Extending FeatuRSEB with Concepts from Systems Engineering

John Favaro and Silvia Mazzini

Intecs SpA
via E. Giannessi, 5y – I-56121 Pisa, Italy
{John.Favaro,Silvia.Mazzini}@intecs.it

Abstract. FeatuRSEB is a method for domain modeling of software system families using the industry standard notation of the Unified Modeling Language. FeatuRSEB/Sys is an extension of FeatuRSEB with constructs from the SysML profile for systems engineering, augmenting it with analysis and modeling mechanisms that are upstream from the original method, while nevertheless conserving the advantages of an industry standard notation and semantics.

Keywords: Domain analysis, reuse, feature, UML, SysML, systems engineering, requirements, analysis, modeling, architecture, trade studies.

1 Introduction

James Neighbors introduced the term *domain analysis* into the literature in 1984 in an article [1] on his pioneering Draco system. Another milestone occurred in 1990 with the introduction [2] of Feature-Oriented Domain Analysis (FODA). FODA became a catalyst for the development of subsequent methods due to a number of characteristics including a straightforward scheme for capture and classification of domain knowledge that is easily understandable by humans. These characteristics have led to the development of several FODA variants over the years [3].

The FeatuRSEB [4] variant of FODA is a feature-oriented extension of an application family development method, the Reuse Oriented Software Engineering Business (RSEB) [5], arising out of collaboration between Hewlett-Packard Research and Intecs. It has two characteristics of particular interest:

- It was one of the first domain analysis methods to employ the notation of the Unified Modeling Language [6]. Since then, other feature-oriented methods using UML notation have appeared [7]. These approaches have the advantage that an industry-standard modeling notation is used instead of an *ad hoc*, proprietary notation, and the semantics of core modeling concepts (such as classes and relationships) are not interpreted in an *ad hoc* fashion. This makes it easier to focus on the value-adding characteristics of the method.
- Due to its origins in the Reuse Driven Software Engineering Business, non-technical, business-oriented factors are given more explicit emphasis with respect to purely technical factors;

S.H. Edwards and G. Kulczycki (Eds.): ICSR 2009, LNCS 5791, pp. 41–50, 2009.

Intecs works with clients in the aerospace, automotive, space, and telecommunications industries, where we advise them on development methodologies and provide specialized toolsets such as HRT-UML for the modeling of hard real-time systems [8]. This work has exposed us to the discipline of *systems engineering*, which is central to the development of large systems in those industries. Based upon this experience, we have extended FeatuRSEB with modeling concepts and constructs from the Systems Modeling Language (SysML) profile [9] of the UML in order to extend its capabilities for high-level analysis for domain modeling of large software system families, while preserving the advantages of an industry-standard notation and semantics.

2 Relating Systems Engineering to Domain Engineering

Systems engineering as a recognized discipline dates back to the 1950s, and has been applied with success to large and complex systems in contexts ranging from defense and air traffic control to commercial products. Systems engineering essentially consists of three major concurrent activities (**Fig. 1**).

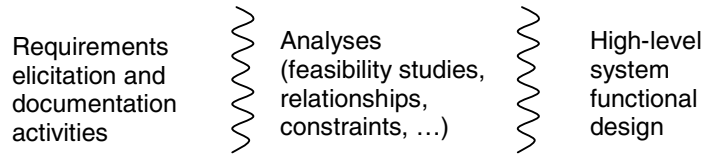

Fig. 1. Three concurrent activities in systems engineering

- Requirements elicitation and documentation is the core activity of systems engineering.
- Numerous types of analyses are carried out, related to the specific disciplines involved, and may include engineering and economic feasibility studies, documentation of relationships among components, and identification and documentation of physical constraints on system elements. In particular, they may include calculations and formulas related to non-functional requirements.
- High-level system functional analysis identifies the principal subsystems, their interfaces, and dataflows through those interfaces.

