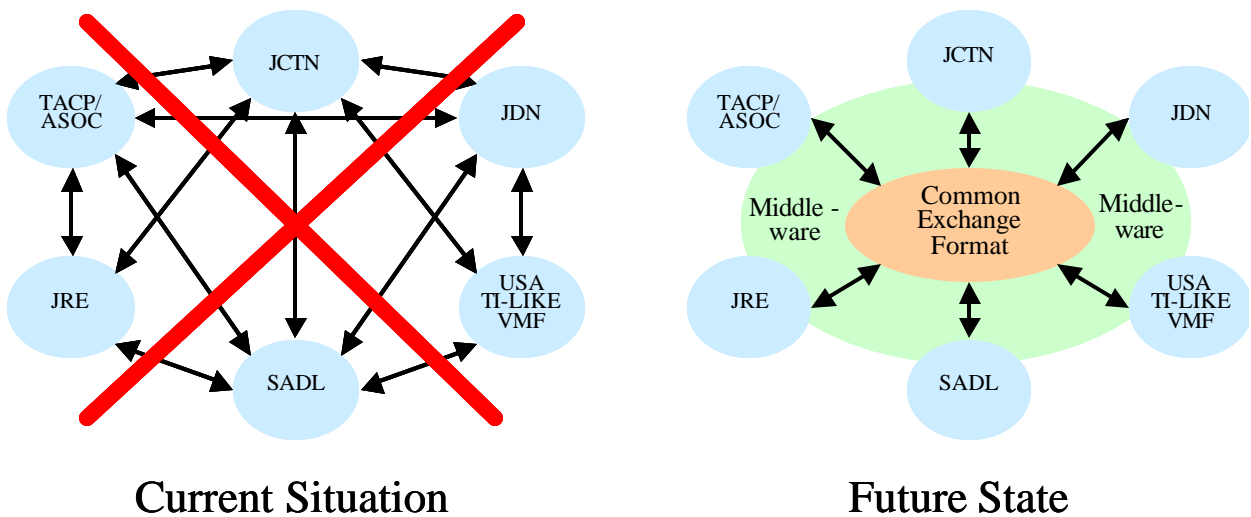


Figure 2 Exchange Standards Outcome

Ideally, a gateway system operator should be able to transition from one system to another by simply moving an already existing database that is compliant with the standard specification. As illustrated in Figure 2, there currently is a need for $N(N-1)$ number of interfaces between producers and consumers of data description information. The proposed approach seeks to produce a more manageable strategy that requires $2N$ potential interfaces. This means a smaller number of interfaces to develop and maintain, and increased flexibility and utility in between the producers and consumers.

This scenario can play out in any number of possible current and future gateway systems where heterogeneous systems need to interact. Hence, a fundamental requirement for the interoperability of current and future systems is to establish a common language for describing how systems communicate. An information model is needed that facilitates common understanding of structure and content for communications. Standards, such as the JTA, offer a part of the potential solution by identifying common mechanisms for representing form and content. However, standards alone will not be sufficient to solve the problem. Software required to accommodate these new and complex standards will be expensive, especially if the standards are to be adopted by the myriad of legacy systems owned and operated by the government. Adopters of these standards could be faced with spending significant dollars per system, which will aggregate to many millions of dollars across the defense operations enterprise. Robust, adaptive software tools and/or services are needed to reduce the cost and complexity of adopting the new standards.

Generative Programming

For many years, building gateway systems from the ground up has been the norm by government and industry. New systems are usually custom-developed to satisfy a pending need on a particular project or program because existing systems are inadequate to meet, or to costly adapt to, the new requirements and no commercial alternatives are available. These custom stove piped gateway systems work well for their intended purpose but require very significant cost and resource investments over their respective life cycles while providing for little software reuse. Additionally, because each system is unique, the costs and resources are often difficult to share, which appreciably compounds their collective total cost of ownership.

It is no longer economically feasible to handcraft and maintain one-of-a-kind gateway systems. Software reuse has always been the logical solution to avoiding these costs. However, until more recently, planned software reuse has been difficult to attain on a large scale. Studies indicate that a major reason why this has been so is chiefly due to architectural mismatch. That is the software structures of components from different sources often do not match, making them very difficult and costly to integrate. It is now recognized that in order to facilitate planned large-scale software reuse, a common architecture must be used and, therefore, *families* of gateway systems must be developed in lieu of single systems.

