REQUIREMENTS EVOLUTION AND REUSE USING THE SYSTEMS ENGINEERING PROCESS ACTIVITIES (SEPA)

K. Suzanne Barber, Thomas J. Graser,
Paul S. Grisham, Stephen R. Jernigan
The Laboratory for Intelligent Processes and Systems
The University of Texas at Austin
Austin, TX 78712
voice: 512-471-6152 fax: 512-471-3316
barber@mail.utexas.edu

ABSTRACT

As more organizations attempt to reuse previous development efforts and incorporate legacy systems, typical software development activities have transitioned from unique ground-up coding efforts to the integration of new code, legacy code, and COTS implementations. This transition has brought on a whole new set of development issues, including resolving mismatches between integrated components and tracing legacy and COTS components to requirements. This paper presents the Systems Engineering Process Activities (SEPA) methodology, developed to address these and other issues in current software development practices. SEPA aids the reuse and integration process by focusing on requirements integration and evolution, while maintaining traceability to requirements gathered from domain experts and end users. The SEPA methodology supports the development process by promoting requirements analysis prior to design, separation of domain-based and application-based (i.e. implementation-specific) requirements, and evaluating system component suitability in terms of domain and application requirements. The paper also presents an example illustrating the application of SEPA in the emergency incident response domain to facilitate requirements management and foster requirements reuse.[2]

INTRODUCTION

Despite the introduction of software development methodologies and CASE tools to help alleviate the "Software Crisis," the development and maintenance of software is still too expensive and error prone. Effective reuse from previous software development efforts has the potential for increasing software productivity and quality an order of magnitude (Caldieri, Gianluigi, & Basili, 1991). However, current methodologies *do not emphasize the reuse of artifacts other than code (analysis or design)* from previous implementations. While code reuse typically occurs only at lower-level system design artifacts, analysis and design reuse often results in whole collections of related artifacts being reused (Department of Defense, 1996). Methodologies can further increase odds for reuse by identifying potential reuse at all system levels (e.g. system, component, code function) and among all participating elements (e.g. new development, legacy systems, and commercial off the shelf (COTS) components).

Reuse of requirements provides a number of benefits, including the following:

1. *motivation for selection of components:* Requirements guide the selection of optimal components for reuse. When requirements are transferred between development efforts, the rationale behind the original component selection decision is made available to the system designer.
2. *context for reuse decisions:* Requirements trace back to information gathered from domain experts and system users. Requirement-based reuse decisions are set in the context of domain processes or specific implementation needs.
3. *parametric constraints:* Requirements come in many forms, including parametric constraints (i.e. the system delivered must run at speed x) as well as general guidelines (e.g. the system's interface should be user friendly) and domain tasks and processes. Parametric constraints allow a static evaluation to narrow the field of available components.

The leveraging of legacy systems that were not designed with reuse in mind comprises a large part of the reuse motivation. However, the reusability of old systems is often difficult to evaluate by direct inspection. Perhaps the rework effort can be determined through code evaluation and system documentation. However, it can be difficult to determine the relative adherence of the old implementation to newly gathered requirements. When originally under development, requirements were likely gathered and analyzed for satisfaction by the old implementation. Given that a requirements effort has already occurred, an alternative is to determine which of the old requirements can transfer to satisfy new requirements rather than take on a fresh evaluation of the old system.

[2] This research was supported in part by the Texas Higher Education Coordinating Board Advanced Technology Program (ATP #003658452) and the Defense Advanced Research Projects Agency (DARPA).

Section 0 provides a model of requirements evolution and a list of features which are strongly desired in a methodology intending to support requirements evolution and reuse. Section 0 introduces the System Engineering Process Activities (SEPA) methodology and positions its distinguishing characteristics among these features. Section 0 provides an example illustrating the application of SEPA in the incident response domain.[3] Section 0 concludes with a discussion highlighting selected SEPA contributions to requirements evolution and reuse evident in the example.

REQUIREMENTS EVOLUTION AND NECESSARY METHODOLOGICAL FEATURES

In this section, we discuss some essential features of good software engineering methodologies to support requirements evolution and reuse, with an emphasis on activities during requirements gathering and analysis.
Figure 6 depicts the requirements evolution scenario and the traceability to implementations under a comprehensive development methodology. *Old Requirements* captured, analyzed, and verified are eventually linked to components in the implementation, *Old Implementation.* As *New Requirements* are captured, analyzed, and verified, they are mapped to *Old Requirements*, thereby indirectly referencing components from the *Old Implementation.* The mapping from *New Requirements* to *Old Requirements* may not cover the complete set of *New Requirements* and some of the *Old Requirements* may not be relevant in the *New Requirements.* As the *Old Implementation* may only partially address the *New Requirements*, there is a need for evolving (modifying) the *Old Implementation.* The remaining *New Requirements* spawn development of new components (*New Implementation*), which when integrated with components from the *Old Implementation*, result in a *Delivered Product.*

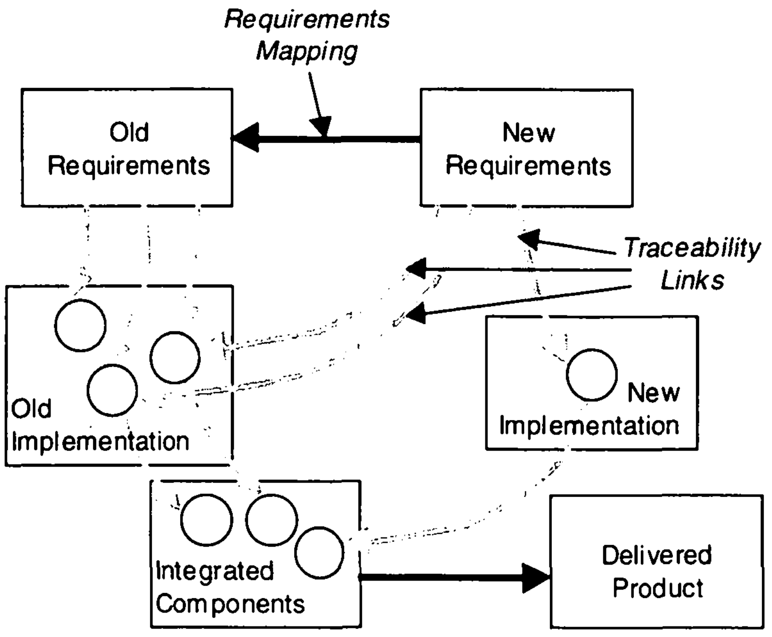


Figure 6: Requirements Evolution and Traceability

The difficulty in implementing the model in Figure 6 lies in matching requirements to implementations. To effectively render this mapping, implementations must be expressed in a meta-specification composed of the same language as the requirements themselves. Furthermore, this meta-specification should operate at an abstraction level that allows it to be independent of implementation specifics (e.g. Component A requires an integer value while Component B provides a float value), yet be expressive enough to capture a broad range of requirements (e.g. component must perform domain Task X on Operating System Y with Throughput Z). Defining this meta-specification at the appropriate level of abstraction along with careful management of *Traceability Links*[4] from requirements to implementations will encourage the reuse of whole collections of related artifacts from analysis and design.
Given the requirements evolution process outlined above, the development cycle becomes an issue of integration on two levels:

[3] Detailed discussions of the operations of SEPA tools are beyond the scope of this paper.

[4] Traceability Links provide a two way path between artifacts and symbolize the refinement or translation of information. For instance, a requirement would be connected to design elements that help to fulfill the requirement. The two way path permits navigation (i) to other artifacts from which a particular artifact is derived and (ii) to other artifacts derived from a particular artifact.

1. mapping, evolving, and integrating requirements previously gathered, analyzed, and verified into new requirements and
2. integrating technologies inferred by the traceability links from integrated requirements.

Figure 7 illustrates this two-level integration problem. Requirements are derived from an array of sources: a previous development effort, available media (e.g., system documentation), multiple domain experts, and multiple system users. Requirements elicited from multiple sources often overlap and possibly contradict. Yet understanding requirements involves incorporating multiple viewpoints, involving the capture, analysis, and resolution of many ideas, perspectives, and relationships at varying levels of detail (Kotonya & Sommerville, 1997). Once integrated, requirements guide the selection of appropriate COTS components, legacy systems, and newly developed implementations. Multiple system configurations may satisfy the same set of requirements, requiring further tradeoff studies for evaluation. Resulting selections are then integrated into a cohesive solution

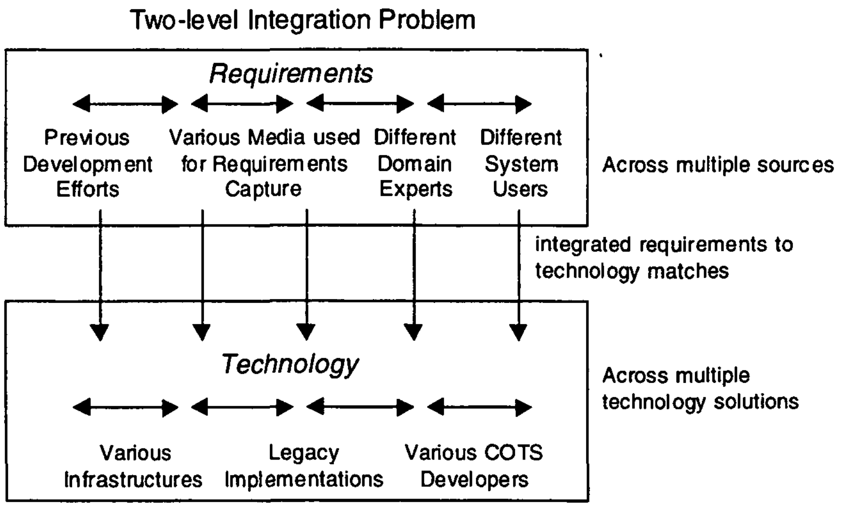

Two-level Integration Problem
Requirements
Previous Development Efforts
Various Media used for Requirements Capture
Different Domain Experts
Different System Users
Across multiple sources
integrated requirements to technology matches
Technology
Various Infrastructures
Legacy Implementations
Various COTS Developers
Across multiple technology solutions

Figure 7: Two-level Integration Problem

Opportunities for successful reuse, as suggested by the model in Figure 6, are increased through the use of a methodology that supports the following six features.

- ***Traceability:*** Throughout the requirements lifecycle, emphasis should be placed on maintaining accurate traceability. In (Gotel & Finkelstein, 1994), Gotel reported the results of a survey of more than 100 software engineering practitioners that indicated "most of the problems attributed to poor requirements traceability were found to be due to the lack of (or inadequate) pre-requirements specification (pre-RS) traceability." Contrary to the motivation of these results, they noted that existing tools mainly supported post-RS traceability only. Moreover, Palmer notes that current tools provide little support for identifying and maintaining traceability information (Palmer, 1997). Methodologies must provide the emphasis for the development of tools that support the identification and evolution of traceability information from requirements elicitation to requirements maintenance.
- ***Support for Multiple Viewpoints:*** The recognition of 'viewpoints' in software engineering is first credited to the CORE methodology (Mullery, 1979) in the late 70s. Since that time, it has become widely accepted that a complete requirements gathering effort must seek out the individual needs of the various stakeholders that will interact with the system to be developed (Sommerville, Sawyer, & Viller, 1998). Often the requirements gathered from these 'viewpoints' will overlap and possible conflict. Compromises are made in every development process to arrive at a single set of requirements for the project. These compromises may be implicit, un-documented, and performed without the input from the original stakeholders. Methodologies that hope to enable reuse of requirements and derivative artifacts (e.g., code) must accurately identify the source of these requirements (see Traceability above). This need requires that the methodology acknowledge the existence of multiple viewpoints and guide the viewpoint reconciliation process.
- ***Computational Requirements Representation:*** Typical requirements management approaches rely on tracing text fragments to design and development artifacts (Chipware, 1999; QSS, 1998; Technologies, 1999). This natural language representation supports only text string searches and provides little ability for structured requirements types and complex relationships among various requirements types (Palmer, 1997). The end result is a greater difficulty associating applications features to originally stated requirements.

