"GENERATIVE GATEWAY TOOLKIT FOR HETEROGENEOUS C3I SYSTEMS"

Greg Hupf, & Rodney Davis
Command and Control Technologies, Titusville, FL

ABSTRACT
Command and Control Technologies Corporation (CCT) has developed a composable reference architecture and a comprehensive integrated software toolkit for developing interoperable platform-independent C3I gateway systems under Air Force sponsored research. Gateways provide interoperability among disparate systems by providing a transparent connectivity solution that eliminates the need to replace or modify the systems themselves. The new Generative Gateway Toolkit (GGTK) approach achieves this interoperability by leveraging common data-transport and data-exchange semantics, and abstracting hardware platform and device interface details from core gateway functionality. Software product line architecture methods are used to ensure large-scale reuse and provide the capability to specify at a high-level 'what to build'. Generative programming techniques extend this capability, using the high-level specifications to actually build functional component-based gateways. Central to the architecture is accommodation for variation and evolutionary extension of its communications and semantic processing services.

The scope of this research is focused on addressing new and legacy system interoperability problems caused by stove piped architectures and a proliferation of interfaces. Primary focus is placed on the current use of gateways to solve DoD Tactical Data Link (TDL) interoperability problems. Traditional gateway development approaches have led to a proliferation of custom point-to-point solutions built from the ground-up on a project-by-project basis. The new GGTK approach exploits the common and variable features of the communications gateway domain, combining reusable software assets that were systematically developed and assembled for generation into fully-functional customized gateways. Instead of coding new gateways, developers 'compose' new gateways by selecting data link protocols, I/O devices and runtime environments so GGTK can produce the actual gateway.

INTRODUCTION AND BACKGROUND
For many years, building gateway systems from the ground up has been the norm by government and industry. Gateways interconnect at least two different systems and operate on the data and/or the communications format. They provide information transfer that preserves the meaning and relationships of the information exchanged at every destination. Simply stated, gateways provide interoperability among disparate systems. By using a gateway, systems using different communications capabilities can communicate without requiring modifications to the existing systems. This communications solution has led to the creation of simple gateways that can be created quickly and relatively inexpensively. Unfortunately, these simple gateways often implement point-to-point solutions and therefore are not easily maintained or reused.

New gateway systems are usually custom-developed to satisfy a pending interoperability need on a particular project or program because existing systems are inadequate for the task, too costly to modify, and no commercial alternatives are available. These custom stove-piped gateway systems work well for their initial purpose, but require significant cost and resource investments over their life cycle 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. Large organizations, such as DoD, indirectly incur high costs caused by duplication of effort as their various systems and programs implement multiple custom point-solutions to a common problem.

In recent years, more and more software engineering organizations and communities have come to the realization that developing single dedicated systems for a particular domain often results in 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.

It is no longer economically feasible to handcraft and maintain a collection of one-of-a-kind gateway systems.

---

1 Dodge, Gateways, A Necessary Evil? The MITRE Corporation, published for the Simulation Interoperability Workshop, September, 2000
Software reuse has always been the logical solution to reduce these costs. The problem with implementing this solution is that the existing systems were not engineered with large-scale reuse in mind. To compound the problem, planned software reuse has been difficult to attain on a large scale until more recently. Studies indicate that a major cause of this problem is architectural mismatch. That is, the software structures of components from different sources 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 related systems must be developed in lieu of multiple independent systems. Consequently, the DoD must develop the infrastructure for the strategic reuse of gateway software.

By developing software architectures designed for families of systems, organizations worldwide are finally reaping the benefits of large-scale software reuse. These families of systems, known as Software Product Lines, 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. The process of producing a new gateway system becomes more of an integration effort than a development effort, where only a few system-unique requirements being newly implemented.

However, to produce new systems from reusable assets, it is necessary to be able to specify what to build, correctly integrate components, and actually build the system. Managing these complexities by hand is difficult and can be a significant challenge for any software development organization, even using a rigorous development process.

