A Framework for Control Architectures

M. K. Senehi

Thomas R. Kramer

Citation

M. K. Senehi and T. R. Kramer; "A Framework for Control Architectures"; International Journal of Computer Integrated Manufacturing; Volume 11, Number 4; July-August 1998; pp 347 - 363.

Abstract

The development of architectures for control systems has been an active area of research for at least twenty years. This research has produced many different architectures which use different terminologies and address different issues. In order to analyse existing architectures and to determine issues that must be addressed in defining a new one, a common terminology for discussing architectures and a framework for organizing information about architectures is needed. Based upon an examination of many control architectures in the areas of computer integrated manufacturing and robotics, the authors propose a terminology for discussing control architectures and a framework for constructing control architectures. The authors have used the terminology and framework to develop a set of issues which need to be addressed when constructing a control architecture. These issues have been used to compare two architectures developed at the National Institute of Standards and Technology. This paper presents the terminology, the framework, and issues related to constructing an architecture using the framework.

Disclaimer

No approval or endorsement of any commercial product by the National Institute of Standards and Technology is intended or implied.

Acknowledgments

Partial funding for the work described in this paper was provided to Catholic University by the National Institute of Standards and Technology under cooperative agreement Number 70NANB2H1213.

1 Introduction

The development of architectures for control systems has been an active area of research for at least twenty years [Bittici]. This research has produced many different structures for understanding and constructing control systems. Usually these structures are called either architectures or frameworks. While the literature uses the terms architecture and framework almost interchangeably, we will make a distinction. We will use the term *architecture* to refer to a description of the design and structure of a system and the term *framework* to refer to a structure *external* to an architecture, which organizes information about the architecture and the application of the architecture.

A *control architecture* is an architecture for a control system. Different control architectures address a wide range of issues and discuss these issues in varying terminology. The variability in terminology makes it difficult to understand the results from the development and application of other researchers' architectures; the breadth of issues makes it difficult to use these results.

To remedy this situation, it is necessary to have a common terminology for discussing control architectures and a common framework for organizing information about control architectures. While some work in these areas has been done, neither a terminology nor a framework has been universally accepted.

Based upon the examination of many control architectures for computer integrated manufacturing (CIM) and robotics, the authors propose a terminology for discussing control architectures and a framework for describing control architectures. Together these give a vocabulary and a structure for discussing the construction of a control architecture. Since our framework is more general than those described in the literature, it can also be used to relate frameworks to each other.

This paper presents our proposed terminology and framework, relates them to other proposed terminologies and frameworks and discusses issues related to the construction of an architecture using the our framework.

1.1 Related Research

In the following sections, we will discuss literature related to both terminologies for control architectures and frameworks for architectures. While our discussion concerning standardized terminology is confined to the literature on control architectures, our discussion about frameworks includes literature from several related fields. The search for a standardized terminology is discussed in Section 1.1.1, and frameworks are discussed in Section 1.1.2.

1.1.1 Terminology

Although the need for a standardized terminology in the field of control architectures has long been recognized, there have been few efforts to define a standardized terminology. One notable effort is a glossary of standard computer control system terminology [ISA] prepared by the Glossary Committee of the Purdue Workshop on Standardization of Industrial Computer Languages held at Purdue University in 1969.

This glossary was updated in 1985 and contains definitions for many useful terms like "controller", "real-time", and "response-time". However, it is oriented toward physical systems and does not contain terms needed by more abstract architectures.

Currently, International Standards Organization Technical Committee 184, Subcommittee 5 Working Group 1 is constructing a framework for enterprise integration which can be used to coordinate standards that pertain to a Computer Integrated Manufacturing (CIM) enterprise. While the primary thrust of this effort is to identify constructs for modeling a CIM enterprise, a terminology is also under development. The results of this effort are not yet mature enough for use [ISO2]. Recently, International Standards Organization Technical Committee 184, Subcommittee 5 Working Group 3 (Electronic Data Exchange) and the ISO Central Secretariat have begun to define a semantic repository of terms which would be unambiguous across several different languages.

1.1.2 Frameworks

The framework which the authors are presenting is geared toward providing a structure for identifying issues which must be addressed in constructing a control architecture. As far as the authors are aware, there is no other framework specifically designed for this purpose. However, there are many frameworks which serve either to aid in the classification of architectures or to organize information about architectures. While a comprehensive literature survey is beyond the scope of this paper, we will indicate some of the important work which has been done in this area.

1.1.2.1 Systems Theory

Systems theory provides a holistic approach to analyzing many diverse types of systems. It has been used to analyze systems in the fields of biology, management, and engineering to name a few. The field of cybernetics, in particular, is concerned with control and communication theory [Checkland]. The framework for enterprise integration under construction by the International Standards Organization Technical Committee 184 Subcommittee 5 Working Group 1, maps systems theory concepts into concepts useful for modeling a CIM enterprise [ISO2]. This framework's stated purpose is "to provide a framework for the coordination of existing, emerging, and future standards for the modeling of manufacturing enterprises to facilitate Computer Integrated Manufacturing (CIM)". The framework identifies a CIM enterprise as a hybrid system with human, physical, and conceptual entities. It identifies key concepts for modeling an enterprise, dividing these concepts into the broad categories of business modeling, organization/technical structure, and operation of a manufacturing facility. Particular emphasis is placed on concepts for business modeling and organization/technical structure.

1.1.2.2 Zachman's Framework for Information Systems Architecture (ISA)

Zachman's framework for information architecture has gained acceptance among those familiar with systems theory for analyzing both information systems and more general systems. "It provides a systematic taxonomy of concepts for relating things in the world

to the representations in the computer" [Zachman]. This framework was derived by analogy from examples in the disciplines of architecture and manufacturing. The framework consists of five rows and three columns, creating 15 boxes for organizing information. The rows represent different views of the information, and in each case a different type of model is required. The rows are:

(1) scope—the overall model of the business

(2) enterprise model—model of the business from the point of view of the owner

(3) system model—model of the information system of the business

(4) technology model—design of the technological implementation of the model in (3)

(5) detailed description—detailed design for the implementation of the technology model

The three columns represent views or ways of looking at the data; these are data, function, and network (or location). Each of the 15 cells may contain a type of information qualitatively different from that of other cells and require a separate type of representation. In 1992, Zachman and Sowa broadened the original framework to include three additional columns, corresponding to who is using the information, when the information is to be used, and why (for what purpose) the information is to be used. A formal mechanism for expressing the ISA framework based on conceptual graphs has also been added [Sowa].

1.1.2.3 CIM Frameworks

Within the field of Computer Integrated Manufacturing (CIM), many frameworks either for classifying architectures or for organizing information about architectures have been advanced. We will discuss only three which we feel to be important and representative.

The Computer Integrated Manufacturing Open System Architecture (CIM-OSA) is an open architecture for the integration of the design and operation of a CIM enterprise. It was developed by the AMICE consortium of ESPRIT, a European program aimed at improving the competitiveness of member companies. CIM-OSA has produced, as part of the architecture, a Modeling Framework which gives main dimensions for modeling a CIM enterprise. This has been accepted as a European pre-standard [CEN].

The framework provides a structure for organizing information about architectures. There are three dimensions in this framework: architectural genericity, modeling, and views. Architectural genericity is a measure of the applicability of a concept across the manufacturing domain. Three levels are identified along this dimension: generic, partial, and particular. Generic means the concept applies in any enterprise; particular means that the concept applies only to a specific enterprise; partial means that it applies to some subset of enterprises. The modeling dimension corresponds to traditional software engineering stages: requirements definition, design, and implementation. The

dimension of views identifies distinct aspects of the enterprise which must be modeled. The views expressed in CIM-OSA are function, information, resource, and organization. [Sastrón] provides a lucid overview of the CIM-OSA architecture.

