Features of the Systems Engineering Process Activities (SEPA) Methodology

K. Suzanne Barber, Thomas J. Graser, Stephen R. Jernigan, and Brian J. McGiverin

The Laboratory for Intelligent Processes and Systems
The University of Texas at Austin
Austin, TX 78712
barber@mail.utexas.edu

Abstract

The presence of software systems within the manufacturing enterprise is significant and growing. The cost and complexity of software components must be controlled to maintain competitive operations. SEPA is a design methodology that creates traceable, comprehensible, and extensible component-based system design specifications based on requirements from system clients and domain experts. Through the application of Artificial Intelligence techniques, the SEPA tool suite presents itself as a unique tool offering among software development tools.

1. Motivation

The presence of software systems within the manufacturing enterprise is significant and growing. Software is integral to factory floor operations: from inventory management systems, production planning and scheduling systems to shop and machine control. According to Gary Gettel, Director of Factory Integration at Sematech, software continues to play an ever-increasing role in manufacturing:

> **Software content in equipment is growing by more than 25% per year. To prevent this increased amount of software complexity from derailing effective factory operation, more reliable, predictable fail-safe software will be required. Higher utilization of commercial software sub-systems, greater maturity of the industry's software development capability and reduced software customization through more configurable architectures will be needed.** (Gettel 1998)

The cost of installing and customizing commercial-off-the-shelf (COTS) software, developing in-house software systems, and integrating new software systems with existing systems can be enormous. In semiconductor manufacturing, for example, the cost of integration is often 3-10 times the basic cost of the manufacturing system product (Weber 1998).

The difficulty in achieving manufacturing component integration is influenced by a number of factors, including ever-increasing process/factory complexity and the interaction of multiple perspectives (e.g. technology, business, personal) (Weber 1998). Furthermore, it is deceptively easy to underestimate the cost of software maintenance when planning a software budget. An often cited study by Schach places maintenance at 67% of total lifecycle costs, while the requirements and specification phases account for only 7% (Schach 1990). The premise of the research described in this paper is: Investment in formal, repeatable requirements analysis and verification will reap rewards in later phases in the lifecycle, specifically reduction in maintenance costs and support for integration of system components.

2. System Engineering Process Activities

SEPA is a design methodology that creates traceable, comprehensible, and extensible system design specifications based on requirements from system clients and domain experts (Barber, Graser et al. 1998). The funnel abstraction is chosen (see Figure 1) to represent a spectrum of user inputs/requirements that are narrowed, refined, and structured into a system design.

As with many domains, software developers and integrators in the manufacturing domain aim to address a number of commonly recognized software engineering issues, including changing requirements, communication among stakeholders, adaptable architectures, and integration of COTS and in-house solutions. The SEPA methodology and tool suite focus on supporting such goals by providing:

1. defined deliverables to improve communication among stakeholders;
2. the use of multiple views on a variety of graphical knowledge models developed directly from knowledge acquisition;
3. a process for merging the knowledge models resulting from different domain experts yielding a unified set of user requirements;
4. a method for distinguishing between requirements relating to a specific system implementation and those relating to general domain knowledge;
5. support for requirements changes during development;
6. the designation of adaptable domain components based on responsibilities for services and tasks. The resulting reference architecture represents the domain

independent of implementation, allowing it to be used for a family of applications in the domain;

7. traceability and verification throughout the analysis and design process; and

8. early detection of component integration issues through a domain-based reference architecture

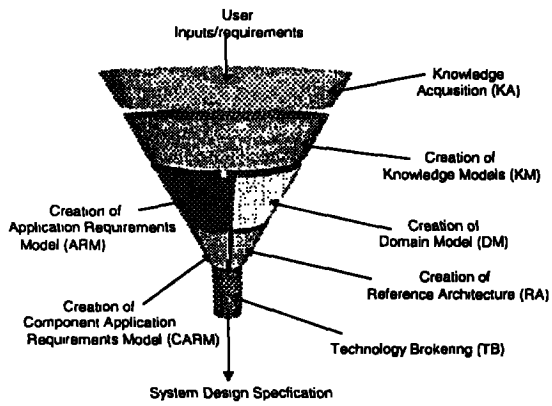

Figure 1: Systems Engineering Process Activities Funnel

representation that includes integration rules describing the constraints and dependencies between components.

