

SEMI E96-1101 (Reapproved 0307) GUIDE FOR CIM FRAMEWORK TECHNICAL ARCHITECTURE

This Standard was technically approved by the Information & Control Global Technical Committee. This edition was approved for publication by the global Audits and Reviews Subcommittee on November 21, 2006. Available at www.semi.org in March 2007; originally published February 2000; previously published November 2001.

NOTICE: This Standard or Safety Guideline has an Inactive Status because the conditions to maintain Current Status have not been met. Inactive Standards or Safety Guidelines are available from SEMI and continue to be valid for use.

1 Purpose

1.1 This Guide describes technical architecture choices that enable application components to cooperate in a computer integrated manufacturing (CIM) environment and reduce the effort required to integrate those components into a working solution. The CIM Framework technical architecture guide builds on publicly available specifications for distributed object computing. It defines manufacturing production systems requirements for the technical infrastructure needed for improved component interoperability, substitutability, and extensibility. It provides guidance for specifying components and addresses options for using an underlying distributed object communication infrastructure.

1.2 This Guide provides guidance for the technical foundation of the SEMI Computer Integrated Manufacturing (CIM) Framework Standards. It discusses a component-based architecture using object-oriented and framework technology that helps implementers achieve component interoperability and substitutability, application extensibility, and reuse. It establishes the role of distributed object communications infrastructure in providing necessary support for the framework technology. Specification methods for mapping a CIM Framework specification to alternative infrastructure technologies are also addressed by this technical architecture. However, these mappings are not intended to be prescriptive. Further work may be required to define additional mappings to emerging technologies. Many implementation issues that should be resolved for a particular software implementation are outside the scope of this Guide.

1.3 Adhering to this Guide for technical architecture alone does not provide interoperability between applications. While the technical architecture provides a foundation for interoperability, it is limited by the following factors:

- Multiple infrastructure implementation choices are possible, and interoperability across these environments is not guaranteed.
- The technical architecture intentionally limits its scope to only the most fundamental infrastructure requirements, leaving additional technical issues for future guide upgrades or for implementers' discretion.
- Conformance to a specification for CIM Framework Domain Architecture is also required for interoperability of domain components.
- More complete semantics (including behavioral constraints and collaboration patterns) for components are needed to ensure consistent interactions among components developed by separate suppliers.

1.4 A guide for technical architecture is a necessary, but not a sufficient, basis to achieve the goals of the CIM Framework specifications. It does not mandate specific solutions to address the identified technical requirements because there are multiple implementation choices that meet these requirements. Rather, the technical architecture identifies those crucial technical requirements that should be considered by both CIM software suppliers and consumers. The proposed standard identifies the technical capabilities implementations should provide, but leaves the implementation options open. It is the responsibility of suppliers to provide and explain an implementation of each capability, and the responsibility of consumers to assess particular implementations for use in their factories.

1.5 This Guide provides guidance on the technical tradeoffs for services provided by the distributed computing infrastructure for the purpose of supporting and enabling the domain specifications of CIM Framework components. These areas are:

- *Distributed Object Communication* — Provides the basic services to enable implementations supporting the CIM Framework interfaces to transparently locate other, possibly distributed implementations and exchange messages

requesting standard CIM Framework operations. Interface Definition Language provides a formal specification of the CIM Framework interfaces that can be automatically transformed into conformant implementations ready for integration and interoperation.

- *Exception Declarations* — Identify the form and structure of return messages that inform requestors that a requested operation resulted in an anticipated, but abnormal outcome.
- *Event Specification* — Establishes the delivery mechanism, identification conventions, and data structures for reporting the occurrence of anticipated state changes to CIM Framework objects.
- *Distributed Transactions* — Define mechanisms needed to coordinate the start, completion, or rollback of units-of-work that cross CIM Framework component boundaries.
- *Component Manager Support* — Identifies the component-level operations needed to create, locate, or remove instances of objects (and manage collections of those objects) that support the CIM Framework specified interfaces.

