Semantic decoupling: reducing the impact of requirement changes

Semantic Decoupling: Reducing the Impact of Requirements Changes

Israel Navarro, Nancy Leveson, Kristina Lundqvist
Massachusetts Institute of Technology
Room 33-334
Cambridge, MA 02139

Keywords: Software changeability, software traceability, coupling, software requirements

ABSTRACT

The continuous stream of requirements changes that often takes place during software development can create major problems in the development process. This paper defines a concept we call *semantic coupling* that along with features of intent specifications can be used during system design to reduce the impact of changing requirements. Traceability matrices are used to evaluate the sensitivity of a given design to requirement changes. The practicality of using the approach on real software is demonstrated using the intent specification of the control software for a NASA robot designed to service the heat resistant tiles on the Space Shuttle.

1. INTRODUCTION

Requirements changes cause havoc in a development process when the effects of the changes ripple through a large part of the system and software design, which then affects other parts until the overall design and conceptual model may be affected and start to degrade. Hopefully, the highest-level, most abstract requirements will not change, but sometimes they do, as system requirements become better understood. Reversals in TCAS (an airborne collision avoidance system required on almost all commercial aircraft in the U.S.) are an example of this problem. About four years after the original TCAS requirements specification and system design were completed and TCAS development was at the point of being tested on commercial flights, experts discovered that the design did not adequately cover the case where the pilot of an intruder aircraft does not follow his or her TCAS advisory, and thus TCAS must alter the advisory to its own pilot. This change in basic requirements (the need to reverse advisories) caused extensive changes in the TCAS system and software design, some of which introduced additional subtle problems and errors that took years to discover and rectify.

While the highest—most abstract and global—level requirements, such as detecting potential threats in TCAS, are less likely to change than lower-level requirements, when they do they have the most important (and costly) repercussions. A change at the highest level (or at any level) may require changes and reanalysis at all the levels below it. The impact of change in safety-critical systems can be extremely costly as changes can require a new system safety analysis. Any reduction in the effort required to accomplish this analysis will have a tremendous impact on cost and schedule.

We believe the reason requirements changes cause so much disruption to a project is rooted in the concept of *semantic coupling*. In general, tightly coupled systems are highly interdependent: Each part is linked to many other parts, so that a requirement change or functionality upgrade in one part (in terms of safety: a failure or unplanned behavior) can rapidly affect the others. Therefore, the key to reach a design that minimizes the effects of changes is the use of widely known modularity concepts applied at the upper level of the requirements definition phase. A decoupled design is therefore better able to manage the effects of requirement changes. In addition to this, the use of traceability matrices at the core of the design process improves the viability of the validation and verification (V&V) activities. These activities are fundamental to the software assurance tasks used in the development of safety-critical systems. Another benefit that can be derived from the application of the semantic coupling concept is the minimization of the number of common mode causes (a single error or failure having a cascading effect at system level). By restricting the functional dependencies between modules, one can limit (or at least identify and arrange for possible mitigations) the effects of having one module behaving in an erroneous or unexpected way.

This paper provides a review of related work in Section 2. Special focus is given to previous research on modularity, traceability and their application to the design of large complex software systems. Section 3 presents the basic concepts of Intent Specifications. This methodology provides a framework for the development of safety-critical systems. In Section 4, the semantic coupling concept is defined and a process outlined for identifying and reducing it. To determine the feasibility of applying the concept to real systems, we have applied the process to the Mobility and Positioning Software (MAPS) of a robot designed for NASA to service the heat resistant tiles on the Space Shuttle. The results of this study are described in Section 5. A brief description of future work planned to further develop and validate the approach presented herein is given in Section 6. Finally a summary of the most relevant conclusions is provided in Section 7.

2. BACKGROUND

2.1 Initial work on Software Modularity and Cohesion

The concept of software module coupling is not new; it has been used in software design for over 30 years to make software components more composable and reusable and to enhance the understandability of individual modules so they are easier to modify. In 1972 Parnas [37] presented the notion of information hiding in order to support the derivation of modular structures for software designs. The underlying principle behind this approach is that software can be made more adaptable by decoupling those design decisions that are likely to change. This highly influential idea has shaped the evolution of some of the most important software engineering advancements (abstract data type programming languages, object oriented design, etc.) during the last decades.