The SEPA methodology emphasizes the separation of user requirements for a particular application from the knowledge applicable to the general domain. Whether elicited simultaneously or independently, an application cannot be created without gathering information about the domain as a whole along with the requirements of the specific application.

During **Knowledge Modeling**, Knowledge Engineers employ "knowledge models" (e.g., message sequence charts, task descriptions, etc.) to graphically depict and document knowledge acquired from domain experts and promote verification and validation feedback cycles. A single KA session may result in several new KMs.

The **Domain Model** (DM) is a unified homogenous model. The representation for the DM contains more information than can be viewed at any one time, resulting in multiple views. These views are often graphical and usually extract relevant model details. During the KM and DM stages, the knowledge engineer repetitively refines and structures the domain information.

Concurrent with the extraction of domain requirements during Domain Model development, the Knowledge Engineer (KE) also extracts application requirements to populate the **Application Requirements Model (ARM).** During the KA process, Knowledge Engineers (KE) are typically presented with two distinct types of information from the Domain Expert (DE): the domain-specific and the application-specific. In an ideal situation, the knowledge engineer would have separate KA sessions with the domain expert for each of these types of information. However, the domain expert does not typically have this abstracted view of his work and may find it difficult to provide information to the KE in this manner. The preferred approach for SEPA is to let domain experts explain their requirements in the context of scenarios which relate to the entire domain, current project, or past projects. This information is captured in Knowledge Models, which necessarily contain both domain-specific and application-specific information. The translation from the Knowledge Models to the Domain Model only preserves the domain-specific information.

A **Reference Architecture** (RA) is a repository of domain components reusable in a "family" of domain applications. A component is an object-oriented class consisting of (1) attributes and services, (2) behavior, and (3) the set of constraints and dependencies between itself and other components. A single component may be realized by one or more actual objects (or components) during implementation.

The SEPA reference architecture must be completely domain-specific and be highly flexible for building similar systems in the future. This flexibility is achieved because the components are described in terms of "what" they do, which is less dynamic over time compared to "how" they do it. As a result, these component definitions can outlive technology solutions that were available during the analysis and design activities.

The requirements are originally captured in knowledge models, which are then translated into Application Requirement Templates (ARTs). An ART details the application requirements in terms of classification, source, target, and intention. Before the application requirements are applied to the RA, they are first translated into the **Component Application Requirements Model** (CARM) which describes these requirements in the "component language" of the RA.

Technology Brokering (TB) constructs a **System Design Specification** by mapping between available technology solutions and RA components. Knowledge in the CARM and relationships to the RA components guide decisions in selecting "how" a domain service in the RA can be satisfied by technology solutions in a particular application. The solutions are chosen based on any number of design trade-off concerns (e.g. cost, availability, ease of implementation, etc.).

3. SEPA Tools

The following section outlines the tool suite designed to support the SEPA methodology. Gathering, managing and refining knowledge is a significant part of the total effort in the development of large, complex software systems. While tool support cannot fully replace decisions and contributions provided by system clients, users, integrators, and developers; it can assist personnel in managing the large quantity of information associated with a development effort. Furthermore, tool support can guide personnel in maintaining traceability, documenting rationale for decisions, identifying inconsistencies, and applying evaluation metrics. Figure 2 shows the SEPA tool suite overlaid on the SEPA activities funnel. Subsection 3.1 briefly outlines the implementation approach used by each tool Subsections 3.2 through 3.5 describe the SEPA suite tools and hi-light the knowledge representations and AI techniques they employ.