2 Scope

2.1 *Intended Audience*

2.1.1 This Document is intended for developers of components and applications, and integrators of MES systems that adhere to the CIM Framework specifications. It is also intended for system architects who contribute to the evolution of the CIM Framework architecture and guides based on implementation experience. A guide for technical architecture is focused on the software technologies that support the architectural goals for the CIM Framework rather than on the manufacturing domain concepts that the CIM Framework encompasses. The technical architecture perspective complements SEMI E81.

2.2 *Architectural Issues Not Covered*

2.2.1 A number of architectural issues are not covered within this document because they are beyond the scope of the CIM Framework standards and are not expected to come within the scope of the standards as they are revised. They are itemized here because a product architecture layered on the CIM Framework Technical Architecture should address these additional architecture issues. In these cases, other more general specifications emerging in the infrastructure technology areas are expected to provide these needed standards. The CIM Framework domain specifications do not require specific conformance in these areas to support component specifications.

2.2.2 *Persistence*

2.2.2.1 Persistence refers to the ability of an object to maintain a nonvolatile copy of its current state such that the object could recreate the state during a future initialization. There are various operations for object persistence, and problems can occur if objects with cross-references do not coordinate their persistence strategies and mechanisms. The CIM Framework excludes persistence as an implementation mechanism.

2.2.3 *System Performance*

2.2.3.1 System performance is highly dependent on the selection of hardware and software platforms for system execution. Tests should be performed to verify adequate system performance and scalability for the anticipated operating environment. Performance tuning mechanisms or measurement tools are excluded from the CIM Framework specifications as an implementation dependent mechanism.

2.2.4 *Data Replication*

2.2.4.1 Data replication is a technique used to provide additional fault tolerance or improve system performance in certain situations. The CIM Framework excludes specification of replication strategies as an implementation dependent mechanism.

2.2.5 *Change Management*

2.2.5.1 Change management is the ability to introduce and control changes to the system configuration. The CIM Framework encompasses change management in the domain context of document control, but the CIM Framework excludes the broader treatment of change management for the MES software configuration itself.

2.2.6 Externalization

2.2.6.1 Externalization can be used to provide a form of persistence or to transfer object state between disjoint implementations. The ability of an object to externalize its data and state supports recovery of data and state for objects that terminated from memory. The CIM Framework excludes externalization as an implementation dependent mechanism.

NOTICE: SEMI Standards and Safety Guidelines do not purport to address all safety issues associated with their use. It is the responsibility of the users of the Documents to establish appropriate safety and health practices, and determine the applicability of regulatory or other limitations prior to use.

3 Limitations

3.1 The CIM Framework should continue to evolve to meet the needs of a competitive and vital industry. The content of this framework represents a significant amount of real development experience from a number of commercial software suppliers and their customers. These specifications reflect the product architectures of those companies, as well as the requirements of their customers. This evolution process should continue as more products based on the CIM Framework are developed.

3.1.1 This Guide acknowledges the following deficiencies that should be addressed in future revisions. These deficiencies are identified in the following sections.

3.2 Mapping to Alternate Distributed Computing Infrastructures

3.2.1 The CIM Framework provides a specification for MES software components, specified in terms of generalized manufacturing production systems requirements that may be implemented using a variety of technical infrastructure foundations. While the intent of this Guide is to provide both rigor in specification and flexibility to make infrastructure implementation choices, these goals often conflict. The use of mapping techniques complicates the task of integrating applications across technology boundaries.

3.2.2 While it is anticipated that a conformant implementation using either CORBA^{®1} or the Microsoft^{®2} Distributed Component Object Model (DCOM) is feasible by mapping the specifications to the implementation, it is recognized that cross-infrastructure integration is significantly more difficult (e.g., merging transaction models). The mapping described here offers more diverse implementation choices, but it does not guarantee that all of those chosen technologies easily work together in a single heterogeneous implementation.

3.2.3 Although the DCOM mapping provides a straightforward transform from the OMG[®] Interface Definition Language (IDL)^{TM3} specifications for static invocation, the Microsoft OLE Automation interfaces may be required for dynamic invocation. There appears to be greater risk in being able to successfully map the CIM Framework to the OLE Automation interfaces. The requirement for dynamic invocation should be evaluated with this in mind.