The need for these measurements of the “goodness”' of a design stem from the more general problem of complexity or intellectual manageability of the systems we are trying to build. Intellectual manageability can be enhanced by organizing information in a coherent, structured form, by using appropriate abstractions, and by reducing the dependencies among system components [34]. At the code level, this approach led to the concepts of coupling and cohesion.

The concept of module coupling characterizes a module’s relationship to other modules, that is, it measures the interdependence of two modules. The more connections between modules, the more dependent they are. Modularity and low coupling help confine the effects of changes to a single module. The counterpart to coupling is cohesion. A module could be said to have high cohesion if all the elements have a strong relationship to each other. Stevens and others [46][33][34][55] identified seven types of cohesion (presented in the order of increasing cohesion): coincidental, logical, temporal, procedural, communicational, sequential, and functional. These levels are defined based on certain associative principles (properties or characteristics) that relate the processing elements in a module. Some of these types of cohesion are related to functionality, while others are not. All have been defined rather vaguely, usually using examples alone.

Yourdon and Constantine [55] and Myers [34] identified three factors related to module coupling: the number of interfaces, the complexity of the interfaces, and the type of information flow (data or control) along the interfaces. These factors affect visibility outside the interface, the amount of information exported through interfaces, and the required interaction among programmers. Five types of module coupling have been identified, but in general modules are considered tightly coupled if they use shared variables or if they exchange control information. More recent software engineering attempts to define coupling have been limited to software module design and primarily to structural rather than semantic dependencies [1][3][35][42][52]. Most of this research has been directed at defining and generating metrics for module coupling.

2.2 Later Generalization of the Modularity Concept

The concept of modularity has attracted the interest from many engineering and management practitioners working in a wide variety of product development processes. Baldwin and Clark [4] observe that modularity is a financial force that can change the structure of an industry. Modularity makes this possible by (i) providing an effective division of cognitive labor [2], (ii) organizing and enabling parallel work [16], and (iii) allowing modules to be changed and improve over time with no effect on the overall system functionality. Considering the way in which humans interact with artifacts, Baldwin and Clark [5] identify three basic types of modularity: modularity-in-design, modularity-in-production, and modularity-in-use. Of these three, modularity-in-design is the most challenging to achieve but also the one providing the most beneficial economic advantages. This is explained by modeling the economic value of a new design as an option using the methods of modern finance theory. By applying modularity, a design goes from having one large design option (to accept the whole design or not) to having many smaller options (one per module). A modular design multiplies the valuable design options in a system and allows each module in the system to evolve in different ways.

The work by Steward [47] and later by Eppinger [15] [39] [44] focuses on refining the representation of the nature of design interactions and its components. This effort resulted in the development of the Design Structure Matrix (DSM) and analysis techniques to study task allocation, organizational structure, and appropriate coordination to improve product development. The DSM is used to represent the mappings related to the design of a complex system. These matrices have been used in numerous fields, including software design [10]. In this mapping technique, the system is first characterized by listing a set of design parameters for the system. The design parameters are then arrayed along the rows and columns of a square matrix. The matrix is filled in by checking, for each parameter, which other parameters affect it and which are affected by it. For example, if Parameter A affects the choice of Parameter B, then a mark “x” is put in the cell where the column of A and the row of B intersect. This process is repeated until all parameter interactions have been recorded. The result is a map of the dependencies that affect the detailed structure of the artifact. The matrix diagonal is completed with dots to indicate the obvious relationship of each design parameter with itself. An example of such DSM follows:

	DP 1	DP 2	DP 3
DP 1	.		
DP 2		.	x
DP 3			.

In this case, the design parameter 3 (DP 3) affects the choice of the design parameter 2 (DP 2). Thus, the DSM provides an immediate way to recognize interdependencies in a system.

In a typical DSM-based design exercise, all those design parameters closely related between each other will be grouped together to form clusters: the designers effort will then be focused on eliminating the interdependencies between those clusters. This is typically done by introducing the so-called design rules (DR1 in the example) as depicted in the following DSM:

	DR1	DP 1	DP 2	DP 3
DR1	.			
DP 1		.		
DP 2	x		.	
DP 3	x			.