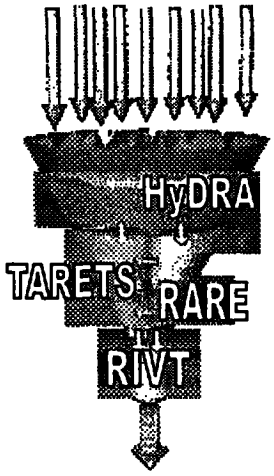

Figure 2. SEPA Tool Suite

3.1. SEPA Tool Implementation Approach

To support such runtime objectives as third party tool interoperability and access by many users in large development projects, the SEPA tool suite implementation uses a CORBA backbone accessible via web-based clients. Figure 3 presents a high-level view of general implementation approach followed by each of the SEPA tools. The architecture includes an interface client process (implemented in Java and accessed through a web browser), a set of SEPA backend services with persistent storage, and a LISP-based reasoning service. The processes run on separate platforms communicating via CORBA. Interface Definition Language (IDL) descriptions of SEPA tools, assist in incorporating third-party tool interactions. (Orfali, Harkey, Edwards. 1998)

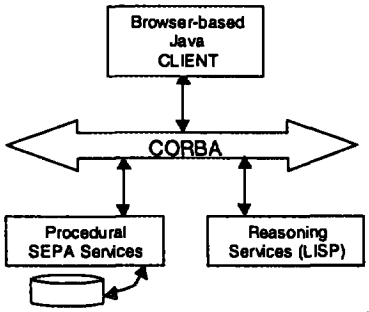

Figure 3 - SEPA Tool Architecture

3.2. Hybrid Domain Representation Archive (HyDRA)

The Hybrid Domain Representation Archive (HyDRA) focuses on knowledge modeling and the translation of the Knowledge Models (KMs) to the Domain Model (DM). HyDRA's objectives are to:

- aid the knowledge engineer by providing tool support for knowledge acquisition and modeling. The tool includes document management functions (e.g. versioning, access control, change logs, etc.) in addition to intelligent reasoning functions to guide the user in model creation and unification.
- automate the transition from unstructured, incomplete requirements to formal, complete, and consistent requirements.

The following section provides an overview of the HyDRA tool and is followed by a more detailed look at HyDRA's knowledge representation issues.

3.2.1 HyDRA tool overview

The benefits of formal requirements to designers are unquestionable. However, clients are not likely to know a formal specification language. The "requirements gap" that ensues typically results in implementation and maintenance cost overruns, rework, and delay. HyDRA provides a semi-automated facility for iterative requirements refinement. The translation process used in requirement refinement identifies inconsistencies and incompleteness in the KMs. The process continues by abstracting away the application requirements and synthesizing the KMs into a complete, consistent DM. It further helps to document rationale for KM and DM creation as well as to evaluate propagation effects of changes on requirements in the future.

During translation, the user guides the application of heuristic rules and corrects default rules where needed. Rule applications and user corrections are cached for documentation and future re-application of the translation process. Traceability is preserved across the translation and assists in the definition and validation of requirements from multiple knowledge sources. HyDRA provides "back verification" of modifications to the Domain Model against the original Knowledge Models and records decisions where the KE wishes the requirements captured in the Domain Model to deviate from information in a Knowledge Model.

The examples shown in Figures 4 and 5 show a portion of two knowledge models that HyDRA could merge. These knowledge models are representative of those that would be derived from knowledge acquisition with experts from an electromechanical assembly plant. When HyDRA attempts to merge these representations, it will detect the inconsistency between the process orderings (i.e., Is "Resin Encapsulation" before or after the creation of sub-assembly 6?). HyDRA asks the user how to resolve the inconsistency and records this decision as a "rationale at issue" point.

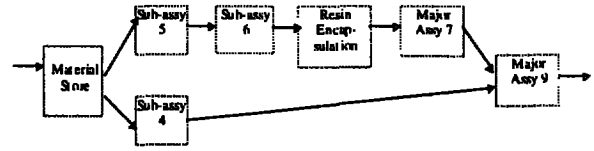

Figure 4 - Portion of an example Materials Flow Knowledge Model

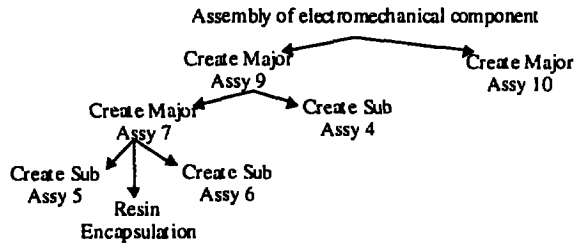

Figure 5 - Portion of an example Assembly Task Hierarchy Knowledge Model

3.2.2 HyDRA Knowledge Representation Issues

Design of the HyDRA's knowledge representations focused on enabling the features identified in Section 2. This section details how the SEPA methodology features impact HyDRA's knowledge representations.