- ***Separation between Domain and Application Requirements:*** To improve chances for reuse, software methodologies often suggest that "what" a system must do should be modeled independently of "how" the system should be implemented. The presumption is that "what" is done in a domain changes far less often than "how" it is accomplished, especially given frequent changes in technology (Tracz, 1991; Tracz, 1993; Tracz, 1995). Despite this advice, other requirements management tools do not explicitly recognize this separation.
- ***Comprehensive Process Support for Requirements Refinement and Management:*** Numerous requirements management tools are available for requirements analysis and refinement, taking different approaches to requirements representation (Chipware, 1999; QSS, 1998; Technologies, 1999). Even more tools support object-oriented design after requirements have already been elicited, validated, and merged (SES, 1999; Verilog, 1997). Some tool suites attempt to bridge requirements identification with subsequent object-oriented analysis and design (Rational, 1998). However, no current tools adequately support the requirements refinement process with automated support for traceability maintenance from elicitation through maintenance (Gotel & Finkelstein, 1994; Palmer, 1997). Without this integration, the transition from requirements capture, to requirements synthesis, to class derivation, to system design becomes excessively difficult. In the absence of a comprehensive process support for requirements refinement and management, some requirements may be ignored while others may not be maintained in the long-term after implementation or as the domain scope changes.
- ***Support for Requirements Volatility:*** Requirements change even during the initial development of a system. Changes may be promoted by more accurate requirements elicitation, changing user needs, changes in the relative importance of various user's requirements, and technology changes. Dobson et. al. (Dobson, Blyth, Chudge, & Sterns, 1992) note that requirements can also be driven by the downstream development process. Failure of the methodology to support changing requirements will result in requirements that are "incomplete, inconsistent with the new situation, and potentially unusable" (Christel & Kang, 1992). The prospect that requirements might be used (reused) beyond the current development effort underscores the need for the methodology to recognize and support requirements volatility throughout the project's lifecycle.

In this section, a model for requirements evolution was presented and six necessary methodological features were identified. In the following section, a methodology is proposed that emphasizes these features so that the requirement evolution and reuse model can be realized.

THE SEPA METHODOLOGY

The Systems Engineering Process Activities (SEPA) Methodology builds upon the DSSA approach (Tracz, 1991) and popular object-oriented methodologies by distinguishing between domain functional requirements and implementation-specific requirements, defining a domain architecture, identifying applicable technology solutions, and finally utilizing the domain architecture and technology solution specifications to assist reuse evaluations and design trade-off decisions. The domain architecture represents an object-oriented partitioning of domain functional and data requirements without designating a particular implementation. To realize an actual system implementation, the system designer must select appropriate technology solutions that satisfy both domain requirements and application implementation-specific requirements.

In this section, activities that comprise the SEPA methodology are discussed. SEPA activities are described by the SEPA funnel structure as shown in Figure 8, emphasizing the continuous gathering, narrowing, and refinement of user requirements into a component-based system design specification. Requirements require refinement for a number of reasons, including the need to: *(i)* merge input from multiple sources, *(ii)* discard irrelevant information, and *(iii)* distinguish between inputs relating to system implementation requirements and those relating to general domain knowledge. The SEPA methodology encourages commitment to requirements gathering and subsequent analysis, emphasizing: (i) requirements analysis prior to design, (ii) the separation of domain and application requirements, and (iii) requirements analysis for component-based development. Although the funnel illustration suggests a one-directional flow of knowledge, SEPA is a highly iterative design process, where new knowledge is continuously added to the top of the funnel through knowledge acquisition from multiple viewpoints: end-users, domain experts, system integrators, developers, etc.

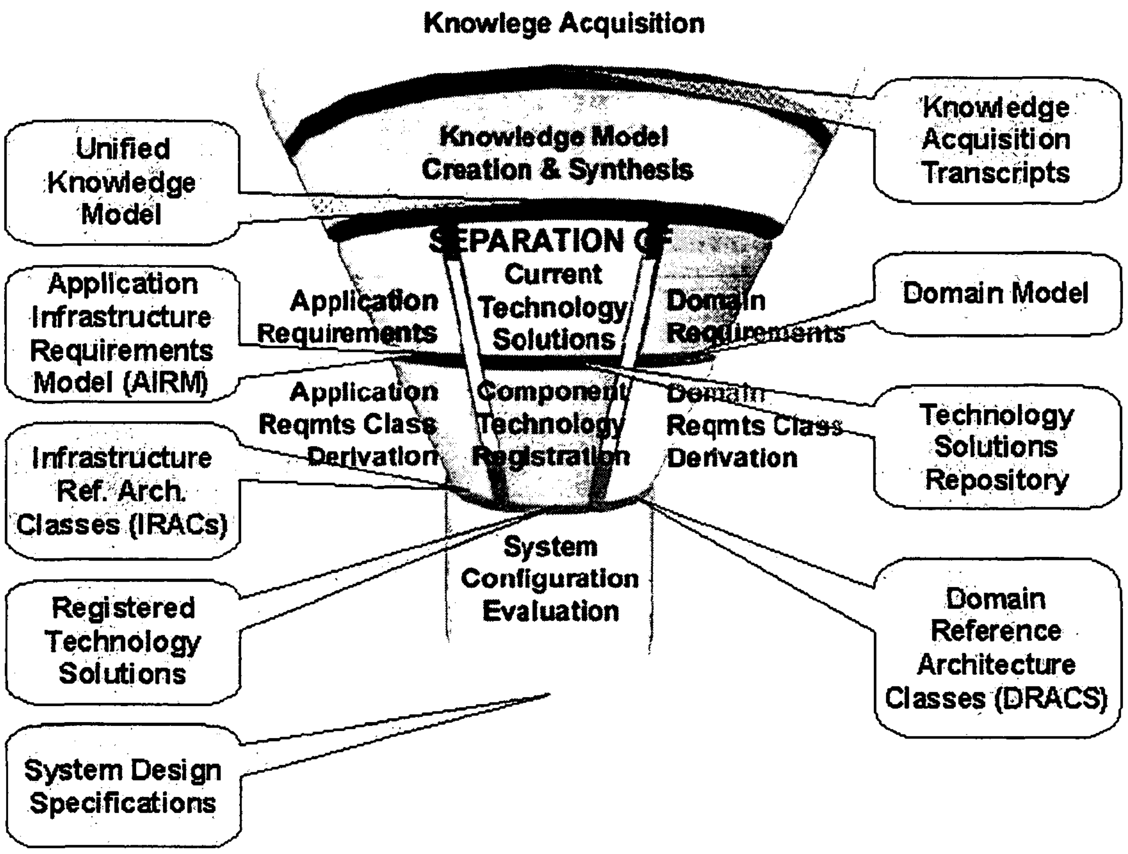

Knowlege Acquisition
Unified Knowledge Model
Knowledge Model Creation & Synthesis
Knowledge Acquisition Transcripts
SEPARATION OF
Application Infrastructure Requirements Model (AIRM)
Application Requirements
Current Technology Solutions
Domain Requirements
Domain Model
Application Reqmts Class Derivation
Component Technology Registration
Domain Reqmts Class Derivation
Technology Solutions Repository
Infrastructure Ref. Arch. Classes (IRACs)
System Configuration Evaluation
Registered Technology Solutions
Domain Reference Architecture Classes (DRACS)
System Design Specifications

Figure 8: The SEPA Process Funnel

Knowledge Acquisition

The SEPA methodology encourages gathering requirements from a diverse group of stakeholders (end-users, domain experts, developers, etc.) that are representative of all the contributing perspectives. The first step in the knowledge acquisition effort is to identify the relevant viewpoints from which domain knowledge will be elicited. To support the knowledge synthesis and resolution process, the domain viewpoints should be organized hierarchically. Next, domain experts must be selected and associated with a domain viewpoint. Ideally, each Knowledge Acquisition (KA) session will be conducted with a single stakeholder, and the artifacts generated as a result of that session will reflect knowledge from a single domain viewpoint.
The preferred approach to knowledge acquisition is to elicit information in the context of scenarios of operation (McGraw & Harbison, 1997), whether the scenario relates to the entire domain, current project, or past projects. In generating the session transcripts, the Knowledge Engineer should not worry about interpreting or filtering the acquired knowledge. Requirements modeling and validation will occur at a later phase, and the session transcripts will serve as an anchor for all subsequent requirements traceability.

Knowledge Model Creation and Viewpoint Resolution

Following the KA sessions, the knowledge that was elicited is represented in graphical Knowledge Models (KMs), including process traces, collaboration diagrams, task decompositions, etc. During knowledge modeling, the KE is often required to make interpretive decisions, such as the relevance of certain details, the proper levels of abstraction, and what to do about inconsistent facts. The knowledge models created from a given knowledge acquisition session are returned to the stakeholder who was interviewed for validation.
Once the knowledge models from a particular session have been validated by the stakeholders, the models must be integrated with other validated domain knowledge. These models reflect the viewpoint, level of abstraction, and terminology of the stakeholder from whom the information was acquired. To be useful to developers, the viewpoints of the individual stakeholders need to be resolved together to form a homogeneous set of system requirements. The primary deliverable of the knowledge synthesis activity is the Unified KM (UKM) synthesized from the viewpoints of multiple stakeholders.

The representation used for the synthesized model must be expressive enough to contain all of the KM types. While UML is a popular meta-language appropriate for certain task and data modeling, additional flexibility can be gained through the use of more abstract data representations (e.g., semantic nets or first-order logic). Elements and assertions from the individual KMs must be mapped into the common target representation of the synthesized model (UKM) and conflicts resolved. The difficulty of this conflict detection and resolution process is reduced if mapping proceeds through three phases. First, the main concepts from all KMs are mapped into the synthesized model so that the model contains the equivalent of a set of domain dictionary entries. Matching synonyms and disambiguating homonyms provides a more stable foundation on which later mappings can be made. The next phase assures that assertions regarding concept types are consistent. For example, a concept may be used as a "task" in one knowledge model but refer to a "state" in another model. Since the SEPA methodology asserts that the "task" and "state" concept types conflict, the Knowledge Engineer must either choose a single type or divide the original concept. During the final phase, relations between concepts are merged into the developing model. Between each phase, the UKM's syntactic rules help to identify conflicts, and the resolution of these conflicts is recorded to ensure traceability.
Knowledge models are combined in a stepwise fashion up the viewpoint hierarchy. First models relating to a specific session or stakeholder are merged. Next, all models related to a specific viewpoint in the domain viewpoint hierarchy are merged. At the end, the synthesized UKM represents the compromises between the disparate viewpoints in the domain. While model synthesis's primary deliverable is the UKM, the intermediate models are also of value. For instance, if an intermediate model is the result of all the knowledge acquisition sessions conducted with a particular stakeholder viewpoint (e.g. physician, nurse, etc.), then that model represents the domain from that viewpoint. These models, a byproduct of the synthesis process, represent new, viewpoint-based models. Viewpoint models are especially useful when developing components, such as GUIs, that are specific to a single type of domain user.

Separation of Requirements

Although the goal of a particular KA session may be to focus on the elicitation of one type of knowledge over another, a typical KA session may gather a broad spectrum of information. The information gathered may contain domain knowledge, specific application requirements, and/or information about specific technologies currently in use (e.g., legacy systems). In an ideal situation, the KE would conduct separate KA sessions with a domain expert or end-user for each of these types of information. However, domain experts and end-users typically do not have this abstracted view of their work and may find it difficult to provide such information. Thus the information captured in the UKM contains both domain-specific and application-specific information.
The SEPA methodology emphasizes the separation of requirements associated with a particular application from the requirements applicable to the domain in general. This emphasis on separation was made in an effort to facilitate reuse of domain analysis and design artifacts.
The domain requirements are the set of functional or data requirements shared by all or some of the respective viewpoints within a domain. The basic rule for identifying domain requirements is to locate tasks and data identified by the stakeholders. This information is gathered together into the Domain Model (DM).
Application requirements reflect the details of a system implementation or the implementation-specific requirements of a particular system client. In general, application requirements will be performance constraints on domain entities (tasks, roles, resources, etc.) or constraints on implementation choices. These requirements are gathered together into the Application Infrastructure Requirements Model (AIRM).
Performance constraints are quantitative requirements on the domain entities irrespective of implementation strategy, e.g., speed constraints, numerical accuracy, etc. Implementation or delivery constraints specify user requirements on the implementation of the system. For instance, a delivery constraint might be that the payroll system must be developed for a specific hardware platform and the system must interact with a particular legacy application. User interface requirements are also an example of implementation constraints. For instance, a particular application may be required to support federal accessibility guidelines, or adhere to the interface defined in a prototype model.
During Knowledge Acquisition, Knowledge Engineers are often given information about specific technology solutions, such as COTS tools or legacy applications. A technology specification template is available to the Knowledge Engineer to assist in gathering application information during this phase. This structure of the software specification template is based on the IEEE 830-1993 Recommended Practice for Software Requirements Specification and our extensive interaction with software developers and integrators on prior/current development efforts.