Two frameworks designed to identify dimensions for the comparison and classification of CIM architectures are advanced by Bohms and Biemans. We will discuss each of these in turn.

In [Bohms] nine dimensions for classifying Computer Integrated Manufacturing systems are identified.

(1) modeling level—which distinguishes between different meta-levels of modeling. Bohms lists three meta-levels: reality itself, models of reality, models of models (frameworks),

(2) language level—different levels of modeling language are required, including those in which to express languages,

(3) aspect—a set of views which are thought to be important for modeling CIM, e.g., function, information, resource,

(4) composition—the amount of detail included in the model, ranges from global to detailed,

(5) scope—the range of applicability of the architecture,

(6) representation—different ways may be needed to express the same semantic content for different purposes and using different languages,

(7) product life cycle—which part of the product life cycle the architecture includes, e.g., design, production, maintenance, etc.

(8) actuality—whether the architecture is to apply to currently existing systems or to future systems,

(9) specification level—the degree of choice left in the architecture for the implementor.

Section 4 of [Bohms] proposes decompositions of each of the nine dimensions into points or regions. For example, the modeling level dimension has three points: CIM Framework, CIM Models, CIM in Practice.

In section 2 of [Biemans] a second set of dimensions is identified. They are not explicitly called dimensions in the paper.

(1) flexibility—the ability to accommodate changes in products, operations or facility layout[1],

(2) precision of architecture definitions—the degree of ambiguity remaining in architectural definition,

(3) generality of a CIM architecture—applicability over different production organizations over time,

1. section heading in [Biemans] is “allocation of tasks”

(4) level of abstraction of a CIM architecture—a facility can be described at many levels of abstraction above the physical level.

The two sets of dimensions just described reveal the difficulty of establishing a comprehensive method of characterizing architectures. Although the area covered by the last three of Biemans' dimensions is largely covered by Bohms' dimensions, in no case is there good match between a single dimension from one set and a single dimension from the other. Each of the dimensions may be of interest in some situations.

1.1.2.4 Frameworks for Integrating Manufacturing Enterprises

In 1990, a joint task group of the IFAC (International Federation of Automatic Control) Manufacturing Technology and Computers Committees and the IFIP (International Federation for Information Processing) Technical Committee for Computer Applications in Technology was formed to study presently available architectures for enterprise integration of manufacturing enterprises. The results of this study have recently been published in a number of forms. [Williams] The study presented a comparison of three architectures for CIM (CIM-OSA, GRAI-GIM, and the Purdue enterprise architecture), recommendations for completing each of the architectures for use as an integrating framework, and a taxonomy of the area of study which aids in the identification of problems which need to be overcome in the integration effort. While the scope of this effort is different from our own, the conclusion regarding the need for a methodology to construct an architecture is relevant.

The task force found that, since each company is different, the methodology for building a CIM system is critical. In the task group report, a methodology includes:

(1) creation of a reference model which shows in a global and generic way how to structure a project to create an integrated enterprise or subsection of the enterprise,

(2) one or more modeling formalisms to build up models to study and evaluate the reference model defined in (1),

(3) a structured approach for the program taking the existing system to a future system meeting the objectives. The structured approach must cover the life cycle of the project,

(4) performance evaluation criteria for evaluation from several points of view (such as economics, reliability, etc.).

2 Proposed Framework

We propose a framework for control architectures which organizes information about architectures in a way which is useful for the construction of an architecture. The framework may also be useful for the analysis and comparison of existing architectures, but these applications of the framework will not be discussed in this paper. The relationship of this framework to others will be discussed in Section 2.5.

Our framework is based upon the notions of *tier of architectural definition* (which we will usually shorten to *tier*) and *element of architectural definition*. These terms will be described in detail shortly.

An instance of the framework, filled out with details of a specific architecture is termed an *architectural complex*. Roughly speaking, an architectural complex consists of a set of populated tiers. Each tier contains all the elements of architectural definition, and the elements from one tier are closely related to elements of the same type in other tiers. An extremely abbreviated example of an architectural complex is provided in Section 2.4.

2.1 Preliminary Definitions

A class of situations in which an architecture is intended to be used is termed its *domain*. For example, an architecture might apply to the manufacture of discrete parts. The realization of an architecture in hardware and software will be called an *implementation* of the architecture.

The concepts used in an architecture which have specific meaning to the architecture will be referred to as *architectural units*. Architectural units are frequently defined by giving each one distinct functional characteristics, although this is not the only mode of definition. We shall refer to the realization of an architectural unit in an implementation as a *component* of the implementation.

An *atomic unit* of an architecture is an architectural unit which the architecture does not break down further into simpler architectural units. Atomic units are the fundamental building blocks of an architecture. An architecture typically specifies the functions and any formal interfaces of each atomic unit.

2.2 Tiers of Architectural Definition

Informally speaking, a tier of architectural definition is a set of the architectural units grouped together to provide distinctions within the definition of the architectural complex. For example, a tier of architectural definition can be made for each of the stages which transform the concepts of an architecture into an implementation. Each tier represents a set of consistent decisions about architectural units at that phase of design or implementation.

The set of tiers of architectural definition for a specific architecture form a partition of the architecture, that is, it has the following properties:

(1) Architectural units representing each of the elements of architectural definition (defined later in this document) are present in each tier,

(2) each architectural unit is present in exactly one tier.

In addition, the tiers have a sense of order and are interdependent, so that decisions made at lower tiers are consistent with decisions made at the higher ones.

Because of these properties, tiers of architectural definition may be used to define conformance classes for the architectural complex. To be in conformance with an architectural complex to a given tier, an implementation must conform to the

specifications of the given tier and all the higher tiers of the architectural complex. Different implementations typically may use the same upper tiers to some tier level of an architectural complex but require different tiers below that level. Two such implementations will be in the same conformance class, as defined by the lowest tier used by both. Being in the same conformance class may guarantee some degree of interoperability or compatibility.

A precise definition of the concept of tiers requires two additional concepts, relations and partial orderings. A *relation* defines an association between architectural units. Some relations can be used to induce an order upon the set of architectural units which they associate. Using these relations, it is possible to generate (partial) orderings[2] of an architectural complex that may be used to group architectural units into tiers.

2.2.1 Relations

Three types of relations which are of interest in control architectures are: *decomposition/aggregation*, *instantiation/classification*[3], and *specialization/generalization*. While relations other than these three types may occur, we are not aware of any that are applicable to control architectures.

Decomposition/aggregation relates an item to its parts. For example, a jigsaw puzzle is an aggregation of its pieces. Each piece of the puzzle is part of the decomposition of the puzzle. Any architectural unit which is not atomic is an aggregation.

As an example of manufacturing architectural units in a decomposition/aggregation relation, let us define a functional architectural unit called a controller. A controller performs all the functions necessary to operate a machine. We can decompose this functionality into two simpler architectural units, a planner (which performs any planning required to control the machine) and an executor (which performs any actions necessary to control the machine). The planner and executor are a decomposition of the controller architectural unit. Conversely, the controller contains an aggregation of the planner and executor.

Instantiation/classification relates a class to instantiations (examples) of the class. For example, the jigsaw puzzle on the corner of my dresser is an example (instantiation) of the jigsaw puzzle class.

As an example of manufacturing architectural units in an instantiation/classification relation, let us define an architectural unit which refers to any abstract concept in the factory and call it an entity. We can define any number of instances of an entity, at various levels of abstraction. We can define an architectural unit called a location, which refers to physical positions in a factory. In manufacturing, nested classifications

2. an ordering on a set is partial when not every element of the set participates in the ordering. This concept will be defined more precisely later in this document.
3. Two senses of classification are intended: (a) to define a new abstract class of which a thing at hand is an instance, (b) to find an existing abstract class of which a thing at hand is an instance.

are frequently used. The location which is on the floor, one foot west and east of the left door jamb is a specific location. This specific location is an instance of a location, which is an instance of an abstract entity.