First, to support communication among system development participants, HyDRA has well defined deliverables. Since HyDRA has been implemented in a CORBA client/server architecture, the objects that together makeup HyDRA's knowledge representation necessarily have their interface defined in the Interface Definition Language (IDL). IDL definitions for the knowledge representations ease the interaction of automated tools by decoupling the implementation details of each tool component from the other tool components and other tools (e.g., current SEPA tool suite development involves both Java and LISP components).

One of SEPA's main tenets is the emphasis on an extended requirements gathering and analysis phase. The implementation responsibility for this tenet rests more so on HyDRA than other SEPA suite tools. When transitioning from KA transcripts to KMs, the KM notations used to describe basic concepts must be similar to the notations used for those concepts in the KA transcripts. The similarity in notations lessens the possibility for errors as knowledge engineers, who may lack experience in the domain, attempt to translate the notations. Therefore, a wide variety of graphical notations is required and the set of required notations changes as the project progresses. That is, representations with higher fidelity are required to capture the fine details that are uncovered late in the requirements gathering process.

Until recently, no clear standard "family" of graphical notations was agreed upon. With the proliferation of object-oriented methodologies, several more complete notations exist for describing systems but they assume an underlying object-oriented semantics. The Unified Modeling Language (UML) [Rational, 1997 #121] is the most successful example of such a notation. Unfortunately, no corresponding standard exists for representing information at a point when objects have not yet been identified. HyDRA incorporates several of the independent notations associated with non-object-oriented methodologies (e.g., task hierarchies, data flows). In addition, HyDRA leverages the popularity of the UML by borrowing its notations where possible, yet remaining cognizant of the fact that UML-defined semantics are often object-based. For instance, provided the activity diagrams are drawn without swimlanes, they are sufficiently non-object-oriented to be useful as a notation in HyDRA. Just as with the UML, HyDRA's multiple representations allow for an extended analysis that progresses through stages with different representational needs.[1]

Another difficulty faced in borrowing from standardized notations is over-formalization. That is, the formalism of graphical notations, such as it is, is still too constrictive when representing early analysis information that is plagued with inconsistencies and incompleteness. The analysis can not be completed instantaneously, but delay between knowledge acquisition and the modeling of the elicited information threatens validity. The evolution of requirements occurs on every development effort and contributes to cost overruns and delays. Therefore, the SEPA methodology and HyDRA have been designed to encourage the capture of incomplete and inconsistent information till such a time when these problems can be resolved. Specifically, HyDRA relaxes the syntax of the notations, increases the expressiveness of the individual notations, and provides a means of checking syntax and semantics when the appropriate level of detail is available.

These changes in the notations often require the underlying representation to be specialized. In the case of the UML activity diagram, the notation retains its similarities to the standard but the underlying representation is quite different. Heterogeneous representations are problematic when the information contained in the various representations is inconsistent. This case occurs often in requirements analysis because the requirements are gathered from a diverse set of users. Each of these users may have a different set of terminology, a different level of abstraction, and possibly different processes for accomplishing the same tasks. *HyDRA's main research contribution is the merging of heterogeneous representations with possibly inconsistent information into a single representation that contains a unified, consistent model of the domain.* The unified representation has more restrictions on syntax and more formality than the individual knowledge model representations. The translation process is a semi-automated, iterative application of heuristic rules that is guided by the knowledge engineer. During the translation process, the knowledge engineer may be asked to help in the resolution of inconsistencies.

In accordance with the SEPA methodology, all of the SEPA tools maintain the ability to trace artifacts back through the process used to derive them. HyDRA establishes the base of this chain by maintaining links from each model back to the knowledge acquisition where the information was originally elicited and the domain expert who stated the requirement. This necessitates the inclusion of these links on each model and the creation of new links when new models are synthesized or derived from existing

[1] The modular architecture for representations has the added benefit of allowing HyDRA to be applicable to a variety of domains with varying special representational needs (e.g. representations with support for advanced geometric reasoning for manufacturing).

models. Furthermore, since SEPA's open architecture may involve third-party tools, this link mechanism must include a stable, external reference that can be used to refer to the target at any later date. Careful management of this external link is required when versioning is also incorporated.

