

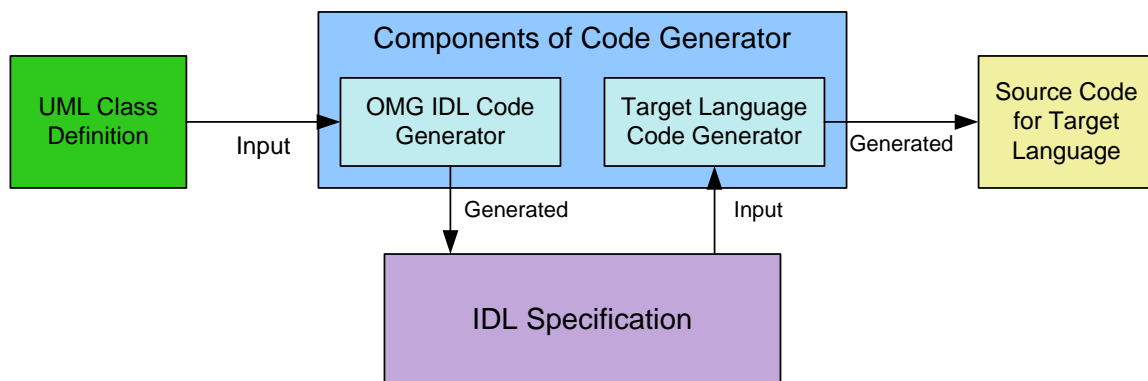
Code Generator Tool

Under contract to Boeing Company, Architecture Technology Corporation is developing a code generation tool (CodeGen) to support the Future Combat Systems (FCS) middleware – System of Systems Common Operating Environment (SOSCOE). SOSCOE must support a wide variety of applications. Some are best supported by a message-oriented communications paradigm, others by an object-oriented one. SOSCOE supports both. The message-oriented paradigm supports multiple connection types. The object-oriented interface supports a variant of the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) Remote Method Invocation (RMI) technique.

CodeGen translates inputs described in either OMG's Unified Modeling Language (UML) or Interface Definition Language (IDL) formats into C, C++, or Java source code for the supported communications paradigms enabling software developers to be more productive by providing a common, less complex interface eliminating many of the vagaries associated with multiple operating systems and dissimilar languages. It reduces system development and integration time by ensuring interoperability within and between objects running on different platforms and/or written in different languages.

The figure depicts CodeGen's basic structure. It accepts UML Class Definitions or IDL Specifications and produces source code in the selected target language. It consists of:

- a) **OMG IDL Code Generator** – accepts Data Type or RMI interface specifications expressed as UML classes and converts them into IDL specifications. It is implemented as a Rational Rose script.
- b) **Target Language Code Generator** – accepts interfaces expressed as IDL specifications and converts them into source code for the target programming language. The Target Language Code Generator is written in Java.



Data Types Code Generation

To provide application design flexibility SOSCOE supports multiple communication models, including: point-to-point, client/server, publish/subscribe, and groups (a peer-to-peer variation). Applications send and receive data to and from the connectors through the use of Data Writers and Data Readers and can control the behavior of the connectors by specifying Quality of Service (QoS) policies. Filtering on various attributes is also supported. CodeGen generates code to support type-safe marshalling/demarshalling of the data attributes and type-specific data reader and writer classes.

CodeGen uses the IDL specifications to generate C++ and Java classes with method implementations appropriate to the contained data attributes. CodeGen provides similar functionality within the constraints of the C language. CodeGen supports SOSCOE defined (non-IDL) Data Types and other IDL data declarations such as structs, unions, and typedefs, and supports the use of these constructed types as data attributes of the Data Type class.

RMI Code Generation

RMI enables remote clients to treat a remote server object as a local object. It requires type-safe marshalling/demarshalling of the method arguments and desired return value on both the client and the server sides. Given an IDL interface description, CodeGen generates a client stub class, server skeleton class, and server implementation class. It provides code for the following types of static invocation:

1. Synchronous with Timeout Method Invocation (SMI) – the client is blocked until a response is received from the server or a timeout occurs
2. Asynchronous Method Invocation (AMI) – the client is not blocked but receives a callback from the server
3. One-way Method Invocation (OMI) – like AMI except no callback is received

CodeGen supports methods declared as Role-Based Access Control (RBAC) protected in the IDL specification. An RMI client is prevented from accessing a server object method if the user's active role set and privileges do not meet the requirements of the RBAC-protected method.

CodeGen supports the Object Description Interface (ODI). SOSCOE supports both Static and Dynamic Invocation Interfaces (SII and DII). With a Static Invocation Interface (SII) the client provides a specific stub method for each RMI interface method. A Dynamic Invocation Interface (DII) provides more flexibility, in that a client does not need to bind to a specific stub object, but can instead discover an RMI server object providing a desirable service and then invoke the server methods by dynamically constructing RMI requests. ODI provides a means for an RMI client to determine the methods available from a given server object.