Domain Requirements Class Derivation

The Domain Reference Architecture (DRA), is comprised of a collection of domain-based component classes derived from information in the Domain Model. The Domain RA provides a key link between analysis and implementation, reflecting domain information found in the Domain Model and providing developers with a template for identifying and developing new technologies for particular implementations. The Domain RA must be completely domain-specific and highly flexible for building similar systems in the future. This flexibility is achieved by describing Domain Reference Architecture Classes (DRACs) in terms of their functional requirements.

In the process of creating the RA, the functional, procedural Domain Model is transitioned to an object-oriented Reference Architecture composed of technology independent DRACs applicable to a family of applications in the domain. While it is feasible to partition DRACs along either functional or object-oriented boundaries, an object-oriented approach promotes qualities such as reusability, faster development, flexibility, scalability, maintainability, and better correspondence to the domain being modeled (Graham, 1995). More importantly, by following an object-oriented approach, the delineation of DRACs based on what services must be performed allows for flexibility and accommodates changes in how a task is performed when DRACs are instantiated in a particular system design.

A DRAC is assigned responsibilities based on domain tasks and represented by three categories of information (Graser, 1996). The Declarative Model (D-M) defines the attributes and services contained within and offered by a DRAC. The Behavioral Model (B-M) defines the states of a DRAC, the transitions between those states, and the events which affect transitions. The Integration Model (I-M) defines the constraints and dependencies between DRACs, capturing integration requirements, potential subsystem compositions, and user integration preferences.

RA derivation is an iterative process, where successive iterations represent increasing coverage of domain information and greater refinement of the RA based on architectural goals. Typical architecture goals include extensibility, comprehensibility, and maintainability, often reflecting the benefits associated with object-oriented approaches. RARE guides the architect towards these goals through the application of architecture heuristics, architecture metrics, and heuristic strategies. These are described as follows:

- *Architecture Heuristic:* A "rule of thumb" compiled from expert experience on past projects which assists the architect in making rational decisions in defining DRACs. One well-known object-oriented heuristic recommends reducing coupling among DRACs to encourage reuse (Riel, 1997).
- *Architecture Metric:* A measurement of a particular characteristic of an RA which provides an indication whether the architect adhered to a given heuristic. Continuing with the previous example, the DRAC inheritance hierarchy and/or number of messages passed among DRACs would provide some evidence as to the degree of coupling in the RA.
- *Heuristic Strategy:* A step-by-step procedure (sequence of actions) used to apply a given heuristic. Following the "reduce coupling" example, a strategy might explicitly state, "move service S1 from DRAC D1 to DRAC D2" to eliminate the need to exchange data between DRACs D1 and D2.

  The resulting collection of Domain Reference Architecture Classes (DRACs) comprise a domain-based, object-oriented Reference Architecture which is traceable to stakeholder requirements through the functional requirements defined in the DM, provides a blueprint for developers, aids in domain understanding, and identifies rules of composition among architecture classes.

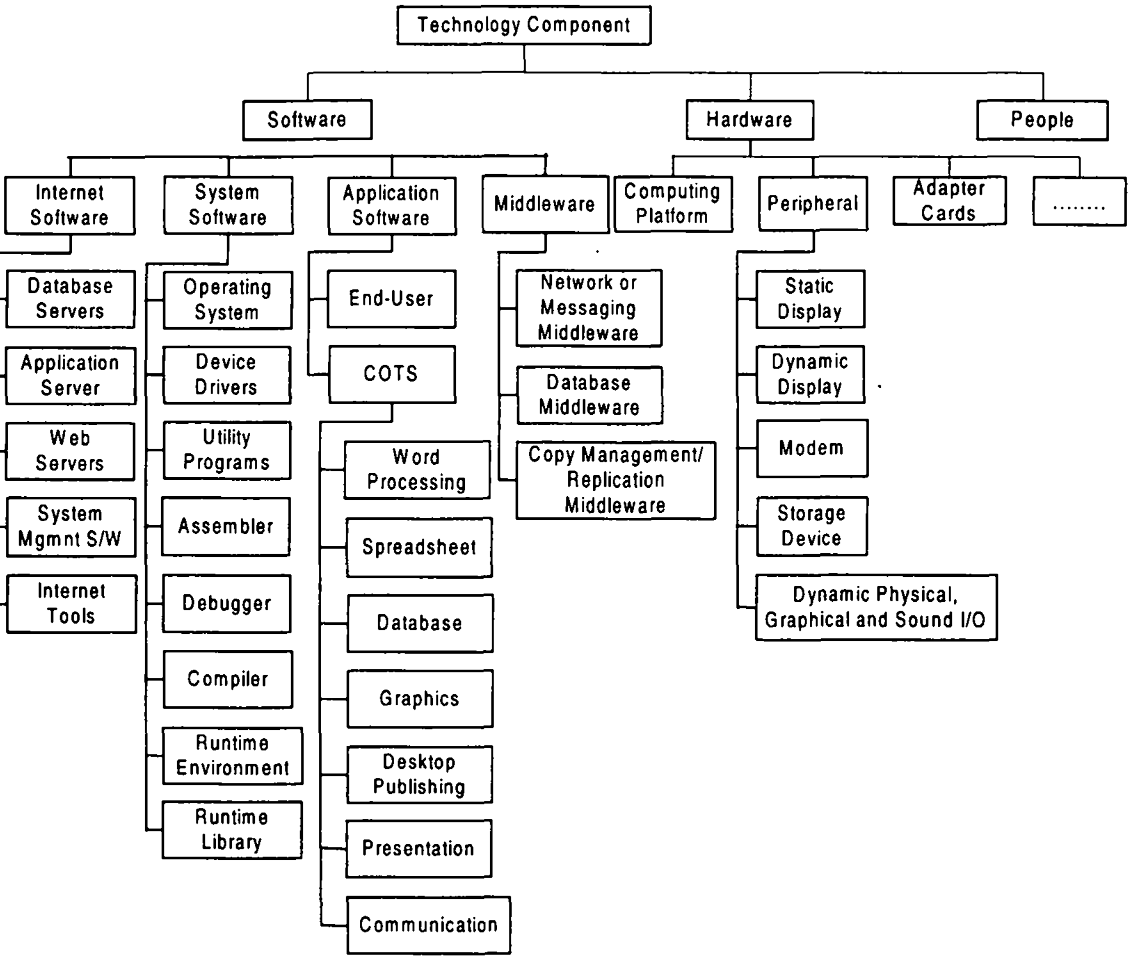


Figure 9: Subset of IRAC Hierarchy

Application Requirements Classes

The Infrastructure Reference Architecture (IRA) models classifications of technology solutions and associated classifications of infrastructure required to support the different kinds of technology solutions. These technology solution classifications span from software, to hardware equipment, to people.

Most software systems require such basic service support as operating systems, database management systems, and middleware (Braun, 1999). The services provided by those kinds of components – *e.g.*, resource/data management, communication, coordination, and user-interface – comprise the system infrastructure and can be constrained by the system stakeholders. For instance, a software system deployment site may require that the system communicate via a wireless network. The system will require specific types of networking hardware, and the networking middleware will have to be configured for broadcast, rather than point-to-point organization. Since these types of requirements refer to and constrain specific implementations, these requirements are not captured in the Domain Reference Architecture.

System delivery constraints specifying implementation and installation constraints were previously identified and represented in the Application Infrastructure Requirements Model (AIRM). Maintaining traceability links to the original source, delivery constraints are mapped to instances of Infrastructure Reference Architecture Classes (IRACs). IRACs model technology solutions as domain-neutral classes of implementations. When technology solutions are registered, their implementation details are specified in terms of IRACs and are evaluated against the delivery constraints. A high-level view of the IRACs is presented in Figure 9. In addition to the classes themselves, the ontology contains relationships between the classes. For instance, a class of applications identified as *Interpreted Programs*, require the services of a *Runtime Environment* to execute. *Networking Middleware* requires a *Network Adaptor* to connect components on different *Computing Platforms*. The technology registration process handles instantiating the specifications of technology solutions with regard to the IRACs and modeling the technology implementation and infrastructure constraints.

Component Technology Registration

Component Technology Registration is the process of modeling technology solutions (e.g. legacy systems, COTS applications, application systems or components under development, or planned systems or components)

by mapping the technology's coverage of functional requirements and describing how the technology is implemented. Since a wide array of technology exists (namely, technology an end-user recognizes as delivering some functional services and infrastructure-related technology), SEPA distinguishes the two during the registration process.
An "end-user application" is a specific technology solution which can be directly mapped to the domain requirements it provides. Registration of an end-user application begins by identifying a DRAC which approximates the basic domain functionality of the application. Ideally, the functionality of the end-user application will align very closely with the structure of the DRAC in the Domain RA, since this will allow systems to take advantage of the object-oriented features of the Reference Architecture design. An application is *registered to a DRAC* if it offers any service or manages any data or events in the specification of that DRAC. However, registration can be handled on a per-service or per-data-element basis as well.
After the domain requirements have been covered, it is necessary to acquire information about the way in which the application has been constructed. An end-user application is composed of one or more infrastructure technologies. Knowledge about the infrastructure requirements of the end-user application are represented as instances of respective IRACs. An end-user application might be composed of an executable software application, a database, an operating system, and a computing platform with a large hard disk. Each of these entities would be represented as infrastructure application technology which must be configured to implement the end-user application.

System Configuration Evaluation

As the system designers and integrators attempt to determine which application technology or systems can be deployed in a particular system configuration for a specific user installation site, they must determine the ability of applications to satisfy domain-related functional and data requirements specified in the Domain RA, the state of the applications (legacy custom system, COTS application, under development, planned), and the ability of the applications to meet delivery expectations with regard to implementation.
A System Design Specification (SDS) is constructed for a site by first selecting the subset of the Domain RA (DRACs or selected services within DRACs) which must be implemented for that site. For convenience, this subset is labeled as a "version" of the DRA. As defined in SEPA, a "site" is a group of physical computing platforms or locations which collectively must provide the services in that version. Once DRA services have been selected, end-user applications registered to those services can be identified. Knowledge about application infrastructure and relationships to DRA component classes guide decisions in selecting how a domain service in the DRA can be satisfied by candidate technology solutions. For example, DRA services are related based on the events and data they exchange; these service dependencies are represented in respective DRAC Integration Models. The selection of a particular service for an installation may infer additional required services based on service dependencies.
An Application Architecture is created for the site which covers the desired subset of the domain requirements in the Domain RA version. Depending on the alignment of the selected end-user applications to their corresponding DRAC boundaries, the Application Architecture may or may not demonstrate the beneficial design characteristics of the Domain Reference Architecture.
If the Application Architecture is satisfactory, then the underlying system infrastructure can be configured from infrastructure applications to create a Site Implementation Architecture. Several dimensions of evaluation are possible. First, the basic installation requirements of the end-user applications can be checked for satisfaction. Second, a preliminary integration report can be generated, driven by the topology of the Application Architecture and utilizing the infrastructure details in the Implementation Architecture. The integration reports can identify potential integration problems and suggest areas for further testing. Finally, the Implementation Architecture can be evaluated to determine if application and site requirements instantiated in the IRACs have been satisfied.
The solutions may be chosen based on any number of design trade-off concerns (e.g. cost, availability, ease of implementation, etc.). The resulting System Design Specification contains a set of application component solutions, system infrastructure components (e.g. COTS tools, middleware, and hardware), and the integration dependencies between respective end-user applications and between end-user applications and the infrastructure. The user may identify multiple system specifications that adequately satisfy requirements.