The key idea is that a modular design can be obtained by fixing the value of the new parameter (i.e. the design rule DR1) in a way that permits the original parameters (DP2 and DP3) to be changed independently as long as they are changed in ways consistent with the new constraint. This approach is known as the Design Rule Theory (DRT).

A complex system design may go from being interdependent to being modular in the following way [5]: (a) The "architects" of the system must first identify the dependencies between the distinct components and address them via a set of design rules. (b) They must create encapsulated or "hidden" modules corresponding to the components of the system. And (c) they must establish a separate system integration and testing activity that will assemble the modular components and resolve unforeseen incompatibilities. A key benefit of systems with modular designs is that they can evolve, especially at the lower levels of the design hierarchy. The information that is encapsulated in the hidden modules can change, as long as the design rules are obeyed. In contrast, the design rules of a modular system, once established, tend to be both rigid and long lasting.

In the previous example, the design was solved by introducing a design rule that decouples the two modules: this is referred by Baldwin and Clark [5] as the *splitting* operator. Holland [20] and others have identified other generic actions that can be applied to an already modular system [48]: *augmentation*, which adds a module to a system, *exclusion*, which removes a module, *inversion*, which standardizes a common design element, and *porting*, which transports a module for use in another system.

Realizing the need to provide a way (not necessarily perfect) to quantitatively evaluate modular designs, Baldwin and Clark [5] set out a theory to define a model for reasoning about the value added to a base system by modularity. They formalized the options value of each modular operator: the value of being able to substitute, augment modules, etc. A model was derived to calculate the NOV (Net Option Values), which is the expected payoff of exercising a search and substitute option optimally, accounting for both benefits and exercise costs. Sullivan et al. [48] explored the feasibility of applying this concept to software despite the difficulty justifying precise estimates for real options in software design.

DSM significance for software design was first explored in depth by Sullivan et al. [48][49] and subsequently by Lopes et al. [30], MacCormack [31], Sangal et al. [43], and Cai [6]. In [48], Sullivan et al. applied DSM modeling to Parnas's canonical example, showing that Baldwin and Clark's approach could be used to visualize and formalize Parnas' theory. They also used options analysis to show that the information- hiding design of KWIC generates a higher total value of the system. Lopes et al. [30] later employed similar methods to compare aspect-oriented design vs. object-oriented design. In [49], Sullivan et al. [19] developed a concept of XPI, a special form of design rules that decouple aspect code with the base code based on DSM modeling and design rule theory. Sangal et al. [43] used a commercial static analysis tool to recover dependency models from source code for the purpose of discovering and communicating software architecture. MacCormack et al. [31] used the DSM modeling techniques to compare two complex software systems, the Linux kernel and the Mozilla web browser. Cai's [6] work focuses on modeling design decisions and dependencies that span the software lifecycle using augmented constraint networks (ACN) and automatically generating DSMs from logic models.

2.3 Traceability

Requirements traceability is a fundamental part of software and information systems development [38] supporting various activities during the life cycle of a software system. Dahlstedt and Persson [7] observe that most requirements cannot be treated independently, since they are related to and affect each other in complex manners. Actions performed based on one requirement may affect other requirements in ways not intended or not even anticipated. Dependencies between requirements may also affect various decisions and activities during development, e.g. requirements change management, release planning, requirements management, requirements reuse, and requirements implementation. This implies that there is a need to take interdependencies into consideration in order to make sound decisions during the development process. The topic of traceability has been the focus of intense research; Ramesh and Jarke [40] present an extended overview of the current state of research within the area.

Pohl [4] in his early work developed a traceability framework, which included a dependency model defining 18 different types of possible dependency links. Pohl’s model describes dependency types that can exist between any type of trace object used in the requirements engineering process. Pohl mentions that knowledge about how the requirements have evolved, and hence relate to each other, is considered to be important when dealing with changes and change integration. Pohl subsequently identifies requirements interdependencies as an enabler of identifying reusable software components. Similarly, Kotonya and Sommerville [24] state that the notion of requirements interdependency is one of the most important aspects of traceability from a change management perspective.