By developing software architectures designed for families of systems, organizations worldwide are reaping the benefits of large-scale software reuse. These families of systems, called software product lines by some, are generally designed to operate within specific domains, implementing general capability through elementary, generic,

components. Customizations are implemented by way of pre-planned support for various options, alternatives, and extensions.

There have been a number of successes in the last ten years using this approach. *Product Line* approaches have been used in an attempt to gain the following benefits.

- New systems can be fielded much more quickly, at a much lower cost and with fewer resources effectively amortizing development costs across all systems.
- New system developments will entail much less risk as the needed cost, schedule, and resources will be much more predictable.
- New system developers will be able to focus on the system specific problems at hand instead of re-implementing the basic functionality previously implemented for other systems.
- New systems will be much more reliable as defects will have been identified and fixed in their predecessors.
- New systems will have come with an established pedigree allowing for the elimination or minimizing of design certification activities.
- Processes, procedures and data can be shared across all of the systems of the product family.
- Support personnel can move seamlessly from project to project because the architecture, components, and development processes will be familiar to them across projects.
- Upgrades, enhancements and improvements to the common assets (software, documentation, and so on) can be realized across all systems of the family.
- System specific applications and products with applicability to the other systems in the family can be added to the common set of assets for reuse.
- Configuration management, requirements management and software engineering tools can be shared across projects.
- Money and resources spent on the development of gateway systems can now be diverted to more strategic goals

However, in producing new systems, it is difficult to specify what to build, correctly integrate components, and actually build the system. Trying to manage these complexities by hand is even trickier. Generative programming provides all of the benefits of engineering for families of software systems but goes a step further in an attempt to eliminate the manual build process.

The Concept

Gateway Reference Architecture

The gateway reference architecture description will be of several architecture views that show the overall structure, and provide a commentary describing rationale, architecture style, architectural constraints and alternative architectures.

As with most communication systems, gateways are structured using a layered architecture. Layered architectures provide portability layers for software systems like those of the enhanced gateway that must run on different operating systems and hardware platforms, providing a common abstraction for communications. The layers are used to hide communication and database protocols, as well as operating system, user interface or other COTS toolkit implementation specifics.

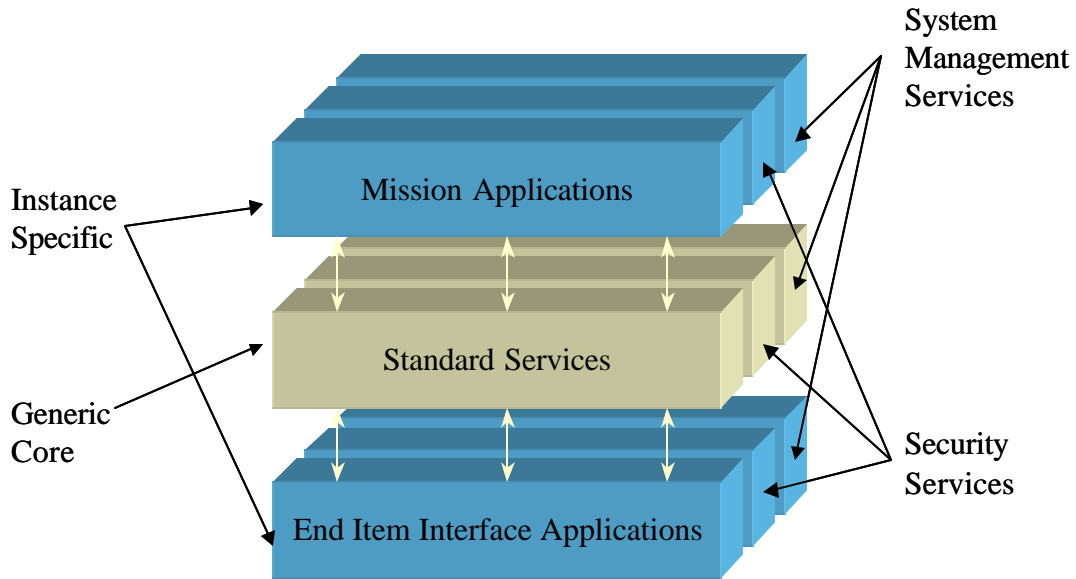


Figure 3 Preliminary Layered Architecture Overview Diagram

The layered architectural style consists of sets of components that implement distinguishable broad categories of functionality. Each of these sets is organized into a layer, isolating or encapsulating unique functionality in a single area. Elements of each layer can, in the strictly layered style, only interact with elements in their layer or an adjacent layer. The layered architecture exists to provide cumulative levels of abstraction on top of some base functionality. These layers exist to hide implementation details and provide for modifiability at varying levels of abstraction [Klein 99].