REQUIREMENTS MANAGEMENT AND REUSE EXAMPLE USING THE SEPA PROCESS AND TOOL SUITE

SEPA is being applied in a number of domains, including cancer treatment protocol development, emergency incident response, and joint military forces command and control. Among them is a Defense Advance Research Project Agency (DARPA) initiative to develop information systems supporting first responders (fire, police, EMS) during emergency situations (e.g., fire, natural disaster, chemical/biological [chem/bio] attacks) that may occur at large public events (e.g. rock concerts, political gatherings). In today's emergency response environment, most incident types are initially managed by first responders. The process by which the incident is managed is locally defined and evolves during the course of the incident, based on pre-defined Standard Operating Procedures and local resource availability. The system under development will have functionality including responder task assignment and location tracking, casualty assessment, chem/bio agent analysis, and post-incident reporting and analysis. A key mandate of the resulting system is that it be flexible enough to support a wide variety of incident response facilities and organizations. This mandate will demand a significant amount of application reuse and system customizability. The requirements of each facility must be evaluated in the context of the system and incident response domain and satisfied by associated applications, making reuse at the requirements level the clear approach to multi-site implementation.

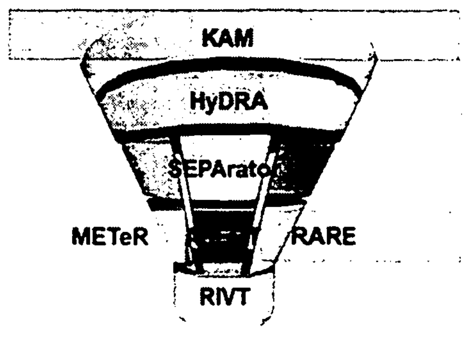
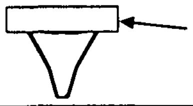


Figure 10: SEPA Tool Suite

On large, complex projects such as the DARPA emergency response initiative, the requirements management processes suggested by the SEPA methodology are impractical without adequate tool support. SEPA process research is complemented by a suite of tools to support each activity in the SEPA process. Figure 10 illustrates how the tool suite covers the activities associated with the SEPA funnel presented in Figure 8. A brief description of each tool follows.

1. ***Knowledge Acquisition Manager (KAM):*** Provides project and document management functions for the Knowledge Acquisition (KA) process through a web interface. KAM allows the knowledge engineer to (i) document KA plans, (ii) specify and maintain participant (e.g. domain experts, end users, knowledge engineers) contact info and background profiles, (iii) document intended KA session objectives, (iv) document elicitation scenarios, and (v) upload and search session reports which document the knowledge acquired during a session.
2. ***Hybrid Domain Representation Archive (HyDRA):*** Provides intelligent reasoning functions that guide the user during the creation of knowledge models (KMs) and unification of the KMs. HyDRA's objectives are to: (i) aid the Knowledge Engineer (KE) by providing tool support for KA and modeling; (ii) automate the transition from unstructured, incomplete requirements to formal, complete, and consistent requirements; and (iii) maintain traceability through the necessary merging of requirements from varying viewpoints into the Unified Knowledge Model (UKM), representing the combined viewpoints of all domain experts.
3. ***SEPArator:*** Assists the Knowledge Engineer in navigating the UKM and separating domain requirements from application requirements, and identifying specifications of existing, current technology solutions.
4. ***Reference Architecture Representation Environment (RARE):*** Semi-automatically guides in the transition from the Domain Model (DM) to a Domain Reference Architecture (DRA) by systematically applying object-oriented (OO) heuristics, software architecture heuristics, and quality metrics. The derivation process is guided by goals prioritized by the system architect, including extensibility, reusability, comprehensibility, and maintainability.
5. ***Modeling Environment for Technology Requirements (METeR):*** Assists the developer in modeling application requirements from the UKM in the interest of domain-independent Infrastructure Reference Architecture Classes (IRACs) to capture (i) delivery requirements imposed by client demands; (ii) installation constraints imposed by site requirements; (iii) application installation requirements; and (iv) integration issues imposed by the selection of cooperating applications.
6. ***Application Specification Environment and Registration Tool (ASERT):*** Provides system developers with an interface to specify applications in terms of data and functional requirements in the (DRA) and application installation and integration requirements in the (IRA).
7. ***Requirements and Integration Verification Tool (RIVT):*** Assists stakeholders in system configuration and interrogating requirements in the DRA and IRA from various perspectives (e.g. end-user questions, developer questions, system integrator questions). RIVT's querying capabilities encourage reuse by (i) aiding the identification and evaluation of applications against domain, application, and system infrastructure requirements and (ii) facilitating impact analysis as requirements and application technology change and/or evolve.

Using the DARPA emergency response project for illustration, the following subsections guide the reader through the activities depicted by the SEPA funnel, introducing the above tools as they become relevant.



SEPA Activity: Knowledge Acquisition
Supporting Tool: Knowledge Acquisition Manager (KAM)

The first step in gathering requirements is to determine the stakeholder viewpoints or perspectives that must be considered when designing the system (Sommerville & Sawyer, 1997). Sommerville suggests that these viewpoints be gathered into a stakeholder viewpoint hierarchy. SEPA's Knowledge Acquisition Manager (KAM) tool actually uses two orthogonal hierarchies. The first hierarchy is used to describe the domain role viewpoints (with leaf nodes such as HazMat Incident Commander or Fire Administrator). The second hierarchy is used to describe the organizational viewpoints (with leaf nodes such as City Y Fire Department or City X Police Department). While the KAM tool uses both viewpoint hierarchies simultaneously, this example will only use the shaded portion of the simple domain role viewpoint hierarchy shown in
Figure 11.

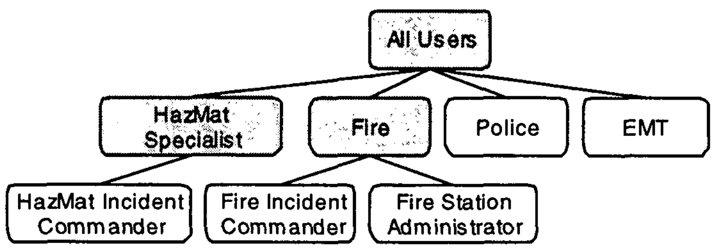


Figure 11: Viewpoint Hierarchy

Domain experts for this project were selected so that a variety of expert viewpoints were represented, which for our example includes fire personnel and hazardous material specialists. The KAM tool was used to maintain contact information on project participants and allowed these domain experts to be assigned to leaf nodes of two hierarchies based on their domain role viewpoint and organization viewpoint. KAM was also used to schedule several KA Sessions with these experts and can record session information such as session goals, times, dates, locations, participants, and KA approaches.

Although KEs used a variety of approaches to elicit domain and system requirements, scenario analysis was the primary approach used to acquire domain task, performer, and timing information (Harbison, 1997). Other KA Sessions focused on acquiring specific implementation requirements through prototype review, yielding information about preferred look-and-feel as well as installation specifics such as required operating system. The SEPA example which follows is rooted in the information acquired from the sessions listed in Table 1.

Table 1: KA Sessions for Incident Response Example

Session	Expert with Viewpoints	Location	KA Approach	Information Gathered
1	Jim Hendrix – *Hazardous Materials Specialist* and *HazMat Incident Commander*	ity X - ity Response Center	Scenario Analysis	Tasks, performers, and resources for chem/bio response
2	• Sam Cook – *Fireman, Fire Incident Commander*, and *Fire Station Administrator*	ity Y - ire Station 1	Scenario Analysis	Tasks, performers, and resources for fire response
3	Bob Smith – *Fireman* and *Fire Incident Commander*	ity Y – ire Station 2	Prototype Review	User Interface (UI) requirements and specific installation needs for fire response

To document each session, Knowledge Engineers (KEs) created one or more KA Session Reports, accompanied by supporting diagrams, videos, and supplemental documents. KA Session Reports represent knowledge from domain role and organizational viewpoints associated with the domain experts involved in each session. Knowledge Engineers use KAM to (1) identify domain experts, (2) define domain and organization viewpoints, (3) define an overall KA session plan, and (4) store the products from each session. Project participants can perform content-based searches with filters based on both domain role and organization viewpoints. These session reports, diagrams, videos, and supplemental documents provide the foundation for requirements traceability in the SEPA methodology.

SEPA Activity: Knowledge Modeling and Knowledge Model Synthesis
Supporting Tools: Hybrid Domain Representation Archive (HyDRA) and third party modeling tools

Information found in KA session reports and related documents is not structured, making effective reasoning by a computer difficult. To transition KA artifacts to a computational representation, the KE interprets the artifacts and creates structured graphical and textual Knowledge Models (KMs) in HyDRA and other third party modeling tools (e.g., Oracle Designer (Oracle, 1999)). Currently, the KE has the choice of a number of models, including data flow diagrams, task decomposition diagrams, task templates, Venn diagrams, entity-relationship diagrams, and concept maps. Appropriate knowledge models are selected based on the type of knowledge acquired. For example, task decomposition diagrams provide an overall view of domain tasks and subtasks, while a task template contains specifics about the data, timing, and performance requirements for a specific task. To facilitate the validation of KA artifacts with domain experts, each KA session yields a standalone collection of knowledge models called a Model Space (MS). The resulting knowledge models generated from the sessions in Table 1 are summarized in Table 2. Corresponding knowledge models are shown in Figure 7– Figure 12.

Table 2: Knowledge Models from Incident Response Example

Knowledge Model	**Source KA Session**	**Data Represented**
Venn Diagram (Figure 7**Error! Reference source not found.**)	1	Attributes composing the "weather" concept.
Task Decomposition (Figure 8**Error! Reference source not found.**)	1	Task decomposition for chem/bio incident response.
Task Templates (Figure 9**Error! Reference source not found.**)	1	Selected task details for chem/bio incident response tasks, including pre/post conditions, performer, and input/output data.
Task Decomposition (Figure 10**Error! Reference source not found.**)	2	Task decomposition for small fire incident response.
Task Templates (Not Shown)	2	Selected task details for small fire incident response tasks, including pre/post conditions, performer, and input/output data.
Task Performance Constraint Template (Figure 11**Error! Reference source not found.**)	3	Prototype presented to expert is associated with respective domain tasks as suggested implementation approach.
System Constraint Template (Figure 12**Error! Reference source not found.**)	3	Overriding system implementation constraints (e.g. operating system).