Significant effort has been devoted in the literature to identify the different kinds of traceability and the activities required to identify them. Gotel and Filkenstein [18] state that traceability can be divided into two main types: *Pre-RS traceability* and *Post-RS traceability*. *Pre-RS traceability* refers to those aspects of a requirement’s life before it is included in the requirements specification and is focused on enabling a better understanding of the requirement. *Post-RS traceability* refers to those aspects of a requirement’s life from the point in time when it has been included in the requirements specification and is focused on enabling a better understanding and acceptance of the current system/software. Requirements Pre-RS traceability is hence concerned with requirements production and focuses on the domain with which we interact when the requirements are developed and in which the systems is to be installed. Requirements Post-RS traceability is concerned with requirements deployment and is focused on the software that is developed based on the requirements. Gotel and Filkenstein report how the majority of the problems attributed to poor requirements traceability are due to inadequate pre-RS traceability.

In a different line of research, Dick [11] proposes improving the system engineering process by applying of a deeper semantics in the traceability relationship. These richer traceability relationships are established by making use of textual rationale and propositional logic in the construction of traceability arguments. Dick identifies three key points required to achieve meaningful traceability relationships: textual rationale, grouping of links, and typing of link groups. Dick argues for the use of structures that use textually supported propositional combinators to document the exact nature of the relationship between a set of requirements and the response to them.

Other research efforts defined traceability using various types of structures (patterns [22], scenarios [14], Petri nets [19], and rule-based frameworks [45]) and most has focused on generating traces in existing designs [32] rather than optimizing traceability and independence (and thus changeability) in the design process itself.

In the following sections, we show how an organization of information, called an intent specification, along with a concept we call semantic coupling, can assist in creating system designs that reduce the effects of requirements changes. Just as module coupling has been used to guide the structural decomposition of modules, semantic coupling can be used to guide the process of functional decomposition of requirements and high-level system design. This design process is based on the use of traceability matrices (instead of DSM); in our work, these traceability matrices adopt a role parallel –albeit in a fashion directly applicable to complex safety-critical systems- to the DSM one in the work performed by the scholars previously introduced.

3. INTENT SPECIFICATIONS

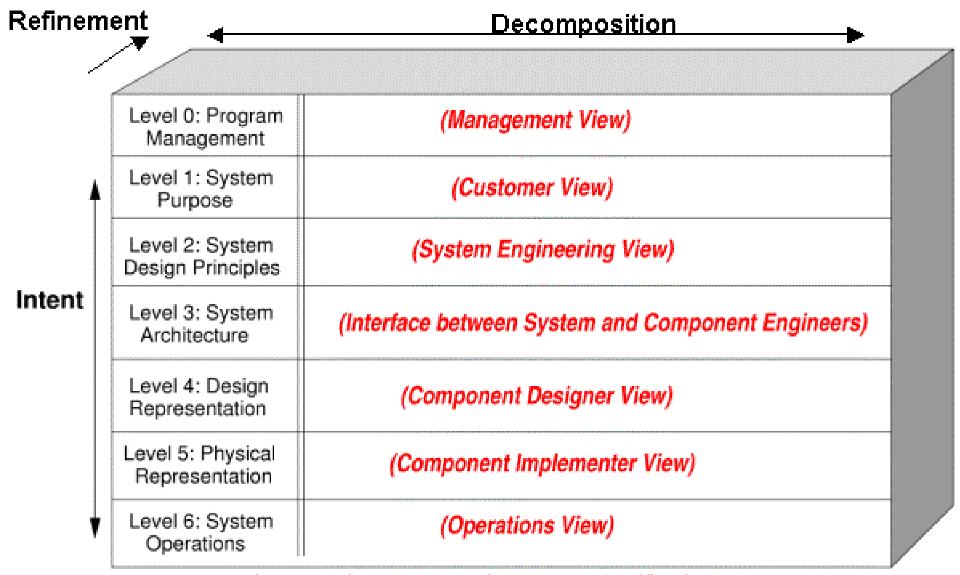
Intent specifications are a framework to support the entire life cycle (from initial definition and development to operational deployment and use) of complex software-intensive safety-critical systems. Special attention is given to the requirements definition process; experience in a multitude of such projects shows that errors made during the early requirements definition activities are one of the biggest problems facing their designers. As Knight [23] states, this is true today even more than in the past: advances in software technology have alleviated the difficulty in implementing the technical solutions but we still do not have a solution to the requirements problem.