Although each of these activities superficially appears to have a direct counterpart in the software engineering life cycle, their nature and purpose are different in systems engineering. Requirements are first-class citizens in systems engineering (in the sense that they are elicited, documented, and managed as separate entities) because they are an *output* of the systems engineering process, whereas in software engineering processes, requirements tend to be managed as an *input*. In fact, it would be only a slight exaggeration to state that systems engineering is mostly concerned with eliciting requirements of systems and about analyzing the feasibility of designs.

Requirements are elicited with the support of *analyses* that in turn elicit relationships and constraints among different parts of the system. The documentation of these analyses, trade-off studies, and associated decisions is likewise considered a first-class citizen in systems engineering, rather than an afterthought as so often happens in

software systems design (often something as poor as a simple text field to "describe rationale in 256 characters or less"). This elevation of rationale and decision-making documentation to first-class citizenship is in line with recent awareness in the software engineering community of its importance, e.g. through the addition of the Decision View in the standard *4+1 views* model of software architectural description [10]. In domain analysis it is even more important, where the recording of trade-off and decision-related information for entire families of systems is crucial for informing the feature selection process for subsequent domain engineering of systems.

The term "functional analysis" in systems engineering has a more general meaning than in software engineering. This is so in large degree because it is essential for determining subcontracting boundaries: systems engineers construct the *product tree* of all identified components of the system (including the software component) and then each such component is contracted out to others to construct (it is not unusual for a systems engineering organisation to have no construction capability at all).

Requirements elicitation, analysis and decision-recording, and system component management mechanisms are much less well developed in software engineering processes because they are *upstream* from where software engineering processes generally start. Indeed, in the ECSS-E40 standard of the European Space Agency for software engineering [11], systems engineering activities are explicitly defined to be precedent to the software engineering activities.

This is where the relationship of system engineering to domain engineering emerges: domain engineering is *also* upstream from the traditional (single-system) software engineering process. The outputs of domain engineering *also* serve as inputs to the activities for engineering (single) software systems. Therefore, in retrospect it is not surprising that we found precisely these mechanisms from systems engineering to add significant value to the domain modeling process.

3 Introducing SysML into Domain Analysis: The Domain Models

The table in the Appendix summarizes the models and approach of FeatuRSEB/Sys, describing also the ways in which each model handles variability. In this section we discuss in more depth those models which integrate systems engineering concepts.

3.1 Context Model

Context analysis was introduced in original FODA. The purpose of the context model is to define the scope of the domain that is likely to yield exploitable domain products. FeatuRSEB/Sys context diagrams are represented as SysML *block diagrams*. Standard UML is biased toward the concrete development of software systems: it naturally forces thinking in terms of objects and methods, quickly driving toward design. (Indeed it has been suggested by the inventors of FODA that it has been pushed beyond its original intention of analysis into design). At the highest levels of domain engineering, though, *interfaces* are much more important than specific "methods" on classes. With SysML blocks (thought of as "whited-out classes") the domain analyst is freed from the "tyranny of the methods." As in other FODA

variants, context analysis provides the high level connections among major components related to the target domain. The exchange of information (dataflow) with external actors (peer domains) and lower domains is indicated at block boundaries by means of SysML *flow ports*. The SysML blocks and flow ports help to pull the context analysis up to a higher level of abstraction more suited to domain engineering.

3.2 Domain Dictionary

The domain dictionary consolidates the definitions of all terms used in the domain and is updated over time. In FeatuRSEB the feature model did "double duty" as a domain dictionary, and this is also true in FORM [12] to a degree. In FeatuRSEB/Sys the domain dictionary is separate and explicit, serving the same role as in systems engineering of establishing a common basis for communication among all participants in the domain implementation activities.

3.3 Domain Requirements Model

In FeatuRSEB the use case model alone represented user requirements, relieving the feature model of FODA from doing double duty to represent both user requirements and reuser features. In FeatuRSEB/Sys a similar phenomenon occurs: the use case model is relieved from representing both use scenarios and *implicitly* the requirements they embody, by adding a requirements model to model the requirements *explicitly*.

