The Systems Engineering Process Activities (SEPA) Methodology and Tool Suite

K. Suzanne Barber, Thomas Graser, Paul Grisham, Stephen Jernigan, Sutirtha Bhattacharya
The Laboratory for Intelligent Processes and Systems
Electrical and Computer Engineering, The University of Texas, Austin, TX 78712
barber@mail.utexas.edu

Research Objective and Motivation

The Systems Engineering Process Activities (SEPA) Program delivers a documented formal methodology and tool suite supporting traceable system analysis and design activities for development of modular, reusable software systems. The SEPA tool suite demands efficient knowledge representations as well as efficient search techniques to manage and interrogate the large, complex associations of artifacts generated in the software engineering process.

The analysis and design of large, complex software systems mandates a formal methodology and supporting tools to assist system development teams throughout the software system lifecycle. The magnitude of personnel, the diversity of personnel backgrounds and agendas, and the transient nature of personnel and technology in relation of the software system lifecycle places a number of requirements on the process by which 1) application domain requirements are acquired, analyzed and model, 2) a system architecture is derived from those requirements, 3) technology decisions are made and implementation progresses, and 4) the system is tested and maintained. A formal methodology provides a *gameplan* for the entire lifecycle keeping team members "on the same page" and offering a mechanism by which to gauge progress. Large software projects with large numbers of personnel making large numbers of decisions require not only a formal process but tools to assist team members in executing their jobs and documenting their decisions. Traceability of decisions and documentation of decision rationale is key to understanding resulting decisions as well as understanding the impact of changes to decisions (e.g. decisions related to modeling, design, implementation, test and maintenance). Current object-oriented design approaches provide minimal guidance to software engineers working to derive an object-oriented design from a functionally specified domain model. Both a formal process methodology and tools are required to assist the engineer in answering the question, *what are the objects (software modules)?* and the required specification of those modules for maximum reuse and ease of integration.

Research Approach

The SEPA effort proposes both a methodology and supporting tool suite to facilitate capture of evolving requirements and object-oriented design for the graceful realization and evolution of systems targeting particular applications. SEPA creates traceable, comprehensible, and extensible system design specifications based on requirements from system clients and domain experts. The funnel, or cone, abstraction is chosen to represent a spectrum of user inputs/requirements that are narrowed, refined, and structured into a system design. User inputs require refinement for a number of reasons, including the need to: (1) merge inputs from multiple sources, and (2) distinguish between inputs relating to system requirements and those relating to general domain knowledge.

Three tenets are key to the SEPA methodology:

- Traceability between artifacts generated at each SEPA phase is critical to assuring 1) accurate modeling and subsequent understanding of the domain and system requirements, 2) derived architecture satisfies and captures requirements, and 3) justifiable and documented rationale for the architecture and design. Artifacts are shown in above boxes at the end of arrows.
- Separation of systems requirements for a particular implementation from those requirements inherent to the general domain
- Reuse of requirements, architecture specifications and designs are all three necessary to evaluate the similarities of requirements across development efforts and subsequently assess the appropriateness of architecture components and designs.

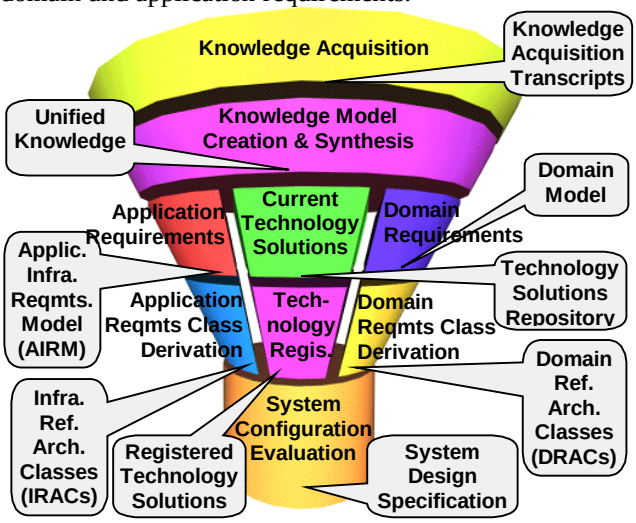
During **Knowledge Model Creation and Synthesis**, Knowledge Engineers employ *knowledge models* (e.g., message sequence charts, task descriptions) to graphically depict and document knowledge acquired from Domain Experts and promote verification and validation feedback cycles. Each Knowledge Model (KM) is the result of a single KA session. However, a single KA session may result in several new KMs. The KM synthesis effort seeks to detect conflicts and similarities among KMs across a particular user perspective or across the entire domain.