Specialization/generalization relates two classes of items[4]. B is a specialization of A (or equivalently, A is a generalization of B), if every item in B is also in A. For example, jigsaw puzzles are a subclass of puzzle (other kinds include crossword puzzles, Chinese finger puzzles, etc.), and, conversely, the class of all puzzles is a generalization of the class of jigsaw puzzles.

As an example of architectural units in a generalization/specialization relation, consider that, one might define a general class of messages called "command." At a somewhat more concrete (specialized) level, "3-axis machining command" might be defined, and at an even more concrete level, "fly cut" might be a kind of 3-axis machining command.

2.2.2 Partial Orderings

Many relations (including all relations of the three types just defined) can be used to define a *partial ordering* for the set of architectural units in an architectural complex. In a partial ordering of a set, any two arbitrary set elements need not be related, but for any two which are, a sense of direction can be established for the relation. If A and B are architectural units, we may say that A is less than B (denoted $A < B$) or B is greater than A (denoted $B > A$) with respect to the relation. A partial ordering may connect a chain of architectural units, with the sense of direction being preserved through the entire chain. For example, if A, B and C are a chain of architectural units with respect to a partial ordering such that A is less than B, and B is less than C, it follows that A is also less than C.

We will require that the "<" symbol for a partial ordering have the following sense for relations of the three identified types:

For decomposition/aggregation relations, $A < B$ means A decomposes into B and other things.

For specialization/generalization relations, $A < B$ means B is a specialization of A.

For instantiation/classification relations, $A < B$ means B is an instance of A, or there is a C such that B is an instance of C, where $A < C$.

The partial orderings we allow have two restrictions beyond the normal mathematical definition: (1) we do not allow $A < A$, (2) the graph of the partial ordering must be a tree (i.e., if branches separate, they may not rejoin).

4. There is a relationship between classification/instantiation and specialization/generalization. Whenever a class has instances which are also classes, instantiation and specialization are indistinguishable.

2.2.3 Tiers

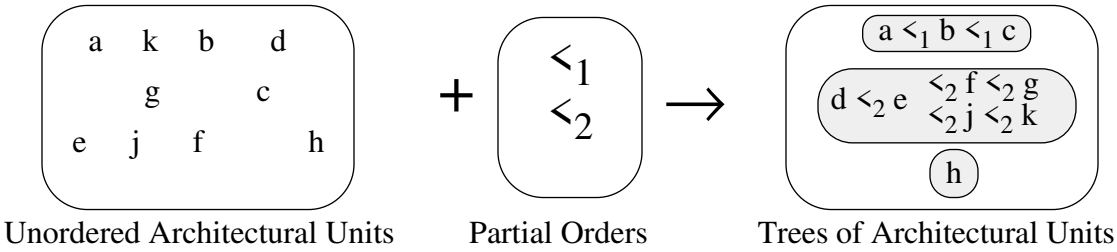
Within an architectural complex, several relations are usually chosen as part of the architecture. Each relation applies to a distinct set of architectural units (i.e. no architectural unit belongs to two such sets), and this set may be partially ordered with respect to the relation. Since the sets are distinct, we may consider each architectural unit to belong to at most one relation.

Using the *set* of partial orderings defined by the relations of an architectural complex, it is possible to divide the architectural units of an architectural complex into groups called *tiers of architectural definition* with respect to that set of relations. The tiers are totally ordered, and, for any relation used in generating the partial ordering $<_p$, if architectural unit A is in one tier, and architectural unit B is in a lower tier, it is not permissible that $B <_p A$. If only one relation is used in constructing the partial ordering (e.g., abstraction), the total ordering on the tiers will have an obvious meaning and name (e.g., abstraction). If more than one relation is used in defining the tiers, the total ordering of the tiers may have a different meaning from any of the relations used to construct the partial ordering. For example, we may order the architectural units in terms of abstraction or decomposition, but the total ordering induced on the tiers may be perceived as domain restriction. Moreover, the total ordering on the tiers may be defined by more than one relation. For example, several tiers of an architecture may be related by abstraction, while the others are related to these tiers and each other by domain restriction.

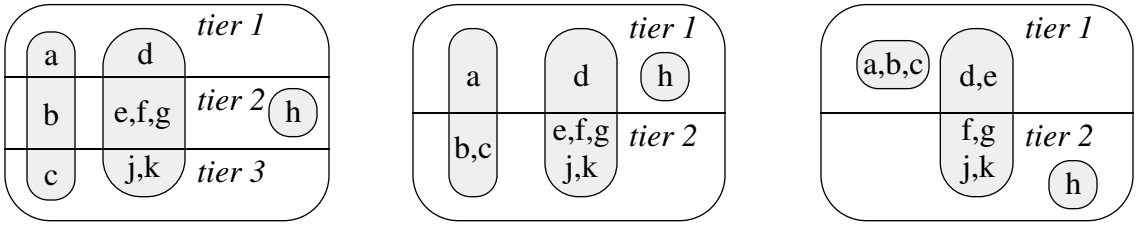
The grouping of architectural units into tiers must preserve the direction of each relation, as just described, but it is permissible for both A and B to be in the same tier with $B < A$. This differs from our previous thoughts on the subject [Kramer]. For each relation, the designers of an architectural complex are free to assign the architectural units of that relation to tiers independently from the methods used for other relations. If an architectural unit does not take part in any of the relations of the architectural complex, it may be placed in any tier.

Because the architectural units of any one relation may be assigned to tiers in several ways, and because the architectural units of different relations are assigned independently; given a set of relations, there is more than one way a set of tiers can be defined. As a trivial example, it is always possible to put all architectural units in a single tier. See Figure 1 for another, more interesting example. Hence, different architectures can use the same set of relations but still have different tiers of architectural definition.

In the remainder of the document, we will abbreviate the term tier of architectural definition to tier and omit the defining relation when it is clear which relation is intended.



Architectural units in an architectural complex can be organized into disjoint (i. e. disconnected) trees using a set of partial orderings generated by relations on the architectural units.



The architectural complex can then be partitioned into tiers in many different ways, preserving the order of the trees. Since the trees are disjoint, each tree can be grouped into different tiers as the designers see fit. Three ways of building tiers consistent with the given set of partial orderings are shown here. Trees are shown in shaded areas.

Figure 1: Defining Tiers in an Architecture Using Partial Orderings

2.3 Elements of Architectural Definition

The definition of an architectural complex requires a number of *elements of architectural definition*. These are:

(1) statement of scope and purpose

(2) domain analyses

(3) architectural specification

(4) methodology for architectural development

(5) conformance criteria

Elements of architectural definition are *conceptual* entities, which may or may not have any physical realization.

These elements of architectural definition vary in indispensability. For example, an architecture must have an architectural specification, but it is possible to use an architecture which omits conformance criteria. Definitions of existing architectures include different subsets of these elements of architectural definition and place emphasis on them in varying degrees. A completely defined architectural complex specifies each of these elements at each tier.

2.3.1 Statement of Scope and Purpose

The *statement of scope* of a tier describes the domain to which the tier is intended to be applied. It is useful to identify explicitly items which are out of scope and to identify general characteristics of the domain which may extend or limit the applicability of the tier to other domains. The scope of a tier is always larger than, or the same as the scope of the next lower tier.

A *statement of purpose* identifies what the objectives of the tier are within the given scope. The statement of purpose of a tier should be a major determinant of the contents of the tier. If the objective is to achieve interoperability between components of an implementation, it would be expected that interfaces between components and the definition of information shared between the components would be stressed. If the objective is to guarantee real-time performance of a conforming control system, models which describe the activities of the system may be stressed.