3.1 Intent Specification Design

The design of intent specifications applies research in systems theory, cognitive psychology, and human-machine interaction to specification design in order to support the tasks humans perform during system and software development [26]. The structure, content, and notation of intent specifications are designed to assist software engineering in developing appropriate mental models about the system while supporting them in using a wide array of problem-solving strategies. This methodology has been successfully used in a wide array of aeronautical and space safety-critical projects and is the basis for a commercial product.

Intent specifications employ a type of hierarchical abstraction based on intent or purpose. In computer science, hierarchical abstraction has been extensively used to enhance both top-down and bottom-up reasoning about complex systems. Two types of abstraction have been used: (1) part-whole abstractions where each level of a hierarchy represents an aggregation of the components at a lower level and (2) information hiding (refinement) abstractions where higher levels contain the same conceptual information but hide some details about the concepts, that is, each level is a refinement of the information at a higher level. Using these hierarchical abstractions, each level of the usual software specifications can be thought of as providing "what" information while the next lower level describes "how." Such hierarchies, however, do not provide information about "why." Information about purpose or intent (i.e., design rationale) cannot be inferred from what we normally include in such specifications. For this reason, intent specifications add the third abstraction as shown in Figure 1. This figure provides the overall view of an intent specification; note how the refinement and decomposition abstractions evolve along the intent axis as the language and models used change.

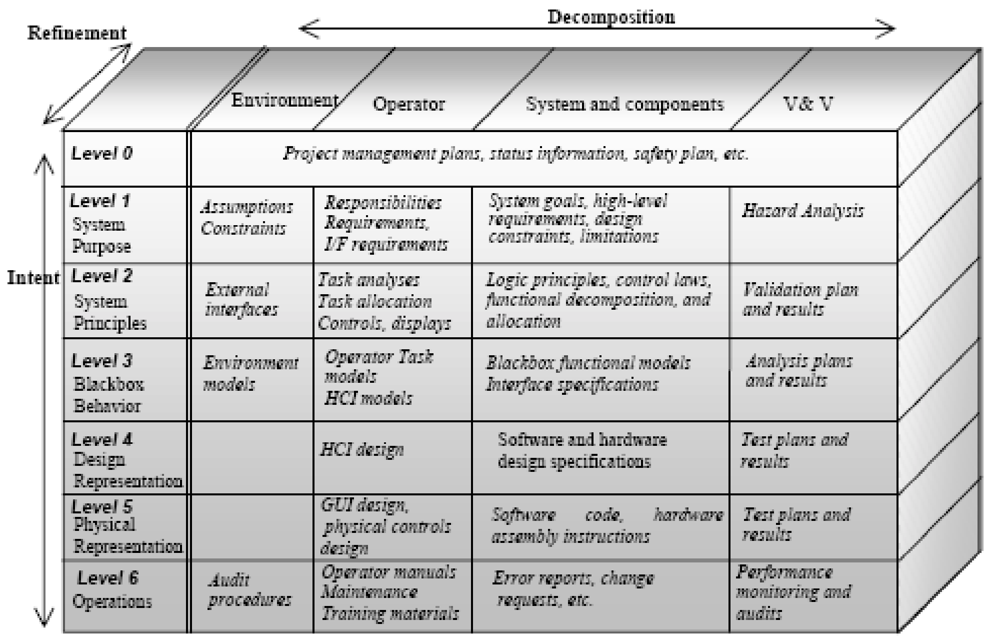
Each level of the intent abstraction contains information about goals or purpose for the level below, described using a different set of attributes or language. High-level goals are not constructed by merely integrating information from lower levels; instead each level provides different, emergent information with respect to the lower levels. A change of level represents both a shift in concepts and structure for the representation (and not just a refinement of them) as well as a change in the type of information used to characterize the state of the system at that level. Thus each level is a different view or model of the entire system and may use a different language.