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 software framework or complete end-product can be automatically manufactured on-demand from a predefined set of elementary reusable components.

Model-based generation can be defined as a multi-paradigm approach that incorporates the following aspects: domain analysis, meta modeling, model driven generation, Template languages, domain-driven, framework design, and disciplined development process. Key features of the approach come from the field of domain engineering, in particular the differentiation between building a product platform including relevant application development tools, and building software applications. The product platform for a product family or product line is developed using domain-driven design principles, and the application engineering process is automated as far as possible using model-driven design techniques. Generative programming techniques that complement domain engineering methodologies provide guidance for managing iterations, co-coordinating parallel domain engineering and application engineering, designing model-driven generators, and designing domain-specific application engineering processes.

Generative programming extends the software product line concept by providing the capability to specify at a high-level what to build and, using configuration knowledge and generators, to actually build the system. Taking a generative approach to managing the development of a family of systems significantly simplifies and automates the system specification and production process.

The nature of a C3I system requires it to belong to a family of systems since it needs to interoperate within a given "domain" (e.g. DOD TDL). Overall, the system-of-systems interoperability problem extends beyond a better gateway solution. Today, the design and development of C3I systems is performed using a number of disparate tools and techniques. Data specification, structural definition, and design is performed multiple times by multiple contractors during the systems lifecycle, often well before the systems are deployed for mission operations. Development of custom interface software is necessary to allow these systems to interoperate, usually involving a manual and time-consuming design process.

Similarly, system-of-systems mission operations require the exchange of information across multiple heterogeneous missions using a common communications infrastructure but using different data definitions. Each system requires related initialization and parameterization information. Information must also be exchanged among all of the operational phases, systems and organizations within the domain.

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 for each system. This customization is inherently error-prone, resulting in the need to compare and revalidate at each step in the lifecycle. Unfortunately, the scope of the current gateway approach is limited to basic system interconnectivity and does not address reuse or standardization of semantic information within a domain.

---

2 Bettin, Model-Driven Software Development, 2004
Although solutions to these broader interoperability problems extend beyond a system's basic external communications interface boundary, the problems are similar and therefore the same concepts and technologies used to implement an enhanced gateway capability can be applied.

**APPRAOCH**

To address these issues, Command and Control Technologies Corporation (CCT) has developed a composable reference architecture and an integrated software toolkit for developing interoperable platform-independent C3I gateway systems. Central to the enhanced gateway architectural approach is accommodation for variation and evolutionary extension of communications and semantic processing services. The Generative Gateway Toolkit (GGTK) is a comprehensive and integrated set of software tools that enforce the new enhanced gateway architecture and simplify creation of interoperable communication gateways.

**GGTK Information Model**

The GGTK information model represents a high-level description of the types of information used to generate, build and deploy the executable components of a functional communication gateway system. This model is specifically tailored to the communications gateway domain so that the necessary information can be described using a semantic ontology that is typical and familiar to those working in the C2 domain. The gateway information is described using a straightforward human-readable textual language that is expressed as machine-readable XML metadata. This model provides the framework for organizing the gateway configuration information so it can be delivered to and reused by the GGTK tools and utilities.

GGTK gateways are composed from a managed set of reusable software assets that handle functional variabilities across different data link protocols, HW I/O interfaces, and target runtime environments. The composition rules for individual gateways are captured in user-defined specifications as described by the GGTK information model. These high-level specifications are initially used by GGTK to generate runtime gateway software via the AGSP code generator, as illustrated in Figure 2.

Overall, the GGTK toolkit consists of three major functional elements:

- An abstract *Information Model* specifically tailored to the communications gateway domain.
- *Auto-Generated Semantic Processing* technology (AGSP) uses generative programming techniques to auto-generate gateway software.
- The *Integrated Gateway Development Environment* (IGDE) provides a complete integrated development environment for rapidly producing, integrating, and testing new gateways.

The GGTK is designed to be extensible. User-defined components can be added at a number of built-in variation points. These are specifically designed to enable the necessary flexibility for the developer, while maintaining the overall integrity of the reference architecture.