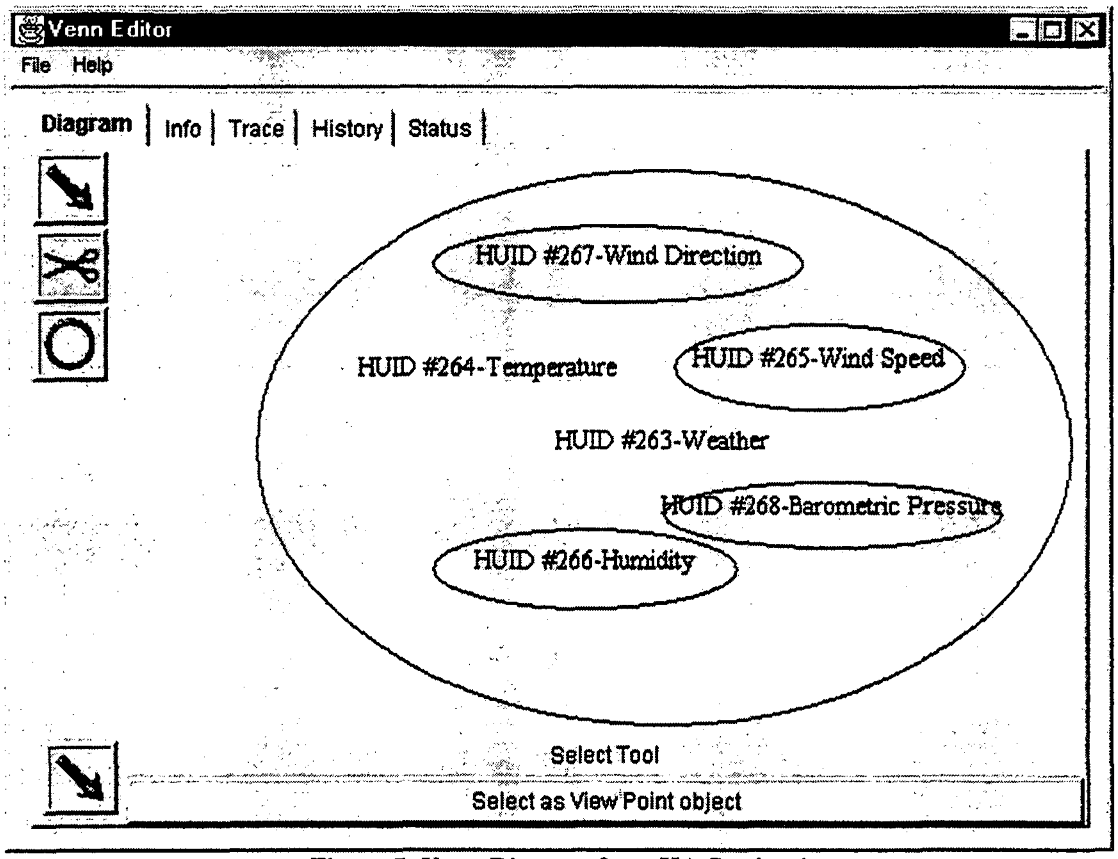


Figure 7: Venn Diagram from KA Session 1

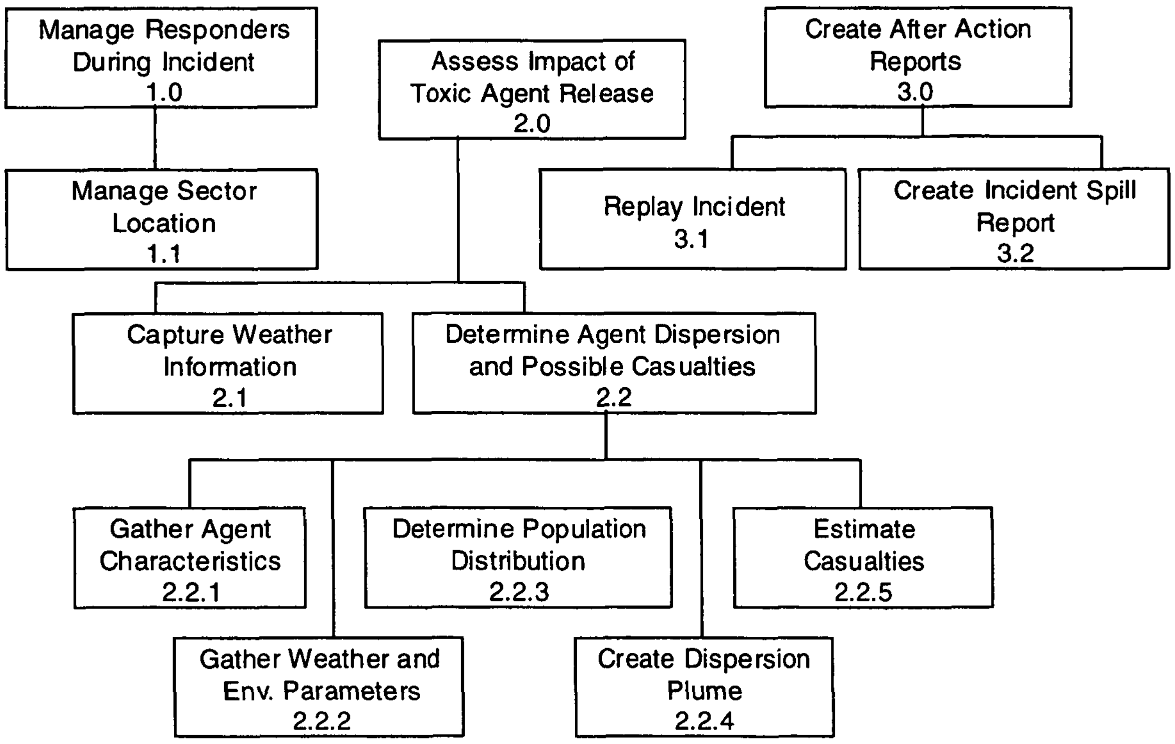

Manage Responders During Incident 1.0
Assess Impact of Toxic Agent Release 2.0
Create After Action Reports 3.0
Manage Sector Location 1.1
Replay Incident 3.1
Create Incident Spill Report 3.2
Capture Weather Information 2.1
Determine Agent Dispersion and Possible Casualties 2.2
Gather Agent Characteristics 2.2.1
Determine Population Distribution 2.2.3
Estimate Casualties 2.2.5
Gather Weather and Env. Parameters 2.2.2
Create Dispersion Plume 2.2.4

Figure 8: Summary of Task Decomposition from KA Session 1

Task: Manage Responders During Incident
Precondition: During Incident
Postcondition: not specified
Performer: Incident Commander
Data Input: not specified
Data Output: not specified

Task: Gather Weather and Env Parameters
Precondition: not specified
Postcondition: not specified
Performer: not specified
Data Input: Weather
Data Output: not specified

Figure 9: Summary of Task Templates from KA Session 1

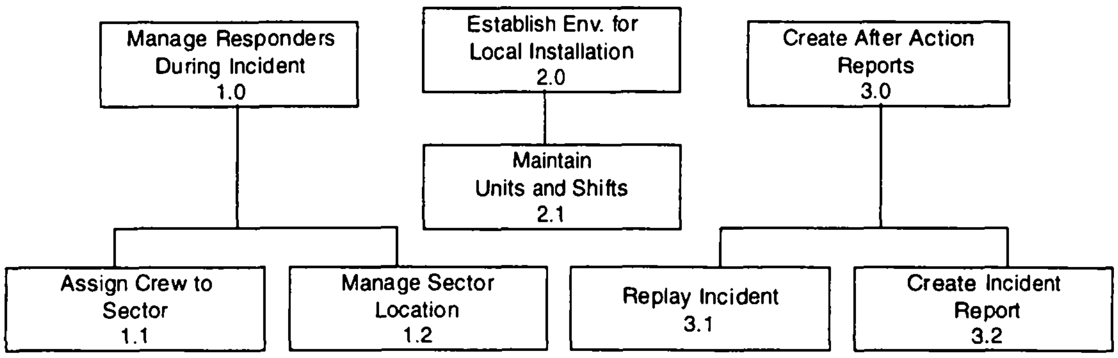

Manage Responders During Incident 1.0
Establish Env. for Local Installation 2.0
Create After Action Reports 3.0
Maintain Units and Shifts 2.1
Assign Crew to Sector 1.1
Manage Sector Location 1.2
Replay Incident 3.1
Create Incident Report 3.2

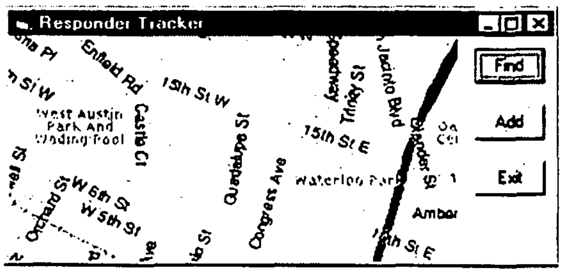
Figure 10: Summary of Task Decomposition from KA Session 2

To achieve a single, unified picture of requirements, HYDRA assists the KE in synthesizing requirements gathered from multiple experts, following a hierarchy of viewpoints defined for a domain. In this example, all models from "Hazardous Materials Specialist" experts would be merged into a "Hazardous Materials Specialist" model space (step 1 in Figure 13**Error! Reference source not found.**), and all models from "Incident Commander" experts would be merged into an "Incident Commander" model space (step 2 in Figure 13). These models space would subsequently be merged into a unified model space, the Unified Knowledge Model (UKM) (step 3 in figure 13).

Implementation ID: City Y Fire Dept.
Constraints:
Operating System: Windows NT
Memory : 128 MB
Fixed Disk: 8 GB
LAN: 10 Mbps Ethernet

Figure 11: Summary of System Constraint Template from KA Session 2

Task: Manage Responders During Incident
Interface Prototype:



Prototype Description: Map-based approach for assigning responders to various incident hotspots and tracking their location as an incident progresses.

Figure 12: Summary of Task Performance Constraint from KA Session 2

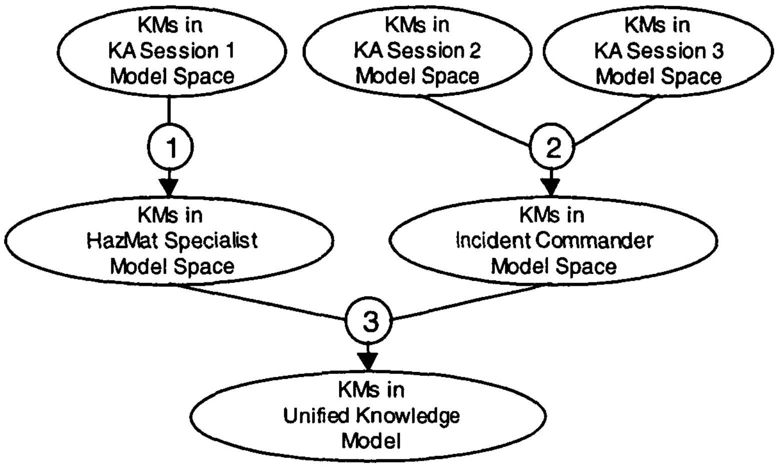

KMs in KA Session 1 Model Space
KMs in KA Session 2 Model Space
KMs in KA Session 3 Model Space
1
2
KMs in HazMat Specialist Model Space
KMs in Incident Commander Model Space
3
KMs in Unified Knowledge Model

Figure 13: Knowledge Synthesis Process

As synthesis proceeds, HyDRA detects conflicts between the knowledge models and presents the KE with possible resolutions. For example, when the ordering of two tasks vary between two models, HyDRA prompts the KE to choose which ordering to retain or to mark the tasks unordered. As each decision is made, HyDRA ensures the KE's rationale is captured. Traceability is retained to record how model elements were changed or merged to produce the resulting element. Details regarding the synthesis operation and representations can be found in (Barber & Jernigan, 1999).

Conflicts detected in this example along with their selected resolutions are:

1. *Task with almost identical children:* Task 3.0 in Figure 8 and task 3.0 in Figure 10**Error! Reference source not found.** have the same name and share a common subtask. However, the other subtask (3.2) is different. During the merge, HyDRA detects this situation and asks the KE if it should (i) keep all three subtasks, (ii) ignore one of the dissimilar subtasks, or (iii) merge the two dissimilar subtasks into one task. In this case, the KE recognized that these are the same task and asked HyDRA to merge the two dissimilar tasks into one with the name "Create Incident Report." Alternatively, the difference between the subtasks under task 1.0, "Manage Responders During Incident," of the two task decompositions will not create a problem if the task decomposition from the first KA session (the session with only one subtask), is marked with an open world semantics or incomplete flag.
2. *Type - usage conflict:* HyDRA holds an ontology that specifies types of model elements, classifications of those model elements, and the relationships between those model types. The Venn diagram for "Weather" indicated the concepts in the diagram were "states" (not shown in panel exposed in Figure 7). However, the "Weather" concept was used in a task template knowledge model as a "resource" (Data Input in Figure 9). HyDRA detects this usage as being inconsistent with the type provided by the Venn diagram (Figure 7). Since, the knowledge models were validated by the domain experts before the merge, HyDRA cannot change them and preserve the validation. However, it can ask if the user would like to ignore the conflict, ignore the usage of "Weather" in the task template, or ignore the type information from the Venn diagram. Finally, the user has the choice of changing a source knowledge model, revalidating it with the domain expert, and re-performing the merge process. In the example, the KE chose to change the Venn Diagram so that it indicated that the concepts were to be classified as type "information," a non-volitional, intangible, and consumable kind of resource.
3. *"Weather" not created:* As part of the last steps of a merge, HyDRA attempts to perform some completeness checks. Due to the decisions resulting from conflict 2 above, "Weather" is now treated as a consumable resource model element type. As a consumable resource, it must have been produced by some task before it can be used by the "Gather Weather and Env Parameters" task (described in Figure 9). Since no task produces "Weather," a conflict arises. To resolve this conflict the KE adds "Weather" as a Data Output on the "Capture Weather Information" task and re-merges.