Refinement
Decomposition
Intent
Level 0: Program Management
(Management View)
Level 1: System Purpose
(Customer View)
Level 2: System Design Principles
(System Engineering View)
Level 3: System Architecture
(Interface between System and Component Engineers)
Level 4: Design Representation
(Component Designer View)
Level 5: Physical Representation
(Component Implementer View)
Level 6: System Operations
(Operations View)
Figure 1: The Structure of an Intent Specification

The design of intent specifications was selected based on the literature analyzing the strategies followed by experts in complex system design [25]. Rasmussen [41] explains how the consideration of purpose or reason has been shown to play a major role in understanding the operation of complex systems. Experts and successful problem solvers tend to focus first on analyzing the functional structure of the problem at a high level of abstraction and then narrow their search for a solution by focusing on more concrete details [17]. Representations that constrain search in a way that is explicitly related to the purpose or intent for which the system is designed have been shown to be more effective than those that do not because they facilitate the type of goal-directed behavior exhibited by experts [51].

3.2 Intent Specification Content

Each of the seven intent specification levels supports a different type of reasoning about the system. Figure 2 shows an example of the content that might appear in an intent specification, although the format and specific content of the levels is up to the user and the application involved.



	Environment	Operator	System and components	V& V
Level 0	*Project management plans, status information, safety plan, etc.*			
Level 1 System Purpose	*Assumptions Constraints*	*Responsibilities Requirements, I/F requirements*	*System goals, high-level requirements, design constraints, limitations*	*Hazard Analysis*
Level 2 System Principles	*External interfaces*	*Task analyses Task allocation Controls, displays*	*Logic principles, control laws, functional decomposition, and allocation*	*Validation plan and results*
Level 3 Blackbox Behavior	*Environment models*	*Operator Task models HCI models*	*Blackbox functional models Interface specifications*	*Analysis plans and results*
Level 4 Design Representation		*HCI design*	Software and hardware design specifications	*Test plans and results*
Level 5 Physical Representation		*GUI design, physical controls design*	*Software code, hardware assembly instructions*	*Test plans and results*
Level 6 Operations	*Audit procedures*	*Operator manuals Maintenance Training materials*	*Error reports, change requests, etc.*	*Performance monitoring and audits*

Figure 2: An Example Contents of an Intent Specification

The Management view (Level 0) provides a bridge from the contractual obligations and the management planning needs to the high-level engineering design documentation.

The System Purpose (Level 1) assists system engineers in their reasoning about system-level goals, constraints, and limitations. It also contains the preliminary hazard analysis and the hazard log used to document and track safety engineering activities [29]. System safety analysis is used to establish safety-related system and software requirements and design constraints [13][36]. Level 1 contains such information as environmental assumptions, requirements, and constraints, high-level system requirements and design constraints, hazard analyses, interface requirements, system limitations, etc.

The System Design Principles level (Level 2) documents the system in terms of the physical principles and laws upon which the design is based. Complete functional structures and interfaces between components are defined at this point. This level also documents task models and other results of human factors analyses for systems containing human operators. The principles are linked to the related higher-level requirements, constraints, assumptions, limitations, and hazard analysis, as well as linked to lower-level system design and documentation. Assumptions used in the formulation of the design principles may also be specified at this level. For the purpose of this paper, we will review in detail the way in which engineers define "how" and "why" the level 2 preliminary design meets the high-level requirements while taking into account all the applicable constraints, assumptions and limitations.

The Blackbox level (Level 3) allows the designers to study the logical design of the system using a formal modeling language called SpecTRM-RL [27]. The tools associated with this language produce executable and analyzable models that engineers can use to study the complex interactions between the different components of the system. SpecTRM-RL models can be built for the non-software components also, or they can be executed together with simulations and prototypes of hardware components.

The Design Representation level (Level 4) presents a detailed implementation-dependent design. Its content will depend on whether the particular function is being implemented using analog or digital devices or both. For functions implemented on digital computers, the fourth level might contain the usual software design documents or it might contain information different from that normally specified. Other information at this level might include hardware design descriptions, the human-computer interface design specification, and verification requirements for the requirements and design specified on this level.