**Figure 1.** The GGTK is a comprehensive and integrated set of software tools that enforce a new enhanced gateway architecture and simplify creation of interoperable communication gateways.

**Figure 2.** GGTK Information Model Overview for AGSP

Once the AGSP components are generated, use of the gateway composition metadata continues within the IGDE, generating controllers that integrate the AGSP component and other coherency logic with a runtime software framework compatible with the target build and execution environment. An integrated software test harness is typically included in the final package to help support gateway verification and certification efforts independent of the gateway generation environment. The result is a complete and integrated source code package that contains all the code necessary to build and test the specified gateway. Once this software package is built and deployed on any platform compatible with that specified in the information model metadata, the gateway production process is complete.
Individual gateways are modeled as a specific aggregation of build-time, init-time and runtime parameters that GGTK uses to identify and configure software assets to be generated new, reused, or externally referenced in order to produce the gateway instance. The GGTK Asset Manager tracks all the information model entities, including user-defined and built-in software assets, so they can be easily shared and reused by multiple gateway developers or different organizations.

The GGTK information model consists of metadata that categorize gateway composition information into the following high-level categories:
- Data link and communication protocol definition
- Interface hardware and I/O device characteristics
- Target build and runtime environment
- External algorithms and user-defined software solutions

GGTK supports different front-ends for ingesting this information into the generation process. A front-end is responsible for interpreting a particular semantic specification or format and producing an internal representation for the code generator. GGTK includes built-in metadata parsers that read and interpret this information and populate its internal data model. These default parsers are designed to ingest a specific defined set of metadata specifications. GGTK is designed to be independent and decoupled from any specific external schema or metadata implementation. GGTK does this through encapsulation and extensibility. The necessary gateway composition information is maintained by GGTK in its own internal representation. Other parsers can then be added as external 'plug-ins' without significantly impacting the existing GGTK design or internal implementation.

**Auto-Generated Semantic Processing (AGSP)**

Auto-Generated Semantic Processing technology (AGSP) uses generative programming techniques to auto-generate reliable, consistent, and highly efficient gateway software from XML metadata specifications. Not only does this significantly reduce the amount of software development effort required to produce a new gateway, but also produces more consistent results so overall quality is improved. As shown in Figure 3 the scope of AGSP addresses all encoding and decoding software needed for stream, message, and parameter based communications.

The approach enhances interoperability by assuring data integrity and consistent interpretation of meaning between collaborating nodes or communities of interest. Central to this concept is an emphasis on specification, and minimizing implementation, effectively replacing the design, implementation and debug of complex protocol conversions, custom device mappings, and unique data manipulation algorithms with abstract platform independent communications model.

![Figure 3, Auto-Generated Semantic Processing Concept](image)

The AGSP code generator contains a front-end that extracts the data processing specifications from the GGTK internal information model, and a back-end that produces language-specific source code (e.g. 'C'), that are connected via an internal pipeline called the road mapping system. The road mapping system is a language neutral knowledge base that knows how to transform bits in one format to bits in another. The road mapping system is extensible by the user so AGSP can be supplied with additional rules to handle new data types and encodings.

**Integrated Gateway Development Environment**

The Integrated Gateway Development Environment (IGDE) provides a complete integrated development environment for rapidly producing, integrating, and testing new gateways. Summarized in Figure 4, the IGDE integrates all the various information models, asset managers, code generators, and gateway composition services into a single development environment with an intuitive user-friendly graphical interface. The IGDE allows the developer to operate at a high level of abstraction to mix-and-match gateway link definition, target platforms, operating systems, and physical device interfaces. Using a Software Product Line approach to enable large-scale software reuse and built-in design patterns to ensure consistent and reliable results, the IGDE can enforce the new enhanced gateway architecture and significantly simplify gateway production and deployment.