Figure 4 decomposes each layer into its individual components.

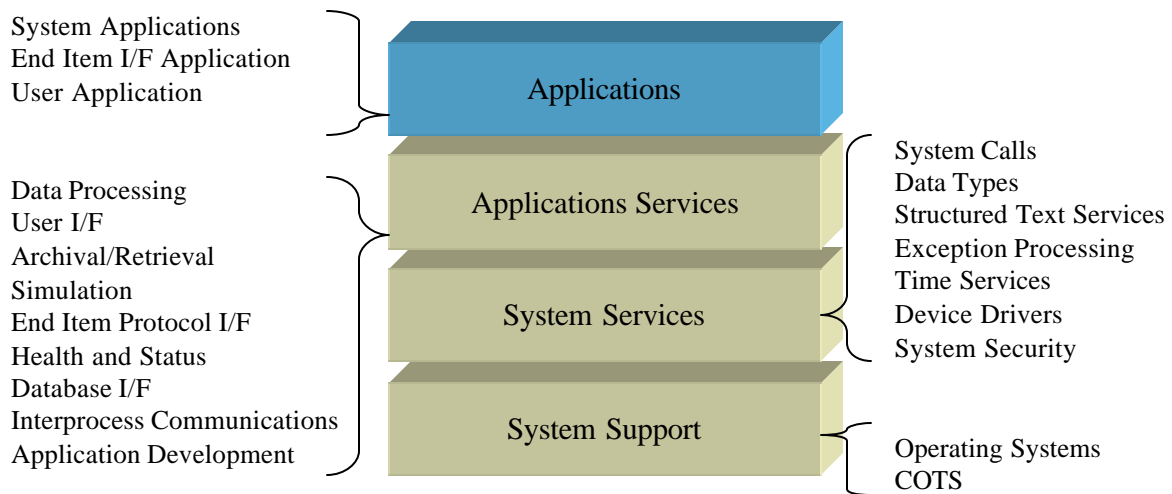


Figure 4, Preliminary Layer Components

Data and Semantic Exchange Approach

The proposed concept is to introduce technology that will ease the adoption of a common data definition and exchange standard across a heterogeneous space operations domain. The solution requires identification and development of an interoperable set of domain-

specific services and middleware that can help the current legacy infrastructure adapt and evolve towards a generic control and monitor capability.

A domain analysis will be performed to identify data definition exchange features that are common across all systems and those that are different for specific systems. The common capabilities, such as accessing and validating data descriptors, will define the basic features of the exchange services. The differences such as system-specific configuration parameters, legacy software interfaces, custom data structures and file formats, will be handled by incorporating variation points into the software architecture design. These variation points will allow for a controlled customization of the exchange services package for a particular application instance.