While verification is most often thought of in terms of implementation compliance, the SEPA methodology applies some form of verification to every activity. In the analysis phase, formal verification of compliance testing is not realistic given the form and completeness of the information. Instead, SEPA encourages the return of each knowledge acquisition transcript and early knowledge model back to the associated domain expert for approval. This feedback gives the domain expert an opportunity to rethink his or her answers and correct misunderstandings. The HyDRA tool assists in this process by maintaining this verification information as a series of changes and digital signatures indicating domain expert signoff. The state of each model proceeds through a cycle (i.e., new, verified, modified, reverified, ...).

Models that represent information gathered from multiple domain experts present a unique verification problem. It may be impossible for any one expert to verify the entire model due to his or her limited perspective and level of abstraction. On the other hand, allowing each expert to verify a portion of a model can lead to models that are only partially verified or models that are completely piecewise verified yet still do not form a valid whole. HyDRA's approach is to couple the verification of single-expert knowledge models with strict traceability links to provide a degree of verification on models derived through the synthesis process. As mentioned above, the synthesis process may require additional input from the knowledge engineer to resolve inconsistencies. This input is recorded as a "rationale at issue" point and included as part of the verification for the derived model.

Future work will expand HyDRA's reasoning capabilities and address project progress estimations through statistical analysis on the current set of models. Metrics may include the amount of model integrated in unified representations, the amount of flux in models, the number of syntax errors, the average age of verified models, etc.

3.3. Reference Architecture Representation Environment (RARE)

The Reference Architecture Representation Environment (RARE) guides the transition from the functional DM generated by HyDRA to a component-based Reference Architecture (RA). RARE provides support for capturing decision rationale and tracing RA components back to elements of the DM. To measure RA quality, RARE uses domain independent metrics quantifying the achievement of domain design principles.

3.3.1 Representing Reference Architecture Components

The manufacturing domain requires a modular, adaptive software architecture that can both accommodate new technologies as they become available and accurately capture domain knowledge. Thus, the architecture's representation must strike a balance between the detail required to describe manufacturing processes and the degree of abstraction required for technology independence.

Object-oriented (OO) knowledge representations, particularly focused on extensibility and reuse are typical desirable qualities in a modular Reference Architecture. OO approaches define classes which present interfaces describing "what" the class can provide, hiding the implementation details.

RARE assigns responsibilities to components based on domain tasks. Task resources (or service dependencies) determine the data (or services) required (or provided) by a component. For example, for a *Car* component to provide a *Drive* service, it requires the *DeliverFuel* service provided by a *Carburetor* component. A component's behavior is defined by the services it provides, constraints on those services, and transitions between states resulting from service execution.

To represent this information, components in the RARE Reference Architecture are represented by three categories of information (Graser 1996):

Declarative Model (DM) - defines the attributes contained and the services offered by a component.

Behavioral Model (BM) - defines the states of a component, the transitions between those states, and the events which affect transitions.

Integration Model (IM) - defines the constraints and dependencies between components described by rules of composition.

Components in a domain are assigned individual responsibilities and, through mutual cooperation, achieve system goals that satisfy domain requirements. This type of cooperation leads to dependencies between components. The integration model represents these dependencies, categorizing them as either "static" or "dynamic." Static dependencies define a consumer/supplier relationship between components. Dynamic dependencies are rules that restrict the set of allowable configurations in which a component instance can participate. For example, for a *Car* component to deliver a specified power, it requires a *Carburetor* component able to *DeliverFuel* at some minimum rate. While all *Carburetor* components provide this service, the ability to deliver fuel at the needed rate depends on the specific *Carburetor* instance. These rules are used by RIVT during application design to determine if respective components meet the requirements of those dependent components requiring data or services in an integrated system (See Section 3.5).