The GGTK enhanced gateway architecture makes it practical to build, from a common set of platform-independent specifications, multiple functionally-equivalent gateways that can interoperate across a heterogeneous runtime environment. Complete
functional gateways can be produced quickly by simply selecting the necessary gateway assets, including GGTK built-ins as well as user-defined add-ons, and the IGDE handles the platform-specific software production and integration. Two of the key GGTK assets that enable this cross-platform capability are the Runtime Shells and the hardware interfaces.

**Runtime Shells**

GGTK Runtime Shells encapsulate common runtime services that are implemented differently across various target platforms and operating systems. Each runtime shell is an integrated set of platform-specific software components for deploying a user-configured GGTK gateway on a specific runtime target environment.

The runtime shell isolates the other platform-independent gateway components from platform-specific implementation details necessary to run on the target by providing a standardized abstract interface to its common services (e.g. timers, events) as illustrated in Figure 5. Developers use the IGDE to select from multiple interchangeable GGTK runtime shells to be integrated into the specific gateway instance (see Figure 5). For example, the gateway developer can re-platform an existing gateway by simply using the IGDE to replace the runtime shell and generate a new gateway software package that is complete and ready to build on the new target platform.

The current research effort completed development of a Linux Runtime Shell asset for the GGTK. The Linux Runtime Shell provides the common operating system services such as timers, memory allocation, process management, network protocols, event handling, etc.; as well as a set of standard I/O interfaces that include files, network sockets, command line, etc.

![Figure 4. Integrated Gateway Development Environment](image)

**External Interface Modules**

The GGTK Interface modules encapsulate the details necessary to communicate with external interfaces such as standard network sockets or custom I/O devices. Like the Runtime Shells, GGTK Interfaces isolate the platform-independent gateway components from platform-specific implementation details necessary to handle external data transfers on the target by providing a standardized abstract interface to its common I/O services (e.g. files, network). For example, the gateway developer can change an existing gateway to write its output to a file instead of a standard network socket by simply using the IGDE to replace the standard network output interface with a file interface. The boxes around the periphery of the GGTK Runtime Shell depicted in Figure 5 illustrate the GGTK Interface module concept.

In addition to the basic set of GGTK Interface components defined by the various target platform's I/O capabilities, more complex GGTK Custom Interfaces are created to isolate the platform-independent gateway components from implementation details related to custom external interfaces such as device drivers for custom I/O cards, third-party tools or software frameworks, custom APIs, legacy systems, etc. The boxes labeled 'Device' in Figure 5 represent GGTK Custom Interfaces.

The current research effort completed development of a GGTK Custom Interface module that provides multi-port synchronous serial I/O services via a COTS PCI card. This custom module successfully abstracts the serial bitstream interface between the HW device, the Linux device driver, and the runtime shell; providing the gateway with device transparency and application-level services for transmitting and receiving data.

**Asset Manager**

The GGTK Asset Manager allows the AGSP tool chain to track reusable assets and make them available during
gateway generation. While the enhanced gateway architecture enables large-scale reuse, and the IGDE simplifies the process of composing reusable assets into a functional gateway, it is the gateway developer who is responsible for specifying "what to build". To compose a gateway, the developer first needs the ability to determine "what exists". To this end, the GGTK asset manager provides the developer with a capability to access repositories of reusable assets via the IGDE graphical user interface. The asset manager makes these assets available as selectable components for use during product generation.

The asset management capability is designed to be robust and flexible to support various gateway development scenarios. Domain analysis indicates that gateway assets can include products that span a wide range of maturity levels, from formally controlled production-quality components that have been validated and certified, to experimental prototypes and example code distributed across development computers. The asset management capability is designed so that the same development environment used for rapid gateway prototyping and ad-hoc protocol testing can also be appropriate for generating production-level gateways.

All AGSP-related development tools use the asset manager to track and access the various assets that a gateway developer accumulates over time. As products such as platform, target, software, hardware, and messages models are created, they can be registered with the asset manager, which scans the specified file to identify the names and types of assets, and adds them to the personal asset database.