3.2.4 Another issue with DCOM mapping concerns exceptions. DCOM returns exceptions using its return value HRESULT. Many CIM Framework operations already use return values and would not be able to return a HRESULT without restructuring the return mechanism for the operation results.

3.2.5 Finally, there has not yet been a detailed analysis of the CIM Framework interfaces to verify that they can be successfully mapped using the CORBA Interworking Architecture.⁴ This is especially true of the OLE Automation mapping resolution.

3.3 Business Rules

3.3.1 The management of factory objects requires the use of a set of business rules; that is, procedures representing common business practices that should be applied under a given set of circumstances in response to some factory event. For instance, “Do not assign a process job to a machine which is scheduled for maintenance within 24 hours.” Factory systems implementations typically specify business rules as event-driven ECA (event-condition-action) or ECAA (event-condition-action-alternative action) rules. For example:

¹ CORBA is a registered trademark of Object Management Group, Inc. in the United States and other countries.

² Microsoft is a registered trademark of Microsoft Corporation, Inc. in the United States and other countries.

³ OMG Interface Definition Language (IDL) is a trademark of Object Management Group, Inc. in the United States and other countries.

⁴ Object Management Group. The Common Object Request Broker: Architecture and Specification, Revision 2.2, Object Management Group, 492 Old Connecticut Path, Framingham, MA: Object Management Group, 1998.

- *event* — request to edit a process specification,
- *condition* — invalid user access privilege,
- *action* — deny access,
- (*alternative action* — deny access and report breach of security).

3.3.2 Business rules can also be embodied in the sequencing logic of sequential process definitions. In this case the business rules define the criteria for making sequencing decisions that affect the flow of work through the factory.

3.3.3 Business rules are intended to be addressed in future revisions of this Guide.

3.4 *Security and Access Control*

3.4.1 The management of sensitive information regarding business processes and product specifications requires that MES implementations (especially distributed systems) provide some level of security and access control services. Typically, such services:

- identify and authenticate any factory object seeking sensitive information,
- permit access to information or operations based upon identity and privilege,
- provide security-related audit trails,
- provide secure communications (not susceptible to being intercepted nor malicious or inadvertent modification), and
- administer an enterprise's security policy.

3.4.2 Security and access control is intended to be addressed in future revisions of this Guide.

3.5 *Internationalization*

3.5.1 Specifications that deal with issues related to internationalization are emerging from several sources, including the OMG. This Guide should encompass the need and ability to incorporate internationalization features into the CIM Framework specifications.

3.5.2 Internationalization are intended to be addressed in future revisions of this Guide.

3.6 *Object Properties*

3.6.1 Object properties refers to a technique that allows additional attributes (data) to be dynamically associated with an object without changing the interfaces of the objects to which the properties are attached. This can be used as a convenient dynamic extensibility mechanism that may be considered in future CIM Framework specifications.

3.6.2 Object properties are intended to be addressed in future revisions of this Guide.

3.7 *Object Collections and Queries*

3.7.1 Object collections and queries allow flexible access to aggregate data for a group of objects. This capability may be a candidate to replace the limited operations for collections found in the component manager interface.

3.7.2 Object collections and queries are intended to be addressed in future revisions of this Guide.

4 **Referenced Standards and Documents**

4.1 *SEMI Standards and Safety Guidelines*

SEMI E30 — Specification for the Generic Model for Communications and Control of Manufacturing Equipment (GEM)

SEMI E40 — Specification for Processing Management

SEMI E81 — Provisional Specification for CIM Framework Domain Architecture

NOTICE: Unless otherwise indicated, all documents cited shall be the latest published versions.