3.3.2 "Searching" for a Satisfactory Reference Architecture

Given the set of all possible architectures that fulfill the requirements represented in the Domain Model, there may exist many viable architectures. There is often no single "right" architecture. RARE approaches the Reference Architecture creation process from the following perspective – given a prioritized set of qualities desired in the Reference Architecture, there is a near optimal Reference Architecture to be "found." Figure 6 characterizes the Reference Architecture derivation "search space." The search begins with a complete Domain Model and a "null" Reference Architecture. The

depth of the search space represents the degree to which the Reference Architecture covers the information represented in the Domain Model (i.e. completeness, to the extent the Domain Model accurately represents the domain). The breadth of the search space represents the multitude of structuring and abstraction options (e.g., concepts X and Y could be abstracted into component A or defined directly as components X and Y) available to the architect given a particular set of desired qualities and domain information from the Domain Model. The end objectives are to sufficiently cover domain information and to satisfy the quality goals set forth by the architect (Graser 1998).

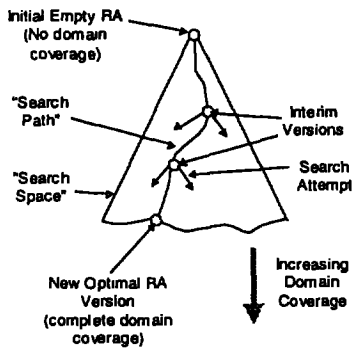

Figure 6 - Reference Architecture "Search Space"

As is typical with many applications involving search, the traversal of a branch in a search path may need to be abandoned if it produces unsatisfactory results; pruning branches and reducing the search space mitigate this problem and backtrack if necessary.

To navigate the search space shown in Figure 6, RARE systematically employs the following concepts: (i) *Goals:* High-level qualities determined by the architect to be important for the RA to exhibit in this domain, such as extensibility, comprehensibility, and maintainability; (ii) *Heuristics:* "Rules of thumb" compiled from expert experience on past projects which assist the architect in making rational decisions in defining RA components; and (iii) *Metrics:* Measurements of particular RA characteristics that indicate whether the architect adhered to respective heuristics.

3.4. Tool for Application Requirements Extraction and Technology Specification (TARETS)

The Tool for Application Requirements Extraction and Technology Specification (TARETS) extracts the application-specific data from the knowledge models for use in the design of the RA and the implementation of a System Design Specification. This extraction process is guided by a set of heuristics that help requirements engineers locate the application requirements within the knowledge models. Once identified, TARETS uses that information to populate an Application Requirements Template (ART). Each ART is an atomic element, describing a single application requirement, and divided into four major sections:

Type - a hierarchical classification of the requirement,

Sources - who or what generated the requirement,

Targets - model elements impacted by this requirement, and

Intention - justification and other descriptive information.

The Sources and Targets sections refer to elements found in the Knowledge Model via "semantic links". As a result, a key responsibility of TARETS during the Creation of the Application Requirements Model (ARM) is to monitor the evolution of the Knowledge Models as they are synthesized into the Domain Model to keep the links valid. For the user, this means that as concepts, tasks, and other model elements are being merged, modified, and renamed by HyDRA, the domain terminology used to describe the application requirements will be updated accordingly.

This monitoring and updating process also occurs during the Creation of the Component Application Requirements Model (CARM), due to the analogous process of building the Reference Architecture from the information found in the Domain Model. Describing the application requirements from a component perspective is necessary for RIVT to index the CARM and insure technology instances satisfy appropriate application requirements.

3.5. Requirements Integration and Verification Tool (RIVT)

The components defined in the SEPA Reference Architecture are domain-based – that is, they are designed to be technology independent. RIVT aids the developer in realizing system designs by performing Technology Brokering based on application requirements from TARETS and domain requirements represented in RARE components. This entails matching RA components to available technologies, insuring that those technologies satisfy the application requirements, and integrating those technologies into a complete system configuration.

Technology Brokering produces a System Design Specification that describes the chosen technology solutions (specifications of existing technology offerings are stored as RA component instances in a repository) and how they inter-operate to satisfy the requirements of the system. If RIVT is unable to locate a satisfactory technology solution during the brokering process, the tool can generate a "notional" component. After the System Design Specification is approved, the specification of this notional component can be exported and sent to technology providers as a "Request To Build".

As component instances are selected from the repository and configured to form a System Design Specification, RIVT's integration engine ensures interdependency requirements (represented in RARE's component Integration Model) are not violated. Furthermore, the set of interdependency rules fired, messages issued, and evaluation results are captured in a log. The log then serves as a rationale to support the System Design Specification.