The asset manager also supports the capability to access reusable assets that are formally controlled as shared read-only products from a fixed repository. In this case, a configuration management organization or system administrator typically controls the actual assets. The asset manager uses order of precedence with a combination of file permissions and hierarchical search paths to locate and identify controlled assets in shared repositories as well as personal assets that may reside in a developer's home directory. New assets can be automatically registered and made visible to all or restricted to a few developers. Duplicate assets can peacefully co-exist and, when conflicts occur, they are resolved predictably.

**Semantic Mediator**

The GGTK semantic mediator provides robust on-demand dynamic mapping of gateway entities to simplify the composition process and avoid gateway component mismatches. Because the functionality required from real-world gateways can be complex, configuration and integration the requisite components can be tedious and require significant effort from the gateway developer.

Most reusable components are parameterized and require at least some interconnections to be specified by the developer as part of the gateway composition process. New gateway assets and information sources need to be brought into the gateway framework without disrupting existing mappings. XML is a platform and application neutral data representation language, but it leaves interpretation of the semantics to the information consumer. The GGTK semantic mediator is intended to help the developer deal with these configuration issues. The GGTK semantic mediator will apply domain knowledge and fusion to translate and map semantic information from these metadata descriptions, the actual parameter names, and context to suggest likely interconnections within the gateway composition framework.

To produce the intended gateway functionality and avoid irreconcilable data interpretations between components, developers configuring gateway components need the ability to determine "what is it" and "what does it do". The semantic mediation capability is intended to assist the developer with these decisions and minimize incompatibility problems. Once selected from a list of available assets, components for a given gateway instance need to be interconnected and integrated into the target gateway framework. Even for compatible components, the interfaces between them can be complex and the number of attributes required to configure and use them can be numerous. Only when the interconnections are defined can the GGTK controllers handle the individual bindings between the component parameters. Even when components use the same names for parameters that are intended to be interconnected, manually configuring a large number of connections can be tedious and error prone. The semantic mediation capability simplifies the overall parameter mapping process for the user.

An example of the GGTK semantic mediator user interface is shown in Figure 6. In this case, the mediator is being used to suggest connections between two controllers that have a number of unbound parameters remaining. The mediator identifies bindings that are compatible and available on the selected controllers, determines the functional relationships that are compatible between the binding types, scores the similarity between individual parameter names, and suggests the most appropriate one-to-one bindings that would complete the interconnections between the two components. The mediator rates the confidence of its suggestions based on the aggressiveness level specified by the user. In this case, the mediator found all the correct parameter mappings.
including the three whose names did not match. Once the user accepts the suggestions, appropriate logic will be generated and incorporated into the software framework for this gateway instance.

**SUMMARY AND CONCLUSIONS**

GGTK research is almost complete under the AFRL Phase II SBIR program and has shown significant promise for improving the efficiency of TDL gateway creation by enabling a product line reuse approach to what has traditionally been a software development activity. Some of the major accomplishments and findings to date include:

- Identified an extensible modeling approach for specifying communications, including concepts for describing communication protocols, hardware platforms, data formats, commands and measurement parameters, as well as rules and algorithms for data acquisition, conversion, processing and forwarding.
- Created an auto-code generator that produces highly efficient software for bit, byte, word, message, data type, and semantic transformation between interoperating heterogeneous systems, particularly ideal for resource constrained environments.
- Created a graphical IDE for composing a complete gateway software solution without having to write application specific software. The IGDE includes platform independent shell assets that allow gateways to be produce for multiple target hardware and operating system architectures, while enforcing a consistent and reusable gateway architecture. Extension points are provided for user extension and customization.
- Created a product line framework for managing gateway assets that include GGTK built-ins as well as user created extensions.
- Produced a fully functional suite of assets that have already yielded production level products for incorporation in to the CCT COTS C² Product Line.

Forward plans for the remaining effort include composition support for Link 16 and JREAP messages and demonstration of the GGTK in an operational environment. Since GGTK will be at approximately TRL 6 at completion of Phase II, Phase III funding is being pursued to bring GGTK up to certifiable maturity.