5 Terminology

5.1 Abbreviations and Acronyms

- 5.1.1 *ACID* — Atomicity Consistency Isolation Durability
- 5.1.2 *CIM* — computer integrated manufacturing
- 5.1.3 *ECA* — event-condition-action (rule)
- 5.1.4 *ECAA* — event-condition-action-alternative action (rule)
- 5.1.5 *ENS* — event notification system
- 5.1.6 *ERP* — enterprise resource planning
- 5.1.7 *GUI* — graphical user interface
- 5.1.8 *MES* — manufacturing execution system
- 5.1.9 *ODL* — Object Definition Language
- 5.1.10 *OMA* — Object Management Architecture
- 5.1.11 *OTS* — Object Transaction Service

5.2 Definitions

- 5.2.1 *application* — 1. one or more programs consisting of a collection of interoperating objects which provide domain specific functionality to an end user or other applications. 2. functionality provided by one or more programs consisting of a collection of interoperating objects.
- 5.2.2 *application interface* — the interface provided by an application or application program.
- 5.2.3 *application object* — an object implementing an application interface.
- 5.2.4 *architecture* — the structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time.
- 5.2.5 *attribute* — an identifiable association between an object and a value. An attribute may have functions to set and retrieve its value.
- 5.2.6 *behavior* — the effects of performing a requested service, including its results (e.g., changes in the state of an object).
- 5.2.7 *binding* — a specific choice of platform technologies and other implementation-specific criteria.
- 5.2.8 *class* — the shared common structure and common behavior of a set of object implementations.
- 5.2.9 *client* — an object that uses the services of another object by sending messages to it or referencing its state.
- 5.2.10 *collection* — an object containing references to (collections of) other objects with services for managing them and providing access to them as a related group of objects.
- 5.2.11 *component* — a reusable package of encapsulated objects and/or other components with well-specified, published interfaces. The component is the element of standardization and substitutability for the CIM Framework.
- 5.2.12 *computer integrated manufacturing* — an approach that leverages the information handling capability of computers to manage manufacturing information and support or automate the execution of manufacturing operations.
- 5.2.13 *conformance* — adherence to a standard or specification in the implementation of a product, process, or service.
- 5.2.14 *conformance requirement* — identification in the specification of behavior and/or capabilities required by an implementation for it to conform to that specification.
- 5.2.15 *conforming implementation* — an implementation that satisfies all relevant specified conformance requirements.
- 5.2.16 *event* — an asynchronous message denoting the occurrence of some incident of importance. For example, state change or new object created.

5.2.17 *event channel* — the intermediate object that forwards published events to interested subscribers.

5.2.18 *exception* — an infrastructure mechanism used to notify a calling client of an operation that an unusual condition occurred in carrying out the operation.

5.2.19 *extensibility* — the ability to extend or specialize existing components and add new object classes or components while preserving architectural integrity and component conformance to standards.

5.2.20 *framework* — a collection of classes or components that provide a set of interoperable services and functionality for a particular domain.

5.2.21 *implementation* — the internal view of a class, object or module, including any non-public behavior. The specific code and functionality that implements an interface.

5.2.22 *implementation conformance statement* — a statement made by the supplier of an implementation or system claiming to conform to one or more specifications and stating which capabilities have been implemented. It specifically includes the relevant optional capabilities and limits.

5.2.23 *infrastructure* — the services, facilities, and communications mechanisms that support the collaboration between and lifecycle of distributed objects.

5.2.24 *inheritance* — the ability to derive new classes, types or interfaces from existing classes, types or interfaces. For example, a derived class ('subclass') inherits the instance variables and methods of the base class ('superclass') and may add new instance variables and methods. In the CIM Framework, inheritance applies to interfaces and their specification of operations rather than implementations of classes.

5.2.25 *instance* — a software entity that has state, behavior, and identity. The terms instance and object are interchangeable. An object is an instance of an interface if it provides the operations, signatures, and semantics specified by that interface. An object is an instance of an implementation if its behavior is provided by that implementation.

5.2.26 *interface* — the external view of a class, object, or module that emphasizes its abstraction while hiding its structure and internal behavior. An interface definition ideally includes the semantics of attributes and operations.

5.2.27 *interoperability* — the ability for two applications or the parts of an application to cooperate. In the CIM Framework, interoperability requires that application components be able to support specified relationships, share data, invoke each other's behavior (operations), return exceptions, and exchange events.

5.2.28 *lifecycle* — the life of an object, including creation, deletion, copy, and equivalence.