2.3.2 Domain Analyses

A critical step which must take place before an architecture can be formulated is to perform analyses of the target domain which reveal its essential characteristics. These analyses are *domain analyses*. The type of analyses done, the order in which the analyses are performed, and the language in which the results are expressed are part of the methodology for domain analysis. The results of the domain analyses may be very much different depending on the types of analysis performed and the analysis methodologies used.

Different frameworks suggest different forms of domain analysis. Functional analysis, information analysis, and dynamic analysis are commonly used forms of domain analysis [Jayaraman]. We will use these three in our subsequent discussion of architectural issues.

A *functional analysis* of a domain is an analysis of all the functions within the scope of the architecture which a conforming control system is supposed to be able to perform and the sequence and dependencies of the functions.

An *information analysis* of a domain is an analysis of all the information external to each atomic unit within the scope of the architecture needed for a conforming control system to function properly.

A *dynamic analysis* of a domain is an analysis of the characteristics of the functions and information in the domain which vary over time during control system operation. It provides qualitative and quantitative information about the timing, sequence, duration, and frequency of change in the application of the functions and information of the domain [Jayaraman]. Real-time requirements would be explored as part of the dynamic analysis.

2.3.3 Architectural Specifications

An *architectural specification* is a prescription of what the architectural units of a tier are, how they are connected (logically and physically), and how they interact. It should specify which of the architectural units are atomic and specify the composition of non-atomic units. The architectural specification should provide the definitions of any relations among the architectural units used in the architecture. The architectural specification forms the core of a tier; it is an essential ingredient.

The form of architectural specification varies widely. Often, natural language is used extensively, but formal methods may be used as well.

2.3.4 Methodology for Architectural Development

A set of procedures for refining and implementing an architecture is called the *methodology for architectural development* for the architecture (which we will shorten to *methodology* when the meaning is clear).

The architectural specification at each tier of architectural definition is related to, and used in, generation of an architectural specification for the other related tiers. The methodology for architectural development specifies how to go about building one tier from another.

A methodology may specify working top-down (from higher tiers to lower), bottom-up, or some combination of both in constructing the complete architectural complex. For example, if the code or specifications for the lowest tier is available, as is often the case when dealing with vendor-supplied equipment, an implementation-independent template for the code may be developed. In this case, the methodology would describe how to use the template.

A methodology for producing an architectural specification at a middle tier of architectural definition from a specification at a high tier of architectural definition might include:

(1) performing an activity analysis

(2) using a CASE (Computer-aided Software Engineering) tool embodying the high-tier specification to define domain-specific tasks, sensors, actuators, etc.

A methodology for producing an architectural specification at a low tier of architectural definition from a specification at a middle tier of architectural definition might include:

(1) rules for assignment of software modules to computing hardware

(2) rules for using computer language code templates

(3) timing analysis

(4) methods for making performance measurements

(5) debug mechanisms

If an architecture lacks a methodology for getting between any two tiers of architectural definition, control systems developers must devise their own methods for making the transition.

2.3.5 Conformance Criteria

Conformance criteria are standards which specify how an architectural unit at one tier of an architectural complex conforms to the architectural specifications of a higher tier, or how a process for building part of an architectural complex conforms to the development methodology given by the architectural complex for building that part.

A *conformance test* is a procedure that determines if conformance criteria have been met.

Conformance criteria and tests are needed for determining whether a control system actually implements the specifications an architectural complex.

We may define *conformance classes* of an architectural complex which identify sets of different and incompatible choices of architectural features. The advantage of defining conformance classes is the ability to have choices within the architecture, while allowing the bulk of the architecture to remain unchanged. As noted earlier, tiers provide a convenient method of defining conformance classes. Conformance classes could be defined within tiers, but it would seem preferable to split a tier into two tiers to avoid this.

2.4 An Example of an Architectural Complex

A full description of a useful architectural complex takes dozens to thousands of pages, so we have not attempted a full example. Table 1 shows an extremely abbreviated view of an architectural complex with three tiers. Each tier is shown in a row of the table. The elements of architectural definition are shown as columns of the table. This example corresponds closely to an architectural complex used in more applied work we have done. Tier 1 is a reference architecture for machine control, tier 2 is an application of the reference architecture to machining (metal cutting), and tier 3 is an application of tier 2 to a specific type of machining center. An architectural complex for a different specific machining center could have the same two top tiers as those shown in the table, with a different third tier.

Table 1: Example of an Architectural Complex

DOMAIN ANALYSES	ARCHITECTURAL SPECIFICATIONS	METHODOLOGY FOR ARCH. DEV.	CONFORMANCE CRITERIA
1. High-level machine tasks are decomposable into subtasks. 2. Most machine systems require closed loop control to operate effectively. 3. State tables provide a good method of describing tasks having discrete states.	1. Controllers are arranged in a hierarchy. 2. A controller executes tasks from a fixed set of parametric task types. 3. Information is to be modeled in EXPRESS. 4. Tasks are defined as subtypes of ALPS primitive_task.	1. Define parametric task types for the domain. 2. Assign a set of task types to each controller. 3. Write a document for each controller in the hierarchy describing the task types it can handle.	1. Are controllers defined in tier 2 arranged in a hierarchy? 2. Are task types defined in tier 2 provided for each controller? 3. Are the tier 2 information models written in EXPRESS?
1. Different cutter types are used for different high-level machining operations. 2. High-level machining operations (such as finish milling a pocket) may be decomposed into primitive machining operations. (such as straight line feeding).	1. High-level machining operations on features include rough_cutting, finish_cutting, counterboring, etc. 2. actual EXPRESS model gives relations among task types. 3. Controller hierarchy will include task, primitive and servo levels.	1. Write a state table for each task type involving discrete states. 2. Write a C++ function for each task type that involves continuous variables.	1. Is each tier 2 task implemented in tier3? 2. Is the tier3 code written in C++ ? 3. Are the relations among task types as implemented in tier 3 as described in the EXPRESS model?
1. Tool tip position must be controlled but workpiece rotation is not a concern. 2. 4-axis servoing (spindle is 4th axis) is required for tapping.	1. PC must run 50 MHz or faster. 2. PC CPU must be a Pentium. 3. float feed_rate; *(in tier 3, source code is part of the specifications).*	1. Revise definitions of task types in tier 2 if adequate performance cannot be attained. 2. Collect feedback from users to see if changes are needed.	1. Does the system use the required hardware? 2. Is the system running the correct software?

	TIER1	TIER2	TIER3
SCOPE AND PURPOSE	Scope: control of a wide variety of machine systems. Purpose: 1. Provide low-cost control systems. 2. Be able to develop control systems quickly.	Scope: control of 3-axis machining centers Purpose: make parts quickly while holding typical required tolerances.	Scope: control of XYZ Corp model ABC 3-axis machining centers Purpose: provide controller that runs on a PC.

2.5 Relationships to Other Frameworks

In specifying that all the elements of architectural definition be addressed at each tier, the proposed framework encourages a complete and grounded description of the architecture. The IFAC report supports the inclusion of domain analysis with formal modeling methods, specification of architectures, and the inclusion of a methodology for the application of the architectures.

As previously stressed, the proposed framework is designed specifically to facilitate the creation of an architecture. Many architectural proposals (CIM-OSA stands out in this regard) provide a multidimensional space with many cells (CIM-OSA has 36, for example) but do not give much guidance regarding development paths to follow through the space. By using tiers, a linear development path is evident, which may be followed top down or bottom up. The multidimensional spaces create the impression that different aspects of an architecture can be handled independently, but experience shows this is not generally true. Rather, decisions about one dimension interact with decisions about others. Tiers form checkpoints at which there is opportunity to verify that decisions on all dimensions are consistent.

Our framework is more generic than other frameworks in that it allows the designer of the architecture choice of which relations to consider.