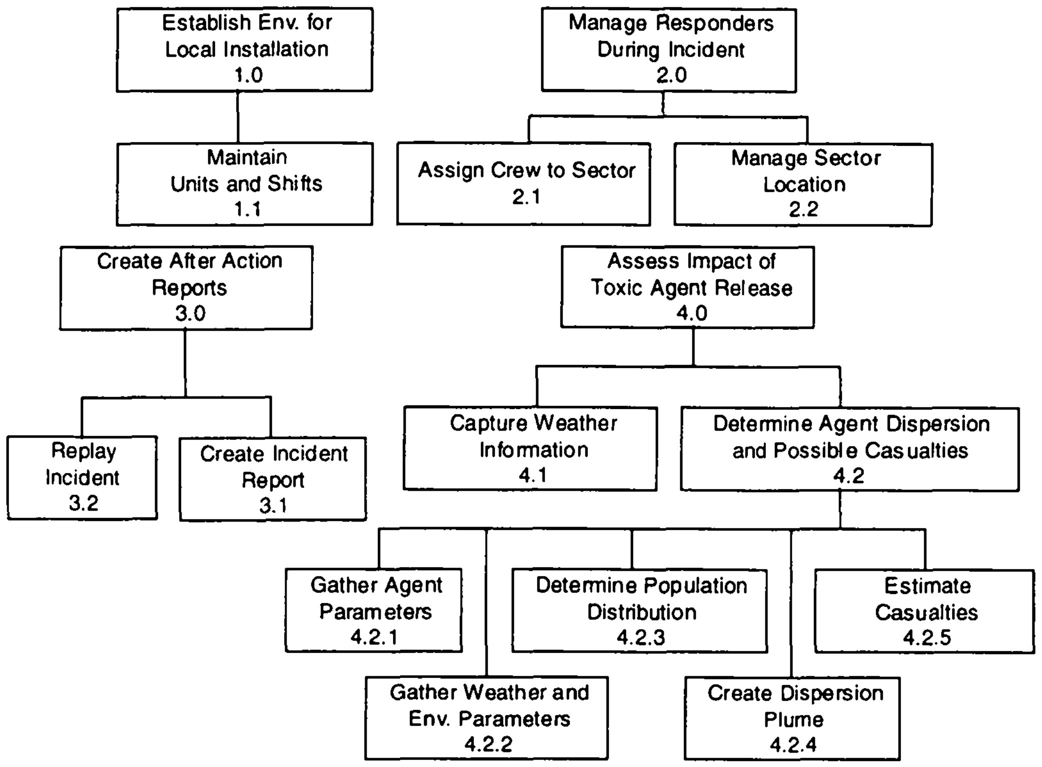

Establish Env. for Local Installation 1.0
Manage Responders During Incident 2.0
Maintain Units and Shifts 1.1
Assign Crew to Sector 2.1
Manage Sector Location 2.2
Create After Action Reports 3.0
Assess Impact of Toxic Agent Release 4.0
Replay Incident 3.2
Create Incident Report 3.1
Capture Weather Information 4.1
Determine Agent Dispersion and Possible Casualties 4.2
Gather Agent Parameters 4.2.1
Determine Population Distribution 4.2.3
Estimate Casualties 4.2.5
Gather Weather and Env. Parameters 4.2.2
Create Dispersion Plume 4.2.4

Figure 14: Merged Task Decomposition from UKM

Following the synthesis operations depicted in Figure 13, the incident response UKM should include concepts originating from the combined set of knowledge models. The merged task decomposition information in the UKM is illustrated in Figure 14 for clarity.

SEPA Activity: Separation of Domain and Application Requirements
Supporting Tool: SEPArator

The UKM artifact provides a single, consistent model that becomes the basis for all further requirements analysis in the SEPA methodology. Before a Domain Reference Architecture can be derived, domain requirements must be separated from application requirements in the UKM. SEPArator aids the process of identifying domain content in the UKM, originally represented as concept models (such as the Venn Diagram in Figure 7) and task-based models. The resulting extraction of domain requirements is captured as the Domain Model (DM). In contrast to concept and task-based models, task performance constraint and system constraint models (Figure 11 and Figure 12) are used to characterize currently available or desired applications and sites. This information is subsequently instantiated in the Infrastructure Reference Architecture (IRA), and becomes particularly relevant during technology registration and system design. In this example only domain requirements were elicited from KA sessions 1 and 2. Application requirements were elicited from KA session 3.

SEPA Activity: Reference Architecture Derivation and Application Registration
Supporting Tools: Reference Architecture Representation Environment (RARE) and Application Specification Environment and Registration Tool (ASERT)

The domain requirements represented in the Domain Model (DM) reflect the synthesis of domain tasks and data expressed by stakeholders and compiled by KEs. This knowledge is captured with respect to scenarios of operation. Stakeholders describe their jobs, the data they exchange, and the technology they use in the context of those scenarios. Given this elicitation approach, the DM follows a functional, rather than object-oriented, decomposition. However, to encourage the reuse of requirements and applications in multiple system configurations, SEPA mandates that domain requirements be organized into object-oriented classes to which applications are registered.

The RARE tool assists in the translation from the function-based Domain Model (DM) to an object-oriented Domain Reference Architecture (DRA). This translation involves the creation of Domain Reference

Architecture Classes (DRACs) and the allocation of tasks and data represented in the DM to specific DRACs. The DRA serves as a blueprint for developers, where each DRAC specifies domain data owned, domain services (tasks) provided, and data and event dependencies with other DRACs. These dependencies arise from data and events exchanged between services in different DRACs.

Domain tasks, data, and events can be allocated among classes in a number of possible combinations. As with traditional object-oriented analysis, the identification of classes is not an exact science but strongly depends on the priorities and skills of the analyst. Furthermore, an entire object-oriented model is rarely defined in a single iteration; the model is typically refined over multiple passes. RARE supports the iterative nature of object-oriented analysis and applies proven object-oriented heuristics based on priorities established by the analyst. To document the analyst's decisions and the evolution of the DRA, the analyst's rationale is captured and traceability to DM tasks and data is established.

To drive the allocation process, RARE follows a set of high-level goals prioritized by the architect. Four primary goals were emphasized in the DARPA emergency response project:

1. *Align DRACs with task performers in the domain:* Many domain tasks will remain un-automated, thus the collections of tasks assigned to a DRAC should closely follow the tasks assigned to the respective domain performer (e.g. Incident Commander).
2. *Align DRACs with existing COTS applications:* To maximize reuse of existing COTS applications capable of performing domain tasks, tasks should be grouped based on those tasks associated with existing applications.
3. *Create classes that increase installation customizability:* To increase customizability for each installation, it is recommended to reduce the number of services offered by each DRAC and thus increase the total number of DRACs. In general, a greater number of smaller DRACs can be combined in more arrangements to support specific installation requirements.
4. *Allocate tasks among classes based on when tasks are typically performed during an incident response:* Incident responders only concern themselves with a certain set of tasks during any particular time period of an incident. For example, tasks such as assigning crews to shifts and defining resources (e.g. fire trucks) are done outside of any particular incident. Tasks in different time periods naturally have fewer coupling constraints than tasks in the same time period. Therefore these large grained time periods provide guidance for the assignment of tasks to DRACs so that inter-DRAC coupling and the number of DRACs the responder must interact with during any given period are reduced.

A subset of the DRA produced for the DARPA emergency response project is depicted in Figure 15 below. The "Incident Manager" DRAC was created based on goal 1, while the creation of the "Weather Manager" DRAC was strongly influenced by goal 3.

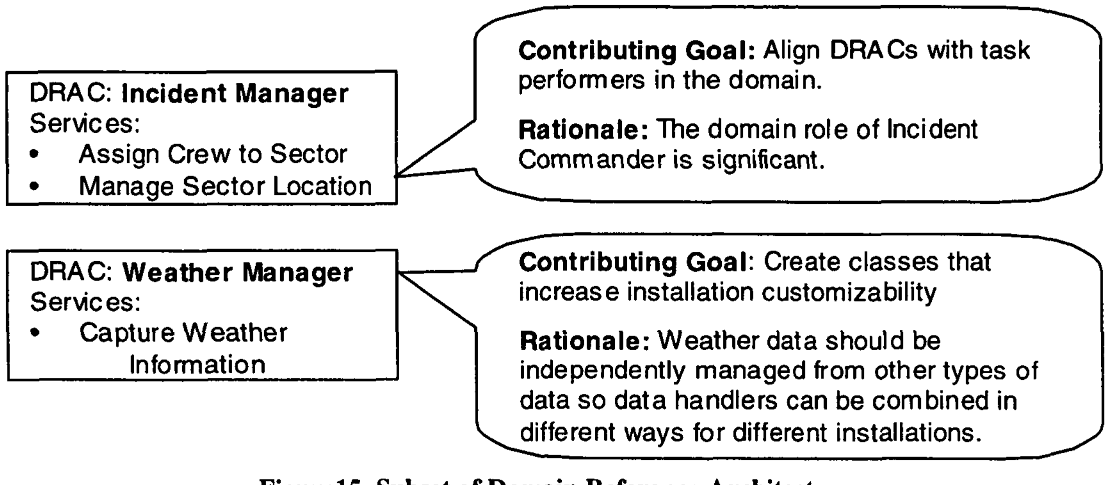

DRAC: **Incident Manager**
Services:
• Assign Crew to Sector
• Manage Sector Location
Contributing Goal: Align DRACs with task performers in the domain.
Rationale: The domain role of Incident Commander is significant.
DRAC: **Weather Manager**
Services:
• Capture Weather Information
Contributing Goal: Create classes that increase installation customizability
Rationale: Weather data should be independently managed from other types of data so data handlers can be combined in different ways for different installations.

Figure15: Subset of Domain Reference Architecture

Once the DRA is defined, the process of associating a new, existing, or proposed (notional) end-user application with DRAC data and functionality is referred to as "registration." The ASERT tool allows the developer to identify DRAC data and services provided by both COTS applications and customized applications. In this example, the applications under consideration and the domain services they provide include the following:

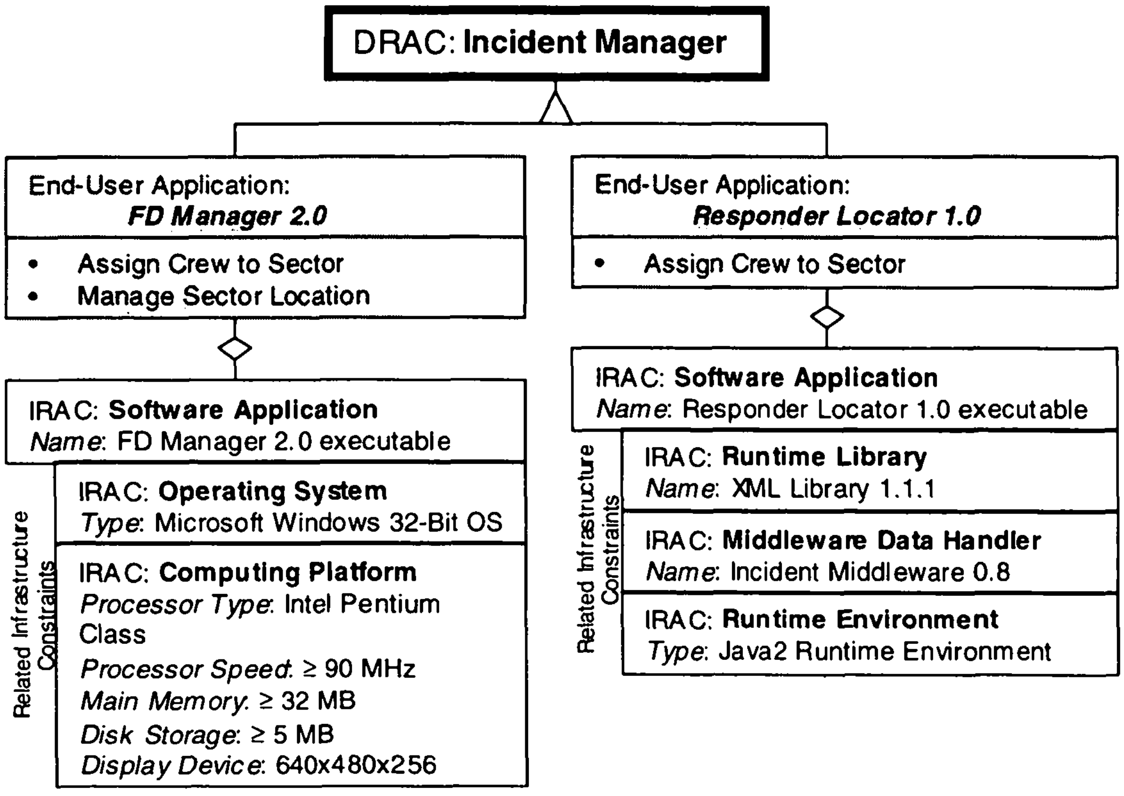
- *FDManager v2.0* – A COTS application that provides a complete set of services to support small and midrange incidents. These services include resource allocation and responder assignment and tracking. *FDManager v2.0* provides DRAC services:
- "Assign Crew to Sector" in DRAC Incident Manager
- "Manage Sector Location" in DRAC Incident Manager