The Physical Representation level (Level 5) a description of the physical implementation of the levels above. It might include the software itself, hardware assembly instructions, training requirements (plan), etc.

Finally, the System Operations level (level 6) includes information produced during the actual operation of the system, which can be used in operational audits, performance analyses, operational safety analyses, and change analysis.

Each level also contains documentation of the plans for and results of the verification and validation process for the design decisions contained in each level.

3.3 Intent Specification Mappings

Mappings between intent specification levels are potentially many-to-many: Components of the lower levels can serve several purposes while purposes at a higher level may be realized using several components of the lower-level model. These goal-oriented links between levels can be followed in either direction. Changes at higher levels will propagate downward, i.e., require changes in lower levels, while design errors at lower levels can only be explained through upward mappings (that is, in terms of the goals the design is expected to achieve). In this paper, we show how the structure of these mappings from one intent level to another define the semantic coupling between the components at the higher level and can be designed to reduce this coupling.

Intent specifications assist in identifying intent-related dependencies. Because each level is mapped to the appropriate parts of the intent levels above and below it, traceability of not only requirements but also design rationale and design decisions is provided from high-level system goals and constraints down to code (or physical form if the function is implemented in hardware) and vice versa. This mapping is not a mathematical relationship, but a mapping based on design rationale and intent. For example, while one level might describe the logic of TCAS II in terms of whether an intruder aircraft is on the ground or not, the next higher level might define “on the ground” (which is surprisingly complex) and why the engineers chose that particular definition [28].

The intent specification approach to specify mappings is supported by research in traceability. Consider the concept of Pre-RS traceability and Post-RS traceability as defined by Gotel and Filkenstein [18]. The recording of intent information in the intent specification as the system design is specified in terms of level 0 (contractual obligations, etc), level 1 (System Purpose), then level 2 (Design Principles) provide the data identified by Gotel and Filkenstein as Pre-RS traceability, which, as they state, is in the origin of most problems attributed to poor requirements traceability. Nevertheless, the more traditional Post-RS traceability is also available in the mappings performed below intent specification level 2 (Design Principles). Thus, intent specifications are able to manage traceability throughout the whole life cycle.

Pohl [4] and other researchers also focused on studying the notion of requirements interdependency and classifying their different kinds. Dahlstedt and Persson [7] identify the following types of structural interdependencies in terms of keywords: “requires”, “explains”, “similar_to”, “conflicts_with”, and “influences”. Two cost/value interdependencies are also relevant:

“increases/decreases cost of” and “increases/decreases value of”. No task associated with intent specifications actively seeks to classify the existing links between levels in such terms. However, the usage of traces from the level 1 requirements, constraints, assumptions, limitations to the level 2 design principles creates the semantic equivalent to the previously identified structural interdependencies. The correspondence is not exact, however, since Dahlstedt and Persson definition concerns itself with a generic design process while intent specifications are focused on a specific framework designed for the development of safety-critical software-intensive systems.

Intent specifications also support Dick’s [11] notions of rich traceability by providing means to record rationale information and thus capture the specialists domain knowledge. While there is no explicit typing of links groups, the implicit dependencies between the elements in an intent specification again provide an equivalence relation. A main difference with Dick’s approach is that intent specifications do not provide propositional combinators to express properties such as design choices (design alternatives). The intent specification approach is focused on designing a flexible methodology that support continuous design iterations rather than managing the burden of complex logical propositions that would be difficult to maintain in a large project.

Finally, we must emphasize the criticality of verification activities in the design of modern safety-critical software systems. Traditional data for the development of software in civil aviation indicates that verification activities can represent up to the 40% of the total cost of a project (with requirement definition and correction related activities taking another 30 %). It is therefore imperative to improve all requirement traceability techniques. Intent specifications provide an adaptable framework to support robust traceability techniques during the whole system life-cycle.