Requirements reuse has received some attention in the literature [13], but its integration into domain analysis methods tends to take the form of implicit inclusion in domain use case models and to some extent feature models. We argue that a separate domain requirements model is a useful addition to the set of domain analysis models, for at least two reasons: first, although use case models are often referred to as "requirements models", we subscribe to the point of view currently argued by many that use cases are at a lower level of abstraction than the requirements themselves – rather, use cases generally represent the results of requirements *analysis*; second, an explicit requirements model is particularly useful for documenting many types of technical, organizational, economic, and legal non-functional requirements, which are not handled well by use case techniques with their bias toward functional requirements, but which a domain analysis must capture in order to be complete. In short, explicit modeling of domain requirements contributes to keeping the level of abstraction of the domain analysis as high as possible.

SysML provides an explicit requirements diagram (**Fig. 2**), which can serve as an important starting point for reusable requirements for new systems in a domain. For example, it gives the domain analyst a place to document the cost of implementing reusable requirements [14]. Furthermore, the requirements model allows requirements management tools to link from the requirements to their implementation in domain systems. This supports requirements traceability, and thereby provides support not only for the construction phase of domain engineering, but also for V&V, which has a heavier footprint in larger, more complex systems.

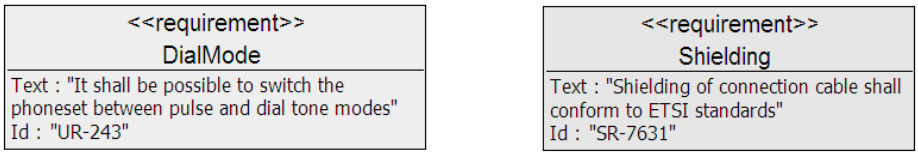

Fig. 2. Two requirements from the domain requirements model corresponding to the telecommunications example in [4]. The requirement on the right-hand side could not be easily captured and modeled using only use case techniques.

3.4 Analysis Models

In the UML, *analysis* has a rather narrow meaning, involving the transition from use cases to object design. FeatuRSEB inherited Jacobson robustness analysis from the RSEB for this purpose. But in systems engineering, analysis has a more general meaning, and is more about exploring feasibility, relationships and constraints at high system levels (relationships between large blocks of system functions), and in general doing supporting analysis to ensure that a system functional architecture is sound and cost-effective. In UML, there are no standard ways to document such characteristics. Some progress is being made with improved mechanisms for describing constraints, but in general the user is left to his own devices.

In SysML, a new diagram has been introduced solely for documentation of constraints and relationships among system parts. *Parametric diagrams* are used to capture the properties of the applications or subsystems of the domain. They can express powerful constraints that abstract away specific constraints of individual systems into more general constraints (e.g. expressed in terms of formulas) characterizing aspects of system families. For example, perhaps a «performanceRequirement» can only be satisfied by a set of constraints along with an associated tolerance and/or probability distribution, or a «functionalRequirement» might be satisfied by an activity or operation of a block. Likewise, cost constraints can be explicitly documented in parametric diagrams.

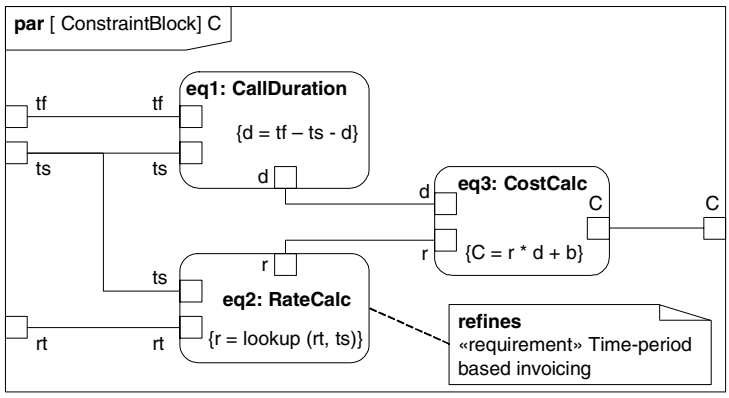

Fig. 3. Parametric diagram for billing feature (simplified) in model of [4], to support analysis and identification of input parameters that must be made available by a domain architecture to support implementation of a requirement for time-period based call invoicing

Parametric diagrams help the domain analyst to analyze constraints on the domain architecture needed to satisfy particular requirements. For example, in [4] one feature variant for the telephone call invoicing subsystem had a requirement to be able to support invoicing based upon time periods (e.g. "late night," "daytime," etc.). A parametric diagram (**Fig. 3**) helps document an analysis to identify the input parameters for such a feature that an architecture will have to support. Such an analysis supports refinement of the requirement and feasibility and cost studies.