The specialization/generalization relation is used to define dimensions in many frameworks. For example:

(1) In Zachman's metamodel, it is used in the definition of the implementation dimension, with associated levels of Scope, Enterprise Model, System Model, Technology Model, Detailed Description,

(2) In Biemans' metamodel, it is used in the definition of levels of abstraction,

(3) In Bohms' metamodel, it is used in the definition of the specification level,

(4) In the CIM-OSA metamodel, it is used in the definition of the dimension of architectural genericity, with associated levels of generic, partial and particular.

Some of the criteria for classification specified by other frameworks cannot be expressed as relations on the architectural units. The criterion of flexibility in Biemans' dimensions or the dimension of representation in Bohms' framework are examples. While each of these is an important characteristic of an architecture, the characteristic

is the result of variations in two or more dimensions which can be expressed as relations on the architecture. Variations in the flexibility of an architecture results from variations in the generality, scope and domain of an architecture. The dimension of representation captures variations in view, abstraction level and language level.

3 Issues in Developing Architectures

We turn now to issues which must be considered in developing an architecture. The issues given pertain to architectures in general, not just to control architectures[5]. Throughout this section, we will assume that architectures have the five elements of architectural definition presented in Section 2, but we will not assume that every architecture has explicit tiers. Where appropriate, we will relate our proposed framework to the issues.

3.1 Balance among Elements of Architectural Definition

Perhaps the most basic question to be decided when developing an architecture is: what should be the balance of emphasis in the architecture's treatment of the five elements of architectural definition.

Certainly all architectures must have a specification. And to develop a specification, some analysis of the domain must have been done, whether this analysis is formal or not. For the other elements, more variation is possible. In the literature, it is common to read about architectures which do not have an explicit scope or purpose, or which omit conformance criteria. Some architectures (see [Dornier1], [Dornier2], or [Quintero], for example) pay great attention to methodology for architectural development. Others (see [Martin1] through [Martin6], for example) do not discuss methodology at all. We believe all five elements should be defined at each tier, but the balance among elements may justifiably vary a great deal.

3.2 Issues Regarding the Scope and Purpose of an Architecture

Defining the scope of an architecture involves determining the degree to which the architecture, or part thereof, depends upon its context. It is possible to define context-free infrastructures (e.g., Common Object Request Broker Architecture [OMG]), but then it is not clear where it is appropriate to use them. It is not clear how context-free it is feasible for architectures to be. Much research on control architectures has emphasized defining architectures with as broad a scope as possible. Some researchers propose that there are aspects of control which are generic [Hatvany], but we propose that every specification of any value has some limit to its applicability.

In determining the broadest possible scope for an architecture, it is frequently necessary to look beyond the primary subject matter of the domain. Secondary characteristics of the domain (such as performance requirements, importance of safety, and need for resource sharing, among many others) are likely to be the determining characteristics.

5. A list of control architecture issues developed using the framework as a basis may be found in [Kramer].

It is not unusual for many domains to share relevant secondary characteristics. The classification of such characteristics is a challenge. To date, we have not seen such a classification.

The purpose of an architecture may also affect the scoping. For example, a control architecture which is designed to make it easy to build conforming implementations will be much different from one which guarantees that controllers built according to its specifications will be interoperable.

The authors propose that there are some generic aspects to control, but that, as the architectural specifications of an architecture become more detailed, the scope of the architecture will usually have to be narrowed. The concept of tiers provides a mechanism for permitting different parts of the architecture to have different scopes.

Natural language seems to be most suitable for the statement of scope and purpose. However, it may be helpful, in addition, to use an N-dimensional space spanning some large range and to identify a portion of the space as being within the scope of the architecture. The selection of axes for this N-dimensional space for the classification of architectural efforts has been one focus for both the work of the CIM-OSA project [Jorysz] and the work of ISO 184 SC5 WG1 [ISO1]. In earlier work, we examined the use of domain, life cycle, and organizational extent as dimensions of scope [Kramer].

3.3 Domain Analysis Issues

Whatever domain analyses are chosen to perform, the purpose and the scope of the architecture affect the content of the analysis which is performed. Moreover, domain analyses should be chosen which are compatible with the purpose of the architecture.

An architecture reflects the domain analyses upon which it was based. Domain analysis is a broad field, and an extensive literature search is beyond the scope of this paper. For more details and a bibliography, the reader is referred to [Prieto-Diaz].

In formulating an architecture, selecting what part of the domain should be analyzed, and the methodology for analysis are decisions with many ramifications. The first of these issues is discussed in Section 3.3.1, the second in Section 3.3.2.

3.3.1 Aspects Covered

As suggested in [Bohms], one dimension along which an architecture can be analyzed is aspects. This is distinct from scope. An *aspect* is a cross-cutting view of an architecture from some specialized viewpoint, such as information, communications, or control flow. Specifying a set of aspects from which to view the problem domain is essential in formulating an architecture, but often aspects must be inferred from the architectural specification, since they are not explicitly stated.

Existing architectures place varying amounts of emphasis on different aspects. As previously mentioned, an architecture tends to reflect the domain analysis aspects used. In Section 2 we mentioned functional analysis, information analysis, and dynamic analysis. The corresponding aspects are, simply, functional, information, and dynamic aspects. The two most widely accepted of these are functional aspects and information

aspects. The *functional aspects* of an architecture describe what a system conforming to the architecture does. A functional specification would describe what roles components could fill in the architecture and what functions each of these roles would encompass. The *information aspects* of an architecture describe the information required for the operation of an implementation of an architecture.

The relationship between analysis of functional aspects and analysis of information aspects is an open issue. Some methodologies insist that they are inextricably intertwined, whereas others view the two as separable.

An additional aspect which the authors have found to be important in their own work is the dynamic aspect of the architectures. The *dynamic aspects* of a control system describe how the information and exercise of functions vary over time. In the case of information, an examination is made of which architectural units need what information and when, and which architectural units create or change the information and when. The dynamic analysis of function looks at when architectural units perform each of their functions. Dynamic analysis includes examining whether the speed of the system is sufficient to meet the requirements of the application.

For each type of analysis, an appropriate representation must be found. The following sections discuss the representation of results for each of these types of analysis.

3.3.1.1 Functional Aspects

Frequently, the functions performed by a system are expressed only in the computer-executable language (C, C++, Ada, etc.) of the implementation. A different approach is to extract the required functions and express them in a more generic fashion. Such an extraction gives a functional analysis of the system.

Functional analyses may be stated in natural language or in a formal language. Examples of formal languages used for this purpose are Activity Scripting Language (AcSL) [Dornier2], Structured Analysis and Design Technique (SADT) [Ross], and IDEF0 [FIPS2]. State tables and petri nets [Tanenbaum] may also embody the results of functional analysis.

3.3.1.2 Information Aspects

As with the functions of a system, required information is often expressed only in data structures of the computer-executable languages (C, C++, Ada, etc.) of the implementation. A different approach is to develop conceptual models of the required information. A *conceptual information model* of a set of information is a description of the information, always giving relationships among the members of the set, usually including the data type of the members of the set, and often giving some of the semantic content of the information. A conceptual model is expressed in a formal *information modeling language* designed for this purpose, such as EXPRESS [ISO3], NIAM (Natural-Language Information Analysis Methodology) [Verheijen], and IDEF1 [FIPS1]. Some compilers exist which can translate a conceptual model into a specific computer language or a database schema.

These languages are suitable for defining items of information in an architecture such as part designs, tools descriptions, or process plans. They can be used for modeling other parts of an architecture but were not built for that purpose.

The existence of "domain-independent" information models for architectures is currently a topic for debate. Some efforts (such as STEP and CIM-OSA) have attempted to construct such models, while others confine themselves to the construction of models for more explicitly limited domains [Barkmeyer], [Fiala], [Wavering].