- *ResponderLocator v1.0* – A newly developed map-based application used to assign tasks and locations to responders and monitor their location. *ResponderLocator v1.0* provides DRAC services:
  - "Assign Crew to Sector" in DRAC Incident Manager
  - "Manage Sector Location" in DRAC Incident Manager
- *WeatherMonitor v0.7* – A newly developed application that provides an interface for collecting weather data and disseminating this data to multiple responders. *WeatherMonitor v0.7* provides DRAC service "Capture Weather Information" in DRAC Incident Manager.

SEPA Activity: Application Requirements Modeling
Supporting Tool: Modeling Environment for Technology Requirements (METeR)

Associating an application to respective DRAC services is only part of the registration picture. During system design, the system integrator performs a brokering activity based on "how" an application performs its functions as well as "what" domain tasks the application provides. Thus, an application must also be characterized by "application requirements," including specific implementation features and infrastructure needs.

The ASERT tool allows an application to be registered against both DRACs and IRACs. Figure 16 depicts both the "what" and "how" registration mappings for the *FDManager v2.0* and *ResponderLocator v1.0* applications listed above. Registering against a DRAC specifies "what" an application does; while registering against an IRAC specifies "how" the application does it. As described in Section 0, SEPA's IRAC ontology is used to represent domain-independent knowledge about application requirements. Following the separation process aided by the SEPArator, the METeR tool assists system stakeholders in representing application requirements, originally from the UKM, using the IRAC ontology (e.g. task performance constraint and system constraint requirements from KA session 3). The IRAC ontology is used to represent *both* the application requirements expressed by stakeholders *and* the specifications of an application being registered through ASERT.


DRAC: Incident Manager
End-User Application: FD Manager 2.0
Assign Crew to Sector
Manage Sector Location
End-User Application: Responder Locator 1.0
Assign Crew to Sector
IRAC: Software Application
Name: FD Manager 2.0 executable
IRAC: Operating System
Type: Microsoft Windows 32-Bit OS
IRAC: Computing Platform
Processor Type: Intel Pentium Class
Processor Speed: ≥ 90 MHz
Main Memory: ≥ 32 MB
Disk Storage: ≥ 5 MB
Display Device: 640x480x256
Related Infrastructure Constraints
IRAC: Software Application
Name: Responder Locator 1.0 executable
IRAC: Runtime Library
Name: XML Library 1.1.1
IRAC: Middleware Data Handler
Name: Incident Middleware 0.8
IRAC: Runtime Environment
Type: Java2 Runtime Environment
Related Infrastructure Constraints
Figure 16: Application Registration Example

With an understanding of the domain and application requirements for a particular installation, the system integrator performs the activity of system design by selecting specific applications that together comply with the client's installation requirements.

As evident from the task decompositions generated from KA sessions 1 and 2, the SEPA process must be capable of specifying two incident management configurations capable of satisfying two very different installations: City X and City Y. To accommodate the requirements of each installation from a single set of

registered applications, SEPA's separate domain and application requirements representations are used to aid the designer in identifying potential system designs and candidates for reuse.
Beginning with a blank slate, system design typically starts with a questions such as "What domain tasks should be automated at this installation?" SEPA tools help the designer to answer this question independently of implementation concerns by perusing the domain model and/or reference architecture to identify candidate tasks. Browsing the reference architecture is often preferred since services are likely organized into DRACs that correspond to familiar domain roles (e.g. Incident Commander role, HazMat Specialist role).
For City X, two required domain functions documented in KA session 1 are:

- Manage Sector Location
- Capture Weather Information

Once the respective set of DRACs has been identified, the Requirements Integration and Verification Tool (RIVT) can be used to determine registered applications capable of performing the selected domain tasks. Based on the domain services selected for City X, candidate applications suggested are:

- Manage Sector Location - *FDManager v2.0* OR *ResponderLocator v1.0*
- Capture Weather Information - *WeatherMonitor 0.7*

For City X, neither task constraints nor system constraints have been specified. Thus the designer may select either *FDManager v2.0* or *ResponderLocator v1.0* to provide the "Manage Sector Location" service. The designer narrows the set of candidate applications by imposing implementation requirements. These requirements (constraints) are expressed in the same language as the application registered infrastructure, the IRAC ontology. Once the designer has successfully satisfied the domain and application requirements associated with each application, the brokering process continues by evaluating (i) the compatibility of the application collection against the system environment and (ii) the mutual compatibility of the applications. Specific IRAC information relevant to this example includes the following:

- *FDManager v2.0*
  - Processor Type: Intel
  - Processor Speed: ≥ 90 MHz
  - Disk-space: ≥ 5 MB
  - Memory: ≥ 32 MB
  - Display Size: 640x480x256
  - requires-OS: { Windows NT Workstation v4.0 SP4 | Windows 95 | Windows 98 }
- *ResponderLocator v1.0*
  - requires-OS: { Windows NT Workstation v4.0 SP4 | Windows 95 | Windows 98 }
  - depends-on-application-framework: *IncidentMiddleware v0.8*
  - uses-system-library: *XMLLibrary v1.1.1*
  - executes-in-runtime-environment: *JavaRuntimeEnvironment v1.2*
  - depends-on-application-framework: *IncidentMiddleware v0.8*
- *WeatherMonitor 0.7*
  - requires-OS: { Windows NT Workstation v4.0 SP4 | Windows 95 | Windows 98 }
  - executes-in-runtime-environment: *JavaRuntimeEnvironment v1.2*

In addition to highlighting possible incompatibilities among applications, the integration process also highlights the need for secondary supporting applications, such as databases, word processors, and web servers. These related infrastructure applications may, in turn, have further infrastructure needs. For example, the infrastructure application *IncidentMiddleware v0.8* required by *Responder Locator v1.0*, requires the storage repository *IncidentRepository v2.0.*

- In contrast to the City X installation, options for the City Y installation are constrained by the requested map-based user interface and the required operating system (Windows NT) based on KA session 3. Applications used in the City X installation are reused for the City Y installation based on their registration to required tasks, providing they satisfy the stated application requirements. These requirements help narrow the set of feasible application, resulting in selection of *Responder Locator v1.0* to provide the "Assign Crew to Sector" task, since the task is constrained by the requirement for map-based application.

SEPA Activity: Requirements Analysis
Supporting Tool: Requirements and Integration Verification Tool (RIVT)

In addition to supporting the system configuration process, RIVT also provides a query facility to interrogate the information contained in IRA, DRA, and registration representations to support impact and reuse analysis from a variety of stakeholder perspectives.

Impact analysis for new application development

In a domain as complex as Incident Management, the domain scope modeled will likely need to grow to satisfy a larger customer base. Through KA, new domain tasks are added in context with existing functionality. Relationships to existing tasks are determined during knowledge model merging and filtered down to the RA as changes in data and event exchange between DRAC services.

When new applications are under consideration, an initial analysis of the Reference Architecture and currently registered applications can provide information regarding the degree to which a proposed application will affect or be affected by other applications. Functionality to be provided by the new application is identified in the RA. If the domain task has not yet been represented in the RA, additional KA yields an expanded DM and results in new RA services. Likely interaction among applications is evident through data and event exchange between the DRACs that provide the functionality under consideration. These interactions can identify (i) other applications already registered which may require modification to correctly interface with the new application and (ii) functionality which has not yet been automated but must now be automated if the functionality under consideration is to be included in the system design.

Requirements analysis to support reuse of existing applications

The SEPA process and tools enable reuse of requirements through requirements modeled in a computational representation. Application implementation and domain requirements represented in SEPA can be used in combination to determine the likelihood of reuse.

As in the brokering examples described in the prior section, the first step in satisfying the needs of an installation site is determining which domain tasks are to be supported. Through application registration against the RA, these domain tasks reference applications that become candidates for reuse. To identify domain functionality of interest, the RIVT user can take a number of approaches in posing questions to the RA.

Users focused on particular domain data elements: Data elements represented in the Domain Model are "owned" by DRACs in the RA. RIVT can be used to determine what DRAC services utilize those data elements and what applications are registered to those services.

Users familiar with high-level domain tasks that are to be automated: High-level tasks in the DM identify lower level tasks that are satisfied by DRAC services in the RA. RIVT can be used to determine what applications are registered to those tasks/services.

Users intending to automate the tasks for a particular domain role (performer): Tasks performed by a domain role (e.g. Incident Commander) are identified in the DM; those tasks are satisfied by services in one or more DRACs. Furthermore, DRACs often closely align with performer roles, thus collecting the services associated with a role. RIVT can be used to determine what applications are registered to the service associated with DRACs.

Once a selection of candidate applications is identified for reuse, infrastructure requirements associated with registered applications provide additional information for determining level of reuse. Questions that can be answered using the SEPA IRAC representation include:

- What types of resources does the application require (e.g. hard disk, memory, peripherals)?
- Does the application run in the chosen runtime environment (e.g. Java, Visual Basic for Applications)?
- Does the application run under the chosen operating system?
- Will an application conflict with other candidate applications if integrated in a single installation (e.g. the sum of memory required by two applications exceeds that offered by the destination server)?
- Considering the pool of available application developers, is the application developed using skills developers possess and technologies they are familiar with (e.g. Java, Visual Basic)?

Figure 112 depicts the result screen for a selected query in RIVT designed to answer these types of questions. This screen provides the developer, end-user, or integrator with a high-level picture of the relationships between RA services (domain tasks), DRACs, registered applications, and installation sites. In the example shown, the service "Manage Sector Location" has been selected. The DRAC offering this service is highlighted, *Incident Manager*. In the third column, applications registered to the *Incident Manager* are highlighted, *FDManager v2.0* and *ResponderLocator v1.0*. The installation sites requiring this service are highlighted, including both City X and City Y.