Knowledge Models reflecting user requirements often describe service and data requirements inherent to the domain and system implementation constraints/ requirements for a targeted application (**Application Requirements**) as well as example **Current Technologies**

Solutions. Separation of requirements and technology specifications promote a separation of concerns for developers as well as a reusable Domain Model which can "live" beyond technology choices to be used for multiple application system implementations where different technology choices are appropriate.

A **Reference Architecture (RA)** is described as a repository of domain components (known as Domain Reference Architecture Classes) which can be reused in a family of applications in a domain. DRACs are special domain classes that are defined at a business entity level of abstraction. A single DRAC may be realized by one or more actual objects during implementation. DRACs are represented by the set of: (1) attributes and services that characterize them, (2) the behavior of the DRAC, (3) the constraints and dependencies between the DRAC being described and (4) other DRACs. These DRACs are defined from model elements in the Domain Model. The resulting RA is completely domain-specific and highly flexible for building similar systems in the future. The translation process from the DM to the RA involves the methodical querying and analyzing of various views in the domain model (concepts, tasks, etc.) to arrive at DRACs with assigned responsibilities such that all responsibilities/ requirements identified in the DM have been satisfied. DRACs are technology independent, focusing on what domain requirements exist rather than how technology solutions satisfy those domain requirements. System constraints specifying how available or potential technology solutions must fulfil domain requirements are captured in the **Application Infrastructure Requirements Model** (AIRM). Following the object oriented decomposition of Reference Architecture, the AIRM is likewise represented in a OO representation (Infrastructure Reference Architecture Classes, IRACs) with links from the IRACs to the applicable DRACs.

KAM manages and monitors artifacts from Knowledge Acquisition (KA) to establish a baseline for traceability. KAM also provide project management facilities to assist in the coordination of personnel (knowledge engineers, developers, domain experts) as well as the documents produced or obtained during the KA process. **HyDRA** aids in creating Knowledge Models (KMs), detecting conflicts between KMs and synthesizing the KMs into a single, functional Unified Knowledge Model. The synthesis process merges domain information from multiple experts while preserving traceability to KA. **SEPArator** distinguishes the information present in the SEPA Unified Knowledge Model into a Domain Model containing domain-based requirements, a Technology Solutions Repository capturing legacy systems, and an Application Requirements Model containing requirements for specific system implementations. **RARE** guides the developer in transitioning from a functional Domain Model produced by HyDRA to an object-oriented Reference Architecture of domain-based components (DRACs). Primarily, RARE assists in the decision process regarding what entities in the domain model should be represented as DRACs in the reference architecture. **MeTER** complements RARE by modeling application requirements gathered from KA and represented in the Unified Knowledge Model to represent IRACs and respective instances. ASERT is used to specify application technology. These technology specification models are linked to DRACs and IRACs for subsequent use in generating a system design specification satisfying domain and application requirements.



RIVT provides a design configuration and trade-off analysis environment to 1) select intended portions of the RA (domain services and data requirements) to be included in the targeted system, 2) select technologies satisfying/delivering requirements posed by selected DRACs and corresponding constraints in the CARM, and 3) evaluate feasibility of integrating selection technologies. As components are integrated, relevant integration rules describing application performance requirements and DRAC interaction (data and service dependencies between DRACs) will trigger. The resulting transcript of rules triggered as well as actions taken by the user to satisfy any detected integration violation (e.g. selection of another technology or relaxation of constraints) provides a validation of system integrity and a trace-ability to user requirements. Allowing the specification of notional technology specifications, users can specify "what if" designs.

Demonstration of the SEPA tool suite will show the need and utilization of rich knowledge representations as well as efficient search techniques to support artifact capture, artifact associations and traceability, progression and evolution of artifacts, and interrogation to support analysis and design decisions. SEPA has been applied to a number of domains. The SEPA tool demonstrations show how the SEPA tools were used in a DARPA initiative to develop information systems support first responders in emergency situations (e.g. fire, natural disaster, ChemBio).