3.3.1.3 Dynamic Aspects.

The performance of a system over time is often one of the most poorly documented aspects of a system. An analysis of the dynamic aspects is most frequently done when the system fails to perform satisfactorily. It is, however, possible to build dynamic models for the system for design and analysis.

Examples of formal languages used for this purpose are IDEF2 [Mayer] and IDEF3 [Menzel]. IDEF2 produces a dynamic model appropriate for constructing simulations; IDEF3 produces a dynamic model which captures the behavioral aspects of the system.

Commercial tools exist (ObjecTime and StateMate, for example) which provide for building executable system models. As the model is executed, a graphical user interface lets the user see how states change, when different modules (which generally embody different functions) are active, and how information changes. These tools generally provide log files and utilities which capture and analyze this data. The output typically might be a table giving how many times a function was executed, how many times a message of a given type was sent, or how much time was consumed by each module.

3.3.2 Domain Analysis Methodologies

Determining a methodology for domain analysis involves two distinct decisions:

(1) determining which domain analyses should be used, and

(2) how the various analyses can be combined to give a picture of the domain sufficient for the purpose and scope of the architecture.

One frequently used set of analyses is the triple of functional, information, and dynamic analyses. This triple is supported by the IDEF languages[6] mentioned in the previous section. The associated methodology specifies that functional analysis is performed first, followed by information analysis, and finally, dynamic analysis.

A currently popular alternative is object-oriented analysis [Dewhurst1]. This technique mandates that function and information be analyzed simultaneously. This approach produces "objects" which have both information and operation content. The operations

6. There are many other languages which support each part of this triple. The IDEF languages were designed to support the triple together with an associated methodology.

are one degree removed from functions; the user is required to specify operations which will cause the system to perform the desired functions. Overall analysis of the dynamics of the system of objects created is not explicit in this methodology.

The Feature-Oriented Domain Analysis Method developed by the Software Engineering Institute [Kang] is another methodology which uses multiple types of analyses to develop an integrated domain analysis. The method is focused on promoting software reuse, and hence extracts the generalizable aspects of the domain. It incorporates the following analyses:

(1) Context analysis—setting the boundaries of the domain,

(2) Features analysis—determination of the external interfaces of the architectural units,

(3) Functional analysis—functional analysis as previously discussed in this paper and state table description of those architectural units which have states,

(4) Information analysis—information analysis as previously discussed in this paper.

Many other alternatives are available. It is unclear what the best methodology is, but the selection may well involve the domain, scope and purpose of the architecture and the availability of tools for assisting in the analysis.

3.4 Architectural Specification Issues

In creating the architectural specification, the two basic categories of issues are:

(1) what type of information the specification should include,

(2) how the specification should be described.

Within each of these categories, additional issues are discussed in the following sections.

3.4.1 Contents of Architectural Specification

In constructing an architectural specification, a key decision is the mode of specification of the architectural units. Frequently, the mode of specification is related to the methods used for domain analysis. For example, if separate informational and functional analyses are performed, it will be natural to describe architectural units as embodying sets of functions, which have certain inputs and outputs. If an object-oriented analysis of the domain has been made providing the operations on certain objects, it will be natural to describe architectural units as objects with associated behaviors. Once the mode(s) of definition of architectural units are determined, the types of architectural units must be determined and defined. If the number of types of atomic units is large, and each type of unit has few functions, the number of required interactions becomes large. In this case, defining the types, defining the interactions,

and understanding the dynamics becomes difficult. If atomic units have many functions and few types of atomic units are used, the overall architecture does not add much functionality above that of the atomic units. Clearly a balance must be maintained.

The way in which architectural units are defined depends on the purpose of the architecture, as well. If, for example, we are interested only in using the architecture to structure our knowledge about a situation, we may well give only a functional definition of the architectural units. If we are interested in providing an architecture which guarantees interoperability of the architectural units, additional interfacing specifications must be defined.

The relationships between the architectural units must also be defined. The designers of the architecture must decide which of these units are atomic and how these units can be combined to form other architectural units. At one end of the spectrum, it might be decided to select an algorithm for combining any number of arbitrary architectural units. In this case, it is not necessary to enumerate the allowed combinations. At the other end of the spectrum, an architecture might allow only specific combinations of architectural units to exist, i.e., it may enumerate all the combinations. An architecture can specify some combination of the two strategies for different sets of architectural units, or may devise a different strategy.

The designers of the architecture must decide which relations should be drawn between architectural units. For example, one might use the abstraction relation to describe which architectural units are specializations of each other, or domain restriction to describe which architectural units are valid in different domains.

The builders of an architecture must decide if they are going to use tiers in the construction of the architecture. If they do, they must determine what the most appropriate division is. If the builders have a clear idea of what conformance classes they wish to define, that may be used as a guideline for specifying tiers, since a tier can be used to define a conformance class, as observed earlier. For control systems, we believe that at least three tiers of an architectural complex should be used. Roughly speaking, tier 1 is a fairly general architecture, tier 2 is an engineering design for an application, and tier 3 is a description of the software and hardware of the application.

If an architecture defines its architectural units functionally, two large transitions from the abstract to the concrete will be encountered in implementing the architecture: first, going from a functional description of an architectural unit to a description of the interfaces and external behavior of the unit, and, second, assigning the interfaces and behaviors to (computer) executable processes. In both cases there is a wide range of choice of how to distribute the more abstract across the more concrete. In the first transition, specific functions (say, sensory processing or control execution) may be assigned to single modules or each function may be distributed as subfunctions across modules. In the second transition, several modules (maybe all of them) may be assigned to one computer process, or each module may be assigned to its own process, or (at the extreme) each module may distributed across several processes. Note that in the latter case, the architectural specification will have to be augmented. We have found that how modules are distributed across processes is critical in the more abstract tiers of an

architecture as well as the more concrete ones, because communications issues that arise in distributed systems percolate back up into the logic of how architectural units deal with one another.

3.4.2 Description of Architectural Specification

Most architectures are defined in natural language, but this is often vague. A degree of vagueness is appropriate at a high tier of architectural definition. In fact, several authors explicitly endorse vagueness. Unfortunately, it is often not clear what is vague by intent and what is vague inadvertently. The areas of intentional vagueness should be clearly defined. This is easy in formal modeling languages, and possible, but rarely done, in natural language.

Formal languages have several advantages over natural languages:

(1) formal languages are much clearer and less ambiguous;

(2) formal languages provide formal methods of linking tiers;

(3) models constructed in formal languages may be checked algorithmically for logical completeness and syntactic correctness; for some languages, compilers do these jobs automatically;

(4) when formal languages are used at a low tier of the architecture, compilers may be written which will produce executable computer code or database schemas automatically.

The elements of architectural definition are very different, so it is appropriate to use different formal languages for different elements. Formal languages for expressing analyses and architectural specifications have been mentioned already. We are not aware of any formal languages for expressing methodologies for architectural development.

When an architecture includes several tiers of architectural definition, it may be appropriate to use different languages for the same element of architectural definition at different tiers. For example, at the lowest tier, the architectural specification could be given in a standard computer language, while at the highest tier a formal modeling language may be suitable.

3.5 Methodology For Architectural Development Issues

As discussed in Section 2, one of the most important elements of architectural definition is a methodology for architectural development. A methodology tells how to build an architecture or how to apply the architecture to create an implementation. Methodologies for the development of control architectures have typically been adapted from software engineering methodologies. While a literature survey of software engineering methodologies is beyond the scope of this paper, we will mentioned three which are frequently used in connection with architectures.

One of the most widely used software engineering techniques is the *waterfall method*. In this methodology, requirements are developed which the system must satisfy, a design is made which satisfies the requirements, a detailed design is developed, and a system is developed according to the design. Parts of the system are tested and integrated, the system is then installed and must be maintained, as long as the system is in use [Conte]. It is important that the requirements and design phase be diligently performed, as correcting design mistakes is quite costly.