Feature modeling permits the documentation of reusable functionality – but what about reusable "non-functionality"? Domain analysis methods have not traditionally provided strong support for a reuse-oriented documentation of non-functional aspects of system families. For example, the theory and methods used to analyze performance constraints in families of real-time systems form an important part of their construction and documentation. The *constraint blocks* of SysML make it possible to package and document non-functional aspects as reusable artifacts (**Fig. 4**). Using the SysML concept of *allocation*, variability can be modeled with such mechanisms – for example, two different algorithms could be *allocated* to different system variants.

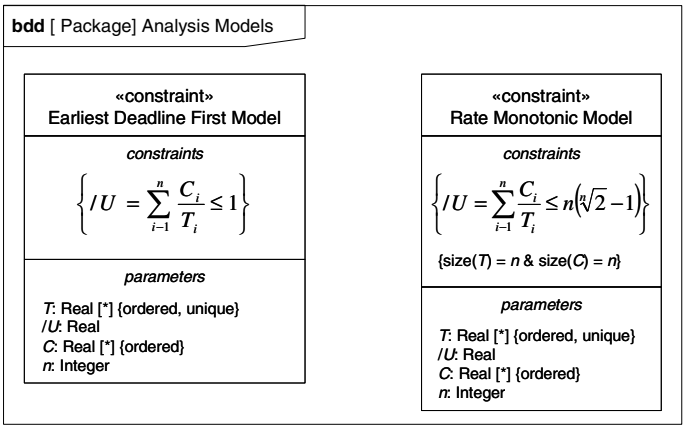

Fig. 4. Reusable analysis models for hard-real time system families packaged as constraint blocks

It is important to emphasize that, especially in a context of domain engineering of large and complex systems in the sectors in which our clients operate, those decisions which concern the scope of system families and the number of variations supported in a domain architecture are primarily *business* driven: considerations: the effort, time and costs required for domain design and implementation must be less than the effort, time and costs that would be required for the expected number of domain applications that may reuse the domain engineering results. Here FeatuRSEB/Sys inherits the business decision mechanisms from the Reuse Oriented Software Engineering Business upon which original FeatuRSEB was based. The difference is that the means to *document* these decisions is now provided in the new mechanisms from SysML. In particular, the parametric diagram mechanism supports the domain analyst in what in systems engineering are called *trade (or trade-off) studies*, whereby so-called *measures of effectiveness* ("moes") are defined within an *analysis context* (**Fig. 5**).

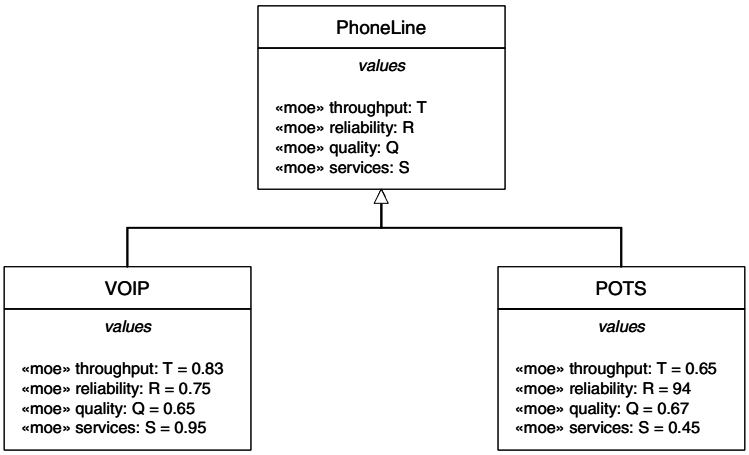

Fig. 5. SysML analysis context for *trade analysis* for the phone line quality feature variants in [4] according to selected measures of effectiveness («moe»). Here another mechanism for handling variability is illustrated: the `PhoneLine` abstraction captures the variability between VOIP and POTS alternatives. Different *objective functions* will place emphasis on different characteristics (e.g. superior service offering flexibility on VOIP line or higher reliability of POTS line), allowing the domain engineer to select feature variants based upon the needs of the particular system.