5.2.29 *message* — in object oriented systems a message is the means by which a client object invokes the behavior specified by an operation of a server object.

5.2.30 *message bus* — a software infrastructure that provides distributed communication between objects in component implementations. It can refer to an Object Request Broker, Microsoft DCOM, Java Remote Method Invocation or other infrastructure for conveying messages between objects.

5.2.31 *name-value pair* — a data structure that associates a name with an arbitrary value, typically used as an extensibility mechanism for conveying information by name-based retrieval.

5.2.32 *namespace* — a namespace is a bounded collection of names with a constraint to ensure that each name is unique within the collection.

5.2.33 *object* — a software entity that has state, behavior, and identity. The terms instance and object are interchangeable. An object is an instance of an interface if it provides the operations, signatures, and semantics specified by that interface. An object is an instance of an implementation if its behavior is provided by that implementation.

5.2.34 *object services* — interfaces for general services that are likely to be used in any program based on distributed objects.

5.2.35 *operation* — an operation is a specification entity, identified by an operation identifier, that denotes a service that can be requested. An operation has a signature that describes the legitimate values of request parameters and returned results, including any exceptions.

5.2.36 *persistent object* — an object that can survive the process or thread that created it.

5.2.37 *productive entity* — productive entity is an abstraction of a physical unit, which is involved in any way in a production process (e.g., production or supporting equipment). A productive entity has its own internal logic and provides a software interface to access this logic.

5.2.38 *query* — a message sent to a server (e.g., the productive entity) by a client interested in some information from the server (state of the productive entity). A query may or may not have arguments and it always has an answer. The semantics of a query is that some information from the server is returned, but the query cannot effect any change to the state of the server.

5.2.39 *service* — a function provided by a service provider that is performed through an operation specified by the provider.

5.2.40 *service provider, server* — an object providing services to other objects as specified by its published operations.

5.2.41 *signature* — a signature is the name, parameters, return values, and exceptions for a specific operation.

5.2.42 *substitutability* — the ability to replace a given component from one supplier with a functionally equivalent component from another supplier without impacting the other components or its clients in the system.

5.2.43 *trader service* — a collection of names with associated properties of features for each name and methods for manipulating and inspection that collection.

5.2.44 *type* — a declaration that describes the common properties and behavior for a collection of objects. Types classify objects according to a common interface; classes classify objects according to a common implementation.

6 Technical Architecture Guidance

6.1 The computing infrastructure provides the distributed computing environment for CIM Framework applications. This infrastructure includes the operating system, networking and communications, data storage and access, user interface and presentation services, event distribution, systems management, and many other elements. Of these many infrastructure elements, a guide for technical architecture specifies a small subset of key services that need to be standardized in order to facilitate and streamline system integration between conformant CIM Framework implementations.

6.2 The CIM Framework relies on publicly available specifications to define the use of infrastructure services wherever such published specifications exist. The largest single source for openly defined specifications for distributed object services is the Object Management Architecture.⁵ Reference to this Guide does not imply that CIM Framework conformant implementations should implement the referenced services. The implementations that realize these infrastructure technologies are outside the scope of this Guide. The technology choices made by implementers should be kept transparent to the CIM Framework to the greatest extent possible.

6.3 Distributed Object Communication

6.3.1 The CIM Framework documents assume the use of software infrastructure to provide distributed communication between objects in an implementation. The acronym ORB was originated by the OMG to describe its distributed object communication infrastructure, but is sometimes used in a more general sense. In this Document ORB is used only to refer to the OMG specified technology and the more general term ‘message bus’ is used for the diverse class of distributed communication mechanisms for communication between objects. The message bus is used to allow objects to make requests and receive responses from other objects. An object can communicate through the message bus with objects that are local or remote. Location transparency allows the object to remain ignorant of the actual location of the object with which it communicates.

6.3.2 A primary criterion for a message bus implementation is its ability to deliver all messages specified by the interfaces of the CIM Framework components. To accomplish this, the message bus should provide the ability to support or map a specified interface, including its inherited features, data types, operations, object references, and exceptions to and from runtime marshaled transport formats.