A popular alternative is the *spiral model* of software development [Boehm]. In this model one begins with requirements definition, develops designs, then prototypes. The prototypes are tested and the results fed back into a design phase where the initial design is refined or modified as required. This process may be repeated several times.

A variant of the spiral model which may be applied to control system development is to create a prototype implementation which demonstrates a vertical slice through all tiers of architectural definition such that only a narrow subset of the total intended capabilities of the control system is included in the slice. This results in a working control system with limited capabilities whose performance can be assessed. The lessons learned from the assessment are applied in revising the architecture. The revised architecture is used in the next turn around the spiral, at which time, a wider slice is included in the implementation. This methodology is commonly referred to as *cyclic development* [Quintero], [Senehi2].

Another useful methodology is the *transform methodology*. In this methodology, a formal specification of the desired product is made, and the specification is automatically transformed into code. An iterative loop can improve the performance of the code, leading to an evaluation of the product. An outer iterative loop may be made to change the specification based on changing requirements [Boehm]. Many CASE tools embody this methodology.

CASE tools can be of great help in developing architectures because many aspects of a methodology for architectural development can be built into a CASE tool. If this is done, using the CASE tool ensures that the methodology is followed. Thus the provision of CASE tools alleviates the problem of there being no formal languages for methodologies.

3.6 Conformance Issues

Conformance criteria and tests are defined in Section 2. In the sections below, conformance issues are discussed.

3.6.1 Conformance Testing Methods

To determine if an implementation of an architecture conforms to the architectures, tests must be devised. Methods for determining conformance might include reading source code, running documents that should be computer-processable through computers, observing an implementation in action and comparing its behavior with the behavior expected from a conforming implementation, devising test cases and using

them to test control systems, and requiring documentation of development activities. Conformance testing could also include establishment of an organization to do the testing.

3.6.2 Usefulness of Conformance Testing

The end user of a control system may want to be assured that a component is conformant with a particular architecture to ensure that it can be used with other conformant components previously installed or being acquired. Conformance tests can provide such assurance.

Conformance testing can be useful to the developers of an architecture in the context of evaluating the architecture. To evaluate an architecture, implementations of the architecture would have to be built. Each implementation would be a test of the architecture, provided that the implementation conforms to the architecture.

3.6.3 Testing Conformance in Development

If an architecture includes one or more methodologies for architectural development, the development process for building an implementation should use them. Using the methodologies is part of conforming to the architecture.

How can conformance to a methodology be tested? To the extent a methodology for architectural development is embodied in a CASE tool, conformance to a methodology may be obtained by ensuring the tool is used. If the methodology is supposed to produce specific documents (or other products) these can be examined.

3.6.4 Conformance Classes

The builders of an architecture must decide whether to define conformance classes and if so, how they should be defined. If it is anticipated that features of the specializations or implementations of an architecture will differ, defining conformance classes may be useful. The conformance classes will make it evident to potential users of the architectures where they share features and where they differ. As discussed earlier, tiers of an architectural complex provide ready-made conformance classes.

3.6.5 Conformance Metrics

In addition to the conformance class itself, which identifies the specifications to which conformance is required, there is the issue of degree of conformance, which concerns how close something is to conforming. If the degree of conformance is to be measured, a *conformance metric* which assesses how closely the implementation conforms to the architecture must be devised. This is commonly done by means of a checklist: an implementation identifies those specifications of the architecture to which it conforms and those to which it does not.

3.6.6 Non-Conformance

Once an implementation is built (or while it is being built), it is common that the implementation offers easy opportunities for improvement by making small changes contrary to the architectural specifications, so that the implementation is no longer fully in conformance. Typically, easy changes of this sort have a high hidden cost, in that they compromise the modularity of a control system, make its behavior less understandable, make it less portable, and make it harder to reuse, maintain, extend, and modify, some or all of which are likely to be undesirable to the system's users. Such changes may, however, have high value in terms of performance or initial cost. These factors should be considered in establishing conformance criteria, classes, and metrics.

3.6.7 Standards Issues

To be readily implementable, it helps if an architecture makes use of international, national or industrial standards. It is unreasonable to expect to find standards for all features of an architecture, but where standards are appropriate for the needs of the architecture, they should be used. Some features of an architecture may be covered by no standards or by developing standards. An architecture must specify which standards are required and which are not.

For developing standards there is an issue of suitability of the current state of the standard. The standard may not yet have a degree of maturity which the developers of an architecture need. In this case, it is possible to use the standard as much as possible and add the necessary enhancements to make it useful. When using a developing standard, there is an additional issue of when to upgrade from one version to another. Considerable cost may be involved in upgrades, so it is important to evaluate the stability of the version before switching to it.

4 Conclusion and Future Work

The authors have found the terminology and framework presented here to be useful in considering control architectures of all types. We put it forth with the hope that others will also find it useful and, perhaps, improve it.

We have used the terminology and framework to develop a set of issues which need to be addressed when constructing a control architecture. These issues have been used to compare two architectures developed at National Institute of Standards and Technology: the Manufacturing Systems Integration (MSI) architecture [Kramer], [Senehi1], and the Real-Time Control System (RCS) architecture [Albus1], [Albus2]. The issues and the results of the comparison are documented in [Kramer]. We are currently using the framework and issues in the construction of an architecture blending the features of both architectures [Senehi3].

References

[Albus1] Albus, James S.; *RCS: A Reference Model Architecture for Intelligent Control*; IEEE Journal on Computer Architectures for Intelligent Machines; May 1992; pp. 56 - 59

[Albus2] Albus, James S.; McCain, Harry G.; Lumia, Ronald; *NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)*; NIST Technical Note 1235, 1989 Edition; National Institute of Standards and Technology; April 1989

[Barkmeyer] Barkmeyer, Edward J.; Ray, Steven R.; Senehi, M. Kate; Wallace, Evan K.; Wallace, Sarah F.; *Manufacturing Systems Integration Information Models for Production Management*; National Institute of Standards and Technology Interagency Report; 1992; (forthcoming). (Available from the National Technical Information Service, Springfield, VA 22161.)

[Biemans] Biemans, Frank P. M.; Vissers, Chris A.; *A Systems Theoretic View of Computer Integrated Manufacturing*; Proceedings of CIMCON '90, NIST Special Publication 785; National Institute of Standards and Technology; May 1990; pp. 390 - 410

[Bittici] Bittici, Umit Sezer; *Computer Integrated Manufacturing Systems: Status and Trends;* Control and Dynamic Systems; Vol. 60; 1994; pp. 1-31.

[Boehm] Boehm, Barry W.; *A Spiral Model of Software Development and Enhancement*; Computer; IEEE; May 1988; pp. 61-72

[Bohms] Bohms, Michel; Tolman, Frits; *RIA: Reference Model for Industrial Automation*; Proceedings of CIMCON '90, NIST Special Publication 785; National Institute of Standards and Technology; May 1990; pp. 114 - 132

[CEN] CEN/CENELEC WG-ARC; *European Prestandard ENV 40 003 Framework for Enterprise Modelling*; CEN/CENELEC; Brussels; 1990

[Checkland] Checkland, Peter; *Systems Thinking, Systems Practice*; Chichester N.Y.; Wiley; 1991c

[Conte] Conte, S. D.; Dunsmore, H. E.; and Shen V. Y.; *Software Engineering Metrics and Models*; Benjamin/Cummings; Menlo Park, Ca., 1986.