4. Conclusion

The Systems Engineering Processing Activities (SEPA) methodology being developed at the University of Texas at Austin in the Laboratory for Intelligent Processes and Systems (LIPS) seeks to improve the systems engineering process by providing of a comprehensive, object-oriented development methodology and a suite of supporting tools. Today's manufacturing systems are critically dependent on the development and maintenance of software systems, including installing and customizing COTS solutions as well as developing in-house solutions. For this reason, the control of software development and maintenance costs has a direct impact on the control of manufacturing process costs.

An overall emphasis of the SEPA methodology is to provide a domain-based approach that expands reuse by enabling the development of multiple system designs and implementations. SEPA also assists in the communication between client and developer during the requirements gathering process.

Recognizing that many modeling methodologies do not adequately support early analysis efforts, SEPA emphasizes earlier activities to provide a sound foundation for component derivation, reuse, and integration. Issues inherent to software development for manufacturing and SEPA's approach for addressing them include:

- *Managing requirements in a complex domain:* SEPA aids in managing complex domain and system requirements throughout the development process. Traceability is an integral part of the SEPA tool suite, linking requirements as they evolve from their original informal representation to a structured, component-based representation.
- *Multiple perspectives:* HyDRA's knowledge representation scheme supports multiple domain perspectives and guides the knowledge engineer in resolving these perspectives into a synthesized Domain Model.
- *Knowledge capture in an increasingly advancing domain:* The increase and refinement of domain knowledge is inevitable, and a considerable amount of effort is spent merging this information into existing knowledge. The knowledge representation schemes in HyDRA and subsequent SEPA tools are designed to support the capture and evolution of requirements over time. HyDRA assists in merging new or additional information into existing domain knowledge and SEPA's traceability features allow this information to be reflected in Reference Architecture components and system designs.
- *Higher utilization of components and sub-systems:* RIVT aids the developer in realizing system designs by performing Technology Brokering based on application (system) requirements from TARETS and domain requirements represented in RARE components. Greater component reuse is achieved because requirements for a specific system are represented separately from requirements inherent to the domain. Furthermore, components are represented in a technology-independent manner, allowing their definition to endure changes in technology.
- *Reduced customization through more configurable architectures:* Based on user-established architecture goals, RARE assists the system architect in transitioning from a functional-based Domain Model to a component-based Reference Architecture.
- *Integrating COTS systems:* RIVT allows the designer to compare system design options using commercial technologies registered as instantiations of domain components. Furthermore, because SEPA Reference Architecture components are represented independent of implementation, commercial technologies can be registered as instantiations of domain components without modifying the Reference Architecture.

Through the comprehensive application of Software Engineering methods and Artificial Intelligence techniques (such as Knowledge Representation and symbolic search and reasoning), the SEPA tool suite presents itself as a unique tool offering among software development tools. Furthermore, it supports the needs of the manufacturing domain by providing a more mature software development capability.

5. References

Barber, K. S., T. J. Graser, et al. (1998). The Systems Engineering Process Activities: The Methodology and Supporting Tools. Austin, TX, The University of Texas at Austin.

Gettel, G. M. (1998). "Future Factory Level Issues and Needs," "Silicon, Software, and Smart Machines: Manufacturing Integration in the Semiconductor Industry", June 18-19, 1998, Austin, TX.

Graser, T. J. (1996). Reference Architecture Representation Environment (RARE) - A Reference Architecture Archive Promoting Component Reuse and Model Interaction, Masters Thesis, Electrical and Computer Engineering. Austin, TX, The University of Texas at Austin.

Graser, T. J. (1998). Reference Architecture Representation Environment (RARE) - Systematic Derivation of Object-Oriented Systems, Ph.D. Proposal, Electrical and Computer Engineering. Austin, TX, The University of Texas at Austin.

Orfali, R., Harkey, D., and Edwards, J. (1997). Instant CORBA, John Wiley & Sons, Inc., New York.

Schach, S. (1990). Software Engineering, Asken Associates.

Weber, A. (1998). "Advanced Process Control Framework: A Case Study in Open Integration Technology," "Silicon, Software, and Smart Machines: Manufacturing Integration in the Semiconductor Industry", June 18-19, 1998, Austin, TX.