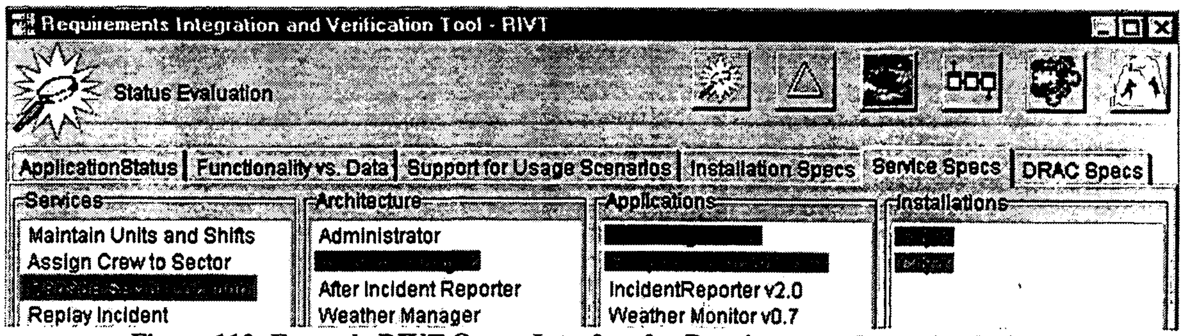


Figure 112: Example RIVT Query Interface for Requirements Reuse Analysis

CONCLUSION

Effective requirements management is a significant ingredient in ensuring successful software projects and the long term reuse of code, design, and modeling artifacts. However, the requirements process becomes increasingly complex based on project size, the number of expert viewpoints, the variety of requirements classifications (e.g. functional, non-functional, project), and the different demands placed on software artifacts by system stakeholders. A breakdown in this process can both hinder current development efforts and limit possibilities for future reuse.

The Systems Engineering Processing Activities (SEPA), under research 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 a comprehensive methodology and a suite of supporting tools that emphasizes requirements analysis prior to implementation. This paper highlighted several salient issues regarding effective requirements management and refinement. Additionally, the SEPA methodology and tool suite is presented in the context of a project in which it is currently being applied. The following issues (indicated in bold text) summarize how the SEPA process and supporting tools address these issues:

Traceability: Attempting code reuse through requirements is exceptionally difficult without an accurate traceability mechanism from knowledge sources, through interim models, to system architecture, to system design. By emphasizing traceability throughout the requirements analysis and refinement process, SEPA is capable of supporting queries against artifacts (models) and discovering relationships among artifacts. In addition to the standard benefits gained by accurate traceability to knowledge sources, the ability to navigate the SEPA traceability links improves the opportunities for stakeholders to discover information about one model in terms of another. For example, a domain expert may be more familiar with high-level domain tasks than with specific applications. To discover applications associated with domain tasks, the tasks residing in SEPA's Unified Knowledge Model (UKM) can be traced to Domain Reference Architecture services, which can subsequently be traced to related applications.

Support for Multiple Viewpoints: Obtaining the viewpoints provided by multiple experts is critical to acquire sufficient breadth for domain modeling and system implementation requirements. However, developers must refer to a common picture of the domain to produce a software system with correctly cooperating application components that fulfill designated functions in the domain. Contrasting developer needs, other stakeholders (e.g., end-users, domain experts) representing individual domain viewpoints want to be assured either that their interests are being reflected in the resulting system or that there is adequate justification for why their interests have not been taken into account. The knowledge synthesis process supported by SEPA's HyDRA tool balances these demands by guiding the analyst during the synthesis process yet retaining and linking to knowledge originally stated by the stakeholders. HyDRA captures the original results from each KA session in a separate model space (a collection of knowledge models). As artifacts from KA sessions are synthesized into a Unified Knowledge Model (UKM), traceability is retained and the rationale for each KE decision during model synthesis is recorded. One benefit afforded by this approach is that even after the unified model is created, an expert can continue to browse domain tasks in their own terminology and referencing unified tasks via traceability links. Furthermore, the HyDRA synthesis process follows a "viewpoint hierarchy," *first* unifying requirements from stakeholders holding the *same* viewpoint. This retains separate model spaces representing the functional and data requirements associated with each viewpoint.

Computational Requirements Representation: SEPA's ability to reason about requirements throughout the lifecycle is grounded in a series of related computational requirements representations. Beginning with individual graphical and textual Knowledge Models (KMs) transcribed by Knowledge Engineers (KEs) from KA sessions, SEPA aids the analyst in transitioning from one representation (e.g., Unified Knowledge Model) to the next (e.g., Domain Model, Domain Reference Architecture) as refinement proceeds, helping to ensure consistency and completeness in the models at each stage. Among other benefits, a computational representation increases the ability to answer questions posed by system stakeholders, including determining if a resulting software system meets expectations. Without a computational representation, it is significantly more difficult to associate application technology features to originally stated requirements. For example, suppose a text requirement stated "The application supporting the billing process shall run on a Windows NT machine with no more than 128MB memory." Breaking this statement into a flat, "text fragment" representation leads to some difficulty for tool supported automated reasoning: (i) The statement actually suggests two different requirements: "Windows NT" and "less than 128 MB." (ii) To determine if "128 MB memory" is satisfied, a numeric representation is required for proper comparison. (iii) Both these requirements act as constraints on the "billing process" task, and any application registered to the "billing process" task should satisfy both the "Windows NT" and "less than 128 MB" requirements.

Separation between Domain and Application Requirements: To improve opportunities for long-term reuse and clarify the analysis process, software process experts often recommend separating concerns during analysis, focusing on "what" a system must do independently of "how" the system should be implemented. SEPA *explicitly* recognizes this division and guides the analyst in making the separation. The resulting two models, the Domain Reference Architecture and the Infrastructure Reference Architecture, allow independent specification and analysis of "what" and "how" requirements. While the Domain RA contains functions and data defined in the context of a single domain, the Infrastructure RA is supported by a domain-independent ontology that provides consistency in representation across projects, domains, and implementations. This approach to "separate concerns" is particularly effective when configuring multiple systems to support the same domain. Applications are "registered" to both Domain RA functions and Infrastructure RA specifications. Thus, an particular application may be selected for a configuration based on its ability to perform domain tasks *independently* of its specific implementation features or required infrastructure. In many instances, multiple applications are registered as capable of providing the same Domain RA function. The consistent infrastructure representation allows applications to be selected based on a common implementation feature, independent of domain functions (e.g. find all applications that run on Windows NT). Once applications have been selected, the representation also simplifies design trade-off analysis between system configurations. The designer can compare and contrast configurations of application technologies according to their ability to (i) meet domain functionality/data requirements, (ii) satisfy site implementation-specific requirements, and (iii) integrate and install as a coherent system.

Comprehensive Process Support for Requirements Refinement and Management: Software analysis and design methodologies typically suggest multiple phases during the requirements analysis process (e.g. gathering, modeling, refinement). However, the practical transition between such phases poses a significant challenge. In a practical setting, an analyst may apply different tools and/or customized approaches at each phase. As a result, the analyst's productivity is hindered by the difficulty in bridging processes and tools. The SEPA methodology and tool suite is designed to guide each phase of requirements management and refinement and to transition seamlessly between phases. Beginning with a series of Knowledge Models (KMs) that capture the requirements expressed by multiple stakeholders, SEPA aids the synthesis into a single Unified Knowledge Model (UKM). The RARE tool then guides the derivation of a Domain Reference Architecture from functional and data information in the UKM based on a set of goals prioritized by the architect. The METeR tool helps the analyst model application requirements, identified in the KA and modeled in the UKM, using a domain-independent infrastructure ontology. These models are then considered in combination by the RIVT tool during system configuration and requirements interrogation. Complementing the support for transition between artifacts, SEPA's common traceability representation between artifacts provides the benefits previously mentioned.

Support for Requirements Volatility: The processes of requirements gathering, modeling, and refinement are inherently iterative. To maintain integrity between representations, traceability links and respective model elements require maintenance as requirements evolve and information is added, changed, and deleted. Thus, a requirements methodology and supporting tool suite must be accompanied by procedures for reanalyzing existing modeled information in light of new contributions and modifications. In particular, changes to an artifact must be appropriately reflected in artifacts preceding *and* following the respective artifact. Current SEPA research is focusing on forward propagation. While back-propagation has been recognized as an important feature, this research has just begun in earnest.
SEPA is currently being applied to complex projects involving large numbers of stakeholders holding many different perspectives. Focusing on one of those projects, the example in this paper illustrated how SEPA is contributing to code and artifact reuse through requirements and provided evidence regarding SEPA's support for the issues outlined above. As research and tool development proceed, additional experiments are planned to evaluate SEPA's ability to support requirements management as domain knowledge and stakeholder demands evolve over an extended period.

REFERENCES

Barber, K. S., & Jernigan, S. R. (1999, June 28-July 1). Changes in the model creation process to ensure traceability and reuse. Paper presented at the **International Conference on Artificial Intelligence**, Las Vegas, Nevada.

Braun, C. L. (1999). A Lifecycle Process for the Effective Reuse of Commercial Off-the-Shelf (COTS) Software. Paper presented at the **Symposium on Software Reusability** (SSR99), Los Angeles, CA.

Caldieri, Gianluigi, & Basili, V. R. (1991, February 1991). Identifying and Qualifying Reusable Software Components. **IEEE.**

Chipware, Inc. (1999). icCONCEPT RTM- Requirements & Traceability Management Tool, [Web site]. Marconi Systems Technology, Inc. Available: http://www.mstus.com/.

Christel, M. G., & Kang, K. C. (1992). Issues in Requirements Elicitation (**Technical CMU/SEI-92-TR-12**). Pittsburg, Pennsylvania: Carnegie Mellon University.

Department of Defense, U. S. (1996). Guidelines for Successful Acquisition and Management of Software-Intensive Systems : Department of the Air Force, Software Technology Support Center.

Dobson, J. E., Blyth, A. J. C., Chudge, J., & Sterns, M. R. (1992, September). The ORDIT Approach to Requirements Identification. Paper presented at **the Sixteenth Annual International Computer Software and Applications Conference.**

Gotel, O. C. Z., & Finkelstein, A. C. W. (1994). An Analysis of the Requirements Traceability Problem. Paper presented at the **First International Conference on Requirements Engineering**, Colorado Springs, CO.

Graham, I. (1995). Migrating to object technology. Wokingham, England ; Reading, Mass.: Addison-Wesley Pub. Co.

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

Harbison, K. (1997). Scenario-based Engineering Process . http://caesar.uta.edu/caesar/process.html: Center for Advanced Engineering Systems and Automated Research, The University of Texas at Arlington.

Kotonya, G., & Sommerville, I. (1997). Requirements engineering with viewpoints. In R. H. Thayer & M. Dorfman (Eds.), **Software Requirements Engineering** (pp. 150-163). Los Alamitos, CA: IEEE Computer Society Press.

McGraw, K., & Harbison, K. (1997). **User-centered Requirements**. Mahwah: Lawrence Erlbaum Associates, Publishers.

Mullery, G. (1979). CORE - a method for controlled requirements specification. Paper presented at the **Fourth International Conference on Software Engineering**, Munich, Germany.

Oracle. (1999). Oracle Designer . http://www.oracle.com/tools/designer/: Oracle Corporation.

Palmer, J. D. (1997). Traceability. In R. H. Thayer & M. Dorfman (Eds.), **Software Requirements Engineering** (pp. 364-374). Los Alamitos, CA: IEEE Computer Society Press.

QSS. (1998). DOORS 4.0 . http://www.qss.co.uk/DOORS4/: Quality Systems and Software.

Rational. (1998). Software Development, Component-Based, Programming Tools: Rational . http://www.rational.com/: Rational.

Riel, A. J. (1997). **Object-Oriented Design Heuristics**. Reading, MA: Addison-Wesley.

SES. (1999). SES/objectbench: object-oriented analysis, simulation and code-generation. .

Sommerville, I., & Sawyer, P. (1997). **Requirements engineering : a good practice guide**. New York: Wiley.

Sommerville, I., Sawyer, P., & Viller, S. (1998, April 6-10, 1998). Viewpoints for requirements elicitation: a practical approach. Paper presented at the **Third International Conference on Requirements Engineering**, Colorado Springs, Colorado.

Technologies, T. (1999). Slate REquire. Available: http://www.tdtech.com/.

Tracz, W. (1991, November 18-20). An Outline for a Domain-Specific Software Architecture Engineering Process. Paper presented at the **Fourth Annual Workshop on Software Reuse**, Reston, VA.

Tracz, W., Coglianese, L., Young, P. (1993). A Domain Specific Software Engineering Process Outline. **ACM SIGSOFT Software Engineering Notes**, 18(2), 40-49.

Tracz, W. (1995). DSSA (Domain Specific Software Architecture) Pedagogical Example. **ACM SIGSOFT Software Engineering Notes**, 20(3), 49-62.

Verilog. (1997). Verilog ObjectGEODE . http://www.verilogusa.com/home.htm.