3.5 High Level Generic Architecture

SysML *block diagrams* are used to describe the top level partitioning of the domain system capabilities into functional blocks, as refinements of the context model structure diagrams; these blocks are defined as black boxes by putting the emphasis on provided and required interfaces. These interfaces do not require a full signature, but mainly describe named entities as SysML *dataflows*.

Dataflow entities exchanged between domains and data that are handled by top-level components (blocks) in the domain are described, categorized (by inheritance) and clustered (by aggregation) using object oriented technology, in order to provide their data structure. For this purpose UML class diagrams are used in a separate data package, thus describing the data model for information exchanged in the domain. The package may correspond to a library of data types available at domain level.

The *functional model* identifies the functional commonalities and differences of the applications in the domain. The elements of the specification of a functional model can be classified in two major categories: First, *specification of functions* describes the structural aspects of the top level partitioning of the applications in terms of inputs, outputs, internal data, logical structures and data flow among them. Second, s*pecification of the functional and dynamic behavior* describes how an application behaves in terms of events, inputs, states, conditions and state transitions, sometimes defined as *control flow*. Activity diagrams and UML StateCharts are generally used to describe functional models, but other discipline-specific methods and languages may also be used, such as those used to model mechanical or control aspects of systems.

3.6 Feature Model

The feature model in FeatuRSEB/Sys remains the "+1" model as described in original FeatuRSEB, summarizing the results of domain analysis and tracing the features back into the respective domain models, whereby the traces are extended to the other, additional models that have been introduced (e.g. the requirements model). In original FeatuRSEB a notation for the feature model was devised by extending the UML with stereotypes for describing the features and their relationships. This notation remains valid for FeatuRSEB/Sys, retaining the advantage of using the UML.

4 Conclusions and Further Work

Systems engineering places emphasis on activities that are upstream from those of software engineering, whereby outputs considered valuable include requirements, feasibility analyses and trade studies, and high level functional breakdown at a higher level of abstraction than in software system architecture. The same is true of domain engineering, and the new concepts and diagrams of SysML help to raise the level of abstraction in domain analysis back to where it was originally conceived. It also provides another advantage: the feature model in the feature-oriented approaches is in its essence a mechanism for organization and *traceability*. Systems engineering provides extended mechanisms for traceability – of requirements, of analyses, and the like – that make it possible to broaden the scope of traceability and provide a more complete documentation of the domain analysis.

FeatuRSEB/Sys originated in the COrDeT project of the European Space Agency, where it was used in the modeling of onboard Space system families [15]. Related efforts in modeling techniques are underway in other contexts – for example, the EAST-ADL2 profile for an automotive architectural modeling language that incorporates concepts and mechanisms from SysML [16].

Tool support is an important aspect of feature-oriented domain modeling, and significant progress has been made since its introduction [17]. The choice of UML and SysML is important not only for notational uniformity and familiarity, but for its implications for tool support. The combination of the improved UML Meta Object Facility (MOF) and its support by the Eclipse framework has made model based development of specialized tools feasible with reasonable costs. The provision of complete tool support for FeatuRSEB/Sys will be the focus of future efforts.