6.3.3 Alternate message bus implementation technologies are supported by the CIM Framework by mapping the OMG IDL for the CIM Framework interfaces into a specific message bus implementation. The Common Object

⁵ Object Management Group. The Object Management Architecture Guide, Revision 3.0, John Wiley and Sons, New York NY, 1995.

Request Broker: Architecture and Specification⁴ contains sections that define this mapping from CORBA to COM and from CORBA to OLE Automation. These sections, called the ‘Interworking Architecture’, cover detailed rules for mapping OMG IDL, types, and exceptions to compatible interfaces in COM and OLE Automation.

6.3.4 From a high level perspective, the DCOM and CORBA message buses are comparable. The DCOM capabilities are roughly equivalent to those of an ORB. However, with a lower level analysis, differences show up in data types, inheritance, object identity, and the handling of exceptions. The CORBA Interworking Architecture⁴ (Chapter 15, “Interworking Architecture,” Chapter 16, “Mapping COM to CORBA,” and Chapter 17, “Mapping OLE Automation to CORBA”) defines mapping approaches covering:

- Interface Mapping,
- Interface Composition Mapping, and
- Identity Mapping.

6.3.5 These areas should be addressed in order to provide a mapping between the OMG IDL used to specify CIM Framework interfaces and the message bus used for implementation. If the mapping is not specific (i.e., can occur in multiple ways) then two implementations may not necessarily be able to communicate even if they use the same message bus type. Although the CORBA Interworking Architecture is specific to Microsoft technologies, it could provide the foundation for future interworking mappings.

6.3.6 The current CIM Framework interfaces are specified in OMG IDL. The interfaces can be directly compiled and used with any of the available ORB implementations on the market.

6.3.7 The only way to provide such a direct solution using Microsoft DCOM would be to create additional CIM Framework interface specifications in DCOM MIDL and/or OLE Automation ODL. This would allow direct support for message bus functionality using DCOM. For example, the CIM Framework memory management requirement of ‘in’ for parameter passing (see ¶ 6.3.15) would be directly supported by MIDL and DCOM but since the CORBA exception model is significantly richer than the DCOM exception model, mapping CORBA exceptions to COM would require an additional protocol to be defined for DCOM.

6.3.8 The CORBA Interworking Architecture supports mapping the current CIM Framework interfaces defined in OMG IDL to DCOM MIDL or OLE Automation ODL. This mapping is detailed enough that the mapping should always provide the same MIDL/ODL solution. Even though the current scope does not include interoperability between implementations on DCOM and CORBA (see § 3.2), the issue of mapping the interface is still the same. This would also provide a step towards true interoperability using CORBA/DCOM bridge products that are beginning to become available.

6.3.9 Microsoft also provides an extension called OLE Automation. These interfaces are described in Object Definition Language (ODL). The OLE interfaces can be invoked dynamically by a client with no compile-time interface knowledge. The OLE data types are a subset of the types supported in DCOM, and there is no support for user-defined constructed types. The mapping solution differs for the DCOM and the OLE Automation. OLE Automation does not provide as clean a mapping from OMG IDL as DCOM does. This limitation may not allow some of the interfaces to translate completely to an OLE Automation implementation. Thus, component suppliers using an OLE Automation implementation should explain impact on interfaces that were not fully supported due to the Automation restrictions.

6.3.10 The CORBA Interworking Architecture covers mapping issues for the major areas of concern for the CIM Framework. The areas of primary importance are the Interface mapping, Interface composition mapping, Identity mapping, and Exception mapping. The CORBA Interworking Architecture gives detailed mappings for each of these areas and deals with the DCOM and OLE Automation mappings separately. The following five subsections summarize these mapping issues.

6.3.11 *Interface Mapping*

6.3.11.1 The OMG IDL primitives, constructed data types, and object references map closely to DCOM. The inherited CORBA interfaces may be represented as multiple DCOM interfaces. The CORBA attributes may be mapped to get and set operations in DCOM interfaces.

6.3.11.2 The OMG IDL primitives map to OLE primitives except for special cases. The OLE interfaces do not support constructed data types and should be mapped to specially constructed interfaces. CORBA object references map to