[Dornier1] Dornier GmbH; E*valuation of Standards for Robot Control System Architectures - Final Report Executive Summary*; European Space Agency Contract Report; August 1990

[Dornier2] Dornier GmbH; *Baseline A&R Control Development Methodology Definition Report*; study managed by P. Putz; European Space Agency Contract Report; October 1991

[Fiala] Fiala, John; *Manipulator Servo Level Task Decomposition*; NIST Technical Note 1255; National Institute of Standards and Technology; October 1988

[FIPS1] Federal Information Processing Standard; *Integration Definition for Information Modeling, Extended (IDEF1-X)*; NIST Computer Science Laboratory; National Institute of Standards and Technology; December 1993

[FIPS2] Federal Information Processing Standard; *Integration Definition for Functional Modeling (IDEF0)*; NIST Computer Science Laboratory; National Institute of Standards and Technology; December 1993

[Hatvany] Hatvany, J.; *Intelligence and Cooperation in Heterarchic Manufacturing Systems*; Robotics and Computer-Integrated Manufacturing; Vol. 2, No. 2; 1985; pp. 101 - 104

[ISA] Instrument Society of America; *Glossary of Standard Computer Control System Technology, 2nd ed.*; Revised and updated by Theodore J. Williams; Research Triangle Park, N.C.; Instrument Society of America; c1985

[ISO1] ISO; *International Standard ISO/TR 10314-1 Industrial automation - Shop floor production - Part 1: Reference model for standardization and a methodology for identification of requirements*; ISO; Geneva; 1990

[ISO2] ISO TC184/SC5/WG1; *Document N-282 Version 3.0, Framework for Enterprise Modelling*; Working Draft; May 1993; (Available from National Electrical Manufacturers Association, 2101 L Street, N.W. Washington, D.C. 20037)

[ISO3] ISO; *International Standard ISO 10303 Industrial Automation - Product Data Representation and Exchange - Part 11: EXPRESS Language Reference Manual*; ISO; Geneva; 1994

[Jayaraman] Jayaraman, Sundaresan; *Design and Development of an Architecture for Computer-Integrated Manufacturing in the Apparel Industry Part I: Basic Concepts and Methodology Selection*; Textile Research Journal; Vol. 60, No. 5; May 1990; pp. 247 - 254

[Jorysz] Jorysz, H. R.; Vernadat, F. B.; *CIM-OSA Part 1: Total Enterprise Modelling and Function-View*; International Journal of Computer Integrated Manufacturing; Vol. 3, No. 3 and 4; 1990; pp. 144 - 156

[Kang] Kang, Kyo; Cohen, Sholom G.; Hess, James A.; Novak, William E.; Peterson, A. Spencer; *Feature-Oriented Domain Analysis Feasibility Study*; Technical Report CMU/SEI-90-TR-21, ESD-90-TR-222; Software Engineering Institute, Carnegie Mellon University; November 1990

[Kramer] Kramer, Thomas R.; and Senehi, M. K.; *Feasibility Study: Reference Architecture for Machine Control Systems Integration*; NISTIR 5297; National Institute of Standards and Technology; November 1993

[Martin1] Martin Marietta; *System*; Draft Volume I of Next Generation Workstation/ Machine Controller (NGC) Specification for an Open System Architecture Standard (SOSAS); Document No. NGC-0001-13-000-SYS; March 1992

[Martin2] Martin Marietta; *NGC Data*; Draft Volume II of Next Generation Workstation/ Machine Controller (NGC) Specification for an Open System Architecture Standard (SOSAS); Document No. NGC-0001-16-000-NGC DATA; March 1992

[Martin3] Martin Marietta; *Workstation Management Standardized Application (WMSA)*; Draft Volume III of Next Generation Workstation/Machine Controller (NGC) Specification for an Open System Architecture Standard (SOSAS); Document No. NGC-0001-14-000-WMSA; March 1992

[Martin4] Martin Marietta; *Workstation Planning Standardized Application*; Draft Volume IV of Next Generation Workstation/Machine Controller (NGC) Specification for an Open System Architecture Standard (SOSAS); Document No. NGC-0001-14-000-WPSA; March 1992

[Martin5] Martin Marietta; *Controls Standardized Application (CSA)*; Draft Volume V of Next Generation Workstation/Machine Controller (NGC) Specification for an Open System Architecture Standard (SOSAS); Document No. NGC-0001-14-000-CSA; March 1992

[Martin6] Martin Marietta; *Sensor/Effector Standardized Application (SESA)*; Draft Volume VI of Next Generation Workstation/Machine Controller (NGC) Specification for an Open System Architecture Standard (SOSAS); Document No. NGC-0001-14-000-SESA; March 1992

[Mayer] Mayer, R.J.; Painter, M.; *IDEF Family of Methods*; Technical Report; Knowledge Based Systems, Inc.; College Station, TX; 1991 (Available from Knowledge Based Systems, Inc. 1408 University Drive East, College Station, TX 77840-2335)

[Menzel] Menzel, C.P.; Mayer, R.J.; Edwards, D.; *IDEF3 Process Descriptions and Their Semantics*; Kuziak, A., and Dagli, C., editors; Knowledge Base Systems in Designing and Manufacturing; Chapman Publishing; 1990

[OMG] Object Management Group; *The Common Object Request Broker: Architecture and Specification*; OMG Document Number 91.12.1, Revision 1.1; December 1991. (Available from ftp.omg.org)

[Prieto-Diaz] Prieto-Diaz, Ruben; Arango, Guillermo; *Domain Analysis and Software Systems Modeling: IEEE Computer Society Press Tutorial*; IEEE Computer Society Press; IEEE; 1991

[Quintero] Quintero, Richard; Barbera, Anthony J.; *A Real-Time Control System Methodology for Developing Intelligent Control Systems*; NISTIR 4936; National Institute of Standards and Technology; October 1992

[Ross] Ross, Douglas T.; *Structured Analysis (SA), A Language for Communicating Ideas*; IEEE Transactions on Software Engineering; Volume SE-3, No. 1; January 1977; pp. 16-34

[Sastrón] Sastrón, F.; *Techniques in CIM Open System Architecture*; Control and Dynamic Systems; Vol. 60; 1994; 87-121

[Senehi1] Senehi, M. K.; Barkmeyer, Edward J.; Luce, Mark E.; Ray, Steven R.; Wallace, Evan K.; Wallace, Sarah F.; *Manufacturing Systems Integration Initial Architecture Document*; NISTIR 4682; National Institute of Standards and Technology; September 1991

[Senehi2] Senehi, M.K.; Wallace, Sarah; Luce, Mark E.; *An Architecture for Manufacturing Systems Integration*; Proceedings of ASME Manufacturing International Conference; Dallas, Texas; April 1992

[Senehi3] Senehi, M. K.; Kramer, Thomas R.; Michaloski, John; Quintero, Richard; Ray, Steven R.; Rippey, William G.; Wallace, Sarah F.; *Reference Architecture for Machine Control Systems Integration: Interim Report*; NISTIR 5517; National Institute of Standards and Technology; October 1994

[Sowa] Sowa, J. F.; Zachman, J. A.; *Extending and Formalizing the Framework for Information Systems Architecture;* IBM Systems Journal; Vol. 31; No. 3; 1992; pp. 590-616

[Tanenbaum] Tanenbaum, Andrew S.; Computer Networks, Second Edition; Prentice Hall; 1988; pp. 245-253.

[Verheijen] Verheijen, G.M.A.; VanBekkum, J.; *NIAM: An Information Analysis Method*; Information Systems Design Methodologies: A Comparative Review; North-Holland; 1982; pp. 538-589

[Wavering] Wavering, Albert J.; *Manipulator Primitive Level Task Decomposition*; NIST Technical Note 1256; National Institute of Standards and Technology; October 1988

[Williams] Williams, T. J. et al.; *Architectures for Integrating Manufacturing Activities and Enterprises*; Computers in Industry; Vol. 24; 1994; pp. 111-139.

[Zachman] Zachman, J. A.; *A Framework for Information Systems Architecture;* IBM Systems Journal; Vol. 26; No. 3; 1987; pp. 276-292.