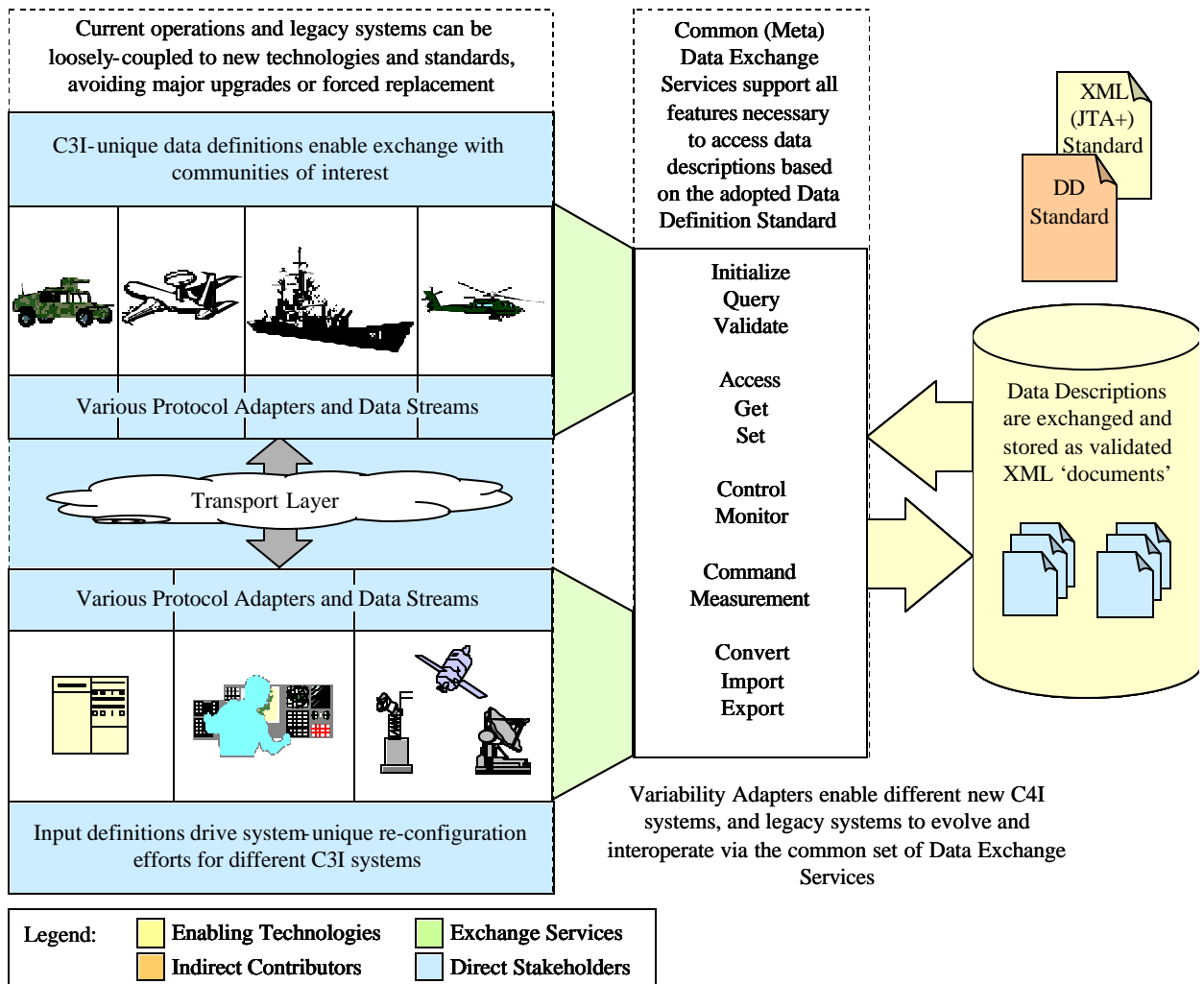


Figure 5 General Exchange Concept

The approach, illustrated in Figure 5 above, seeks to prove the viability of using a standard communications exchange and data definition language to enable heterogeneous systems to interoperate on a platform independent basis. The advanced data description

exchange services will provide a reusable framework of loosely coupled software components that can be configured in different ways to produce solutions optimized for a particular system.

The incorporation of generative programming techniques will provide a means for future evolution so that systems can continue to adapt and maintain compliance with minimal software development effort. The program generation for a particular system instance is facilitated by a standardized procedure, or checklist of decisions, for specifying the options and variations that will cause an optimized solution to be generated. The focus of the optimization will be to maximize interoperability within the domain in the most cost effective manner. For many legacy systems, the solution that requires the least amount of change to the existing infrastructure and caused the least operational impact is the most cost effective.

Generative Approach

Generative programming is an emerging software engineering paradigm based on modeling software system families such that, given a particular requirements specification, a highly customized and optimized intermediate or end-product can be automatically manufactured on demand from elementary, reusable implementation components by means of configuration knowledge. Generative programming extends the concept to design system families by providing the capability to specify at a high-level what to build, and through the use of configuration knowledge and generators to actually build the system. We propose to determine the feasibility of using such techniques for the automatic generation of gateway systems.