References

1. Neighbors, J.: The Draco Approach to Constructing Software from Reusable Components. IEEE Trans. Software Eng. 10(5), 564–574 (1984)
2. Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21. Software Engineering Institute, Carnegie Mellon University, Pittsburgh (1990)
3. Frakes, W., Kang, K.: Software Reuse Research: Status and Future Directions. IEEE Transactions on Software Engineering 31(7) (July 2005)

4. Griss, M., Favaro, J., D'Alessandro, M.: Integrating Feature Modeling with the RSEB. In: Fifth International Conference on Software Reusability, pp. 76–85. IEEE Press, New York (1998)
5. Jacobson, M., Griss, M., Jonsson, P.: Software Reuse: Architecture, Process and Organization for Business Success. Addison-Wesley-Longman, Menlo Park (1997)
6. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual, 2nd edn. Addison-Wesley, Boston (2004)
7. Gomaa, H.: Designing Software Product Lines with UML: From Use Cases to Pattern-based Software Architectures. Addison-Wesley, Boston (2004)
8. Mazzini, S., D'Alessandro, M., Di Natale, M., Domenici, A., Lipari, G., Vardanega, T.: HRT-UML: taking HRT-HOOD into UML. In: Proc. 8th Conference on Reliable Software Technologies Ada Europe (2003)
9. The Object Management Group, Systems Modeling Language (OMG SysML), Final Adopted Specification, document ptc/06-05-04 (2006), `http://www.omg.org`
10. Kruchten, P., Capilla, R., Dueñas, J.C.: The Decision View's Role in Software Architecture Practice. IEEE Software 26(2), 36–42 (2009)
11. European Space Agency, ECSS-ST-40C – Software General Requirements (March 6, 2009), `http://www.ecss.nl`
12. Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: a feature-oriented reuse method with domain-specific reference architectures. Annals of Software Engineering 5(1), 143–168 (1998)
13. Lam, W.: A case-study of requirements reuse through product families. In: Annals of Software Engineering, vol. 5, pp. 253–277. Baltzer Science Publishers, Bussum (1998)
14. Favaro, J.: Managing Requirements for Business Value. IEEE Software 19(2), 15–17 (2002)
15. Mazzini, S., Favaro, J., Rodriguez, A., Alaña, E., Pasetti, A., Rohlik, O.: A Methodology for Space Domain Engineering. In: Proceedings Data Systems in Aerospace (DASIA), Palma de Majorca, Spain (2008)
16. Lönn, H., Freund, U.: Automotive Architecture Description Languages. In: Navet, N., Simonot-Lion, F. (eds.) Automotive Embedded Systems Handbook. CRC Press, Boca Raton (2009)
17. Antkiewicz, M., Czarnecki, K.: FeaturePlugin: Feature Modeling Plug-in for Eclipse. In: Eclipse 2004: Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange, OOPSLA, pp. 67–72. ACM Press, New York (2004)

Appendix: Summary of FeatuRSEB/Sys Models and Method

Domain model components	Sub-components and Notation(s)	Utilization
Context model	Context diagrams – SysML block diagrams	Positioning of the target domain with respect to higher, peer and lower domains. Variability expressed through different diagrams
	Use case diagrams – SysML use case diagrams	Description of domain capabilities. Variability expressed through different diagrams
Domain dictionary	Text	Updated throughout the domain analysis
Requirement model	Requirement diagrams – SysML Requirement diagrams	Link to Use cases and Blocks. Required variability has to be explicitly specified by link to related use cases and/or blocks and by link or requirement annotation.
High level generic architecture	Top-level partitioning (into applications) – SysML Block diagrams	Refinement of the context model structure diagrams
	Data package (library of data types/structures) - UML class diagrams	Data flow entities exchanged between domains and data handled by top level components are described, categorized (by inheritance) and clustered (by aggregation)
	Application capabilities Application Use cases – SysML Use case diagrams Application Behaviors – SysML Interaction (sequence) diagrams	Refinement of context level Use cases
	Functional model Specification of functions – SysML activity diagrams Specific of dynamic behavior – SysML StateChart Parametric diagram – SysML Parametric diagram	Structural aspects of top level partitioning (inputs, outputs, internal data, logical structures and data flow among applications). Application behavior (events, inputs, states, conditions and state transitions) Capture functional and non functional requirements through formulation of constraints. Document trade-off studies on architectural variants, including cost and performance factors.
Feature model	Feature model – UML notation from FeatuRSEB	Synthesis of domain analysis variability as emerging from the above models.