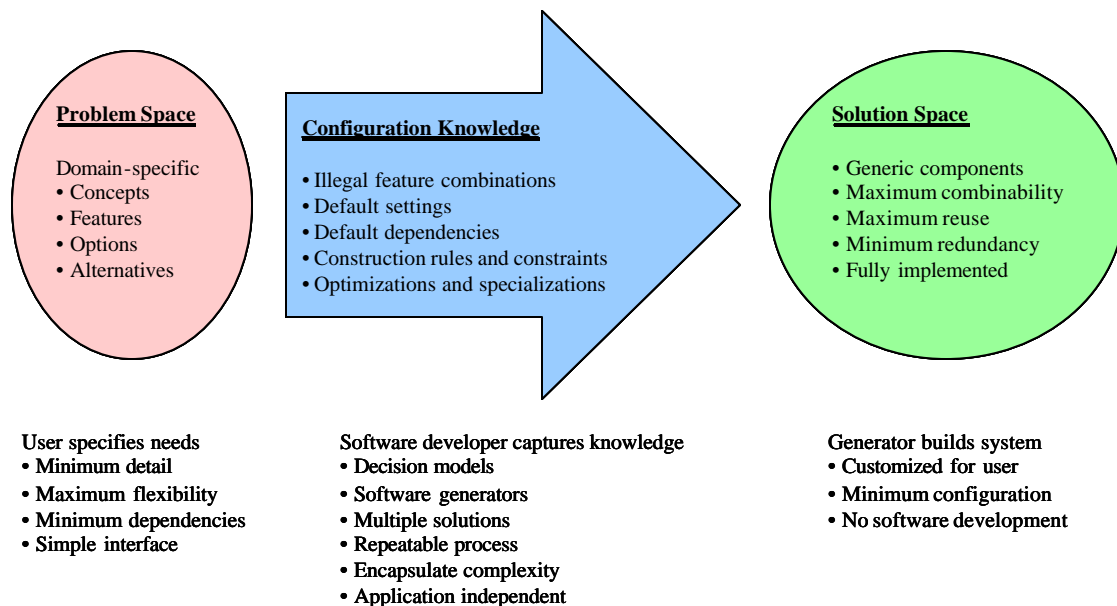


Figure 6 Elements of Gateway Program Generation

Directly Related Outside Work

Directly related published work includes:

- W3C Recommendation - Extensible Markup Language (XML) 1.0 (Second Edition)", www.w3.org/TR/REC-xml, Oct. 2000
- CEN/ISSS Electronic Commerce Workshop, Framework, Architectures and Models for Electronic Commerce Group, CEN Workshop Agreement, February 003.
- Parr, Keith-Magee, Making the Case for Model Driven Architecture, SISO 2002.
- Dodge, Gateways – A Necessary Evil?, SISO 2000.
- Comella-Dorda, Lewis, Place, Plakosh, Seacord, "Incremental Modernization of Legacy Systems", CMU/SEI July 2001.
- [Batory 98] Batory, Don. "Product-Line Architectures," Smalltalk and Java in Industry and Practical Training, Erfurt, Germany, October, 1998. (see [ftp://ftp.cs.utexas.edu/pub/predator/stja.pdf](http://ftp.cs.utexas.edu/pub/predator/stja.pdf))
- [Aksit 99] M. Aksit, B. Tekinerdogan, F. Marcelloni, L. Bergmans. "Deriving Object-Oriented Frameworks From Domain Knowledge", TRESE Project, Department of Computer Science, University of Twente, The Netherlands 1999. (see <http://trese.cs.utwente.nl/publications/papers/deriveframeworks.pdf>)
- Czarnecki, Krzysztof; Eisenecker, Ulrich W., "Generative Programming: Methods, Tools, and Applications", Addison-Wesley, Reading, MA, 2000
- [Northrup 99] Northrup, Linda. "Essentials of Successful Product Line Practice," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, October, 1999.
- [Bosch 00] Bosch, Jan and Höglström, Mattias. "Product Instantiation in Software Product Lines: A Case Study," Second International Symposium on Generative and Component-Based Software Engineering, October 2000. (see <http://www.cs.rug.nl/~bosch/papers/InstantiationCaseStudy.pdf>)
- J. C. Cleaveland, "Program Generators Using XML and Java", Prentice-Hall, XML Book Series, 2001, <http://craigc.com/pg/>

Summary

The availability of reusable platform independent, enhanced gateways that can be easily adapted to varying applications will provide a tremendous cost savings via reduction of development costs. It will also enable improved interoperability between legacy and new systems.

This evolvable architecture approach is equally applicable to civilian and commercial domains where interoperability between desperate systems is business critical. Extension and adaptation serve to broaden market applicability.

An enhanced gateway architecture will serve to improve the capabilities of CCT's existing command and control product line, allowing CCT to offer interoperability options not previously available in military and commercial products.

Phase 1 of this research will evaluate alternative architectural strategies and methods for generation of gateways from the toolkit, culminating in a proof of concept prototype by the end of 2004. Pending funding approval, Phase 2 will create a fully functional toolkit of reusable gateway components that can be automatically composed into operational configuration, demonstrating both systems interoperability of heterogeneous systems and method of instance specific generation.

We are currently in the 1st phase of this process and are seeking collaborators and contributors to this effort. We currently have limited financial support from the Air Force

under the SBIR program. Other participants are welcome and desirable. Organizations interested in participating, contributing to the effort, or simply interested in obtaining more information should contact Greg Hupf, 321-264-1193 x108, hupfg@cctcorp.com or Rodney Davis, x120, davisrd@cctcorp.com.