3.4 Research Related to Intent Specifications

This concept of compiling rationale data is also used in other methodologies, such as in goals-oriented requirements engineering. Van Lamsweerde [50] describes why goals are important in the requirements engineering process: goals can provide a precise criterion for sufficient completeness of a requirements specification and be used to eliminate irrelevant requirements [54], improve requirements readability, etc. [8][53]. Goals are defined by their type, attribute, and links. Van Lamsweerde also presents a goal taxonomy: goals can be associated with certain functional services that need to be provided or they can be associated with quality of service objectives. However, goals are only one part of the rationale behind design decisions and do not provide the complete rationale argument.

Intent specifications document the goal (requirements) hierarchy. While goal classification is not performed explicitly, the derivation of high-level functional requirements, limitations, design constraint, safety constraints, etc is performed as a natural part of the design process in a way in which goals are effectively refined into functional and non-functional requirements as defined in the goals-oriented requirements approach.

Beyond that, in goals-oriented requirements engineering, as reported by Van Lamsweerde [50], significant effort has been directed at defining goal structures, frequently with AND/OR graphs, to capture goal refinement links [8]. Furthermore, a series of semi-formal and formal approaches using typically verbal keywords and a textual or graphical syntax [9] have been defined in order to provide a means to support goal verification and validation, requirements elaboration, conflict management, etc. Goal/requirement elicitation is achieved either by refinement (asking “HOW” questions about the goals already identified) or by abstraction (asking “WHY” questions about operational descriptions already available). Some work [8] is also available to support the process of deriving pre-, post-, and trigger conditions on software operations so as to ensure the terminal goals in the refinement process.

In contrast, intent specifications provides support for a hierarchical goal/requirement structures but no provision is made for goal textual or graph syntaxes, the use of the “OR” attribute (design alternatives), or keyword-based formal approaches. Intent specifications are specifically designed to adapt themselves to the users in the way they think. There is research to show that forcing users to use one representation of a problem inhibits problem-solving success [24, Dulac 2002] and the adoption of tools by sophisticated users. Formalization efforts in intent specifications are limited to level 3 (Blackbox Behavior) using executable state-machines. Even here, different modeling languages or visualizations might be used to provide flexibility to users [Dulac 2002]. Intent specifications also focus on recording the rationale behind every design decision; the "WHY" question is not used as a way to reverse engineer a design. The latter approach is unlikely to be feasible in any large project with any significant workforce turnover. To the contrary, intent specifications recording of rationale data is integrated into the design process and documentation to ensure this critical information is never lost.

4. SEMANTIC COUPLING

4.1 Semantic Coupling Fundamentals

Intent specifications allow isolating the assumptions that need to be considered or reconsidered to make a particular change and provide traceability between requirements and design at each level of the intent hierarchy. Therefore, they will not only help in understanding the system and the relationships between requirements and design but they will also assist in validating any changes to make sure no errors are introduced that may compromise compliance with the system high level requirements or constraints,

While recording design rationale in a usable and traceable way should reduce the effort involved in responding to changed requirements, it is still possible to design intent specifications for which changes in requirements will cause extensive changes (rippling effects) at each level of the specification. To assist with this problem, we introduce the concept of semantic coupling.

Semantic coupling or independence is defined in terms of the mappings between the levels of the intent specification, that is, mappings between the goals and the means for achieving the goals. The emphasis in this paper will be on the first two levels, but the same definition applies at each level. At the lower levels of an intent specification, where the logic is mapped to software structures and modules, the concept will be parallel to that of module cohesion and coupling.

Using these mappings, three types of coupling or independence can be identified[1]:

1. Uncoupled: the mappings or functions from requirements to design principles are all one-to-one.
~~1.~~2. Loosely coupled: the mappings are one-to-many.
~~1.~~3. Tightly coupled: the mappings are many-to-many.

Formatted: Bullets and Numbering

For any complex control system, a completely uncoupled design, while allowing changes to requirements with minimal impact, is usually not practical. A more realistic goal is to design to reduce the impact of requirements changes, i.e., to eliminate the rippling effects to other requirements and therefore through the system design as a whole by designing the mappings to be one-to-many. Our design goal then becomes: eliminate (or if not possible, reduce) many-to-many mappings and create a design such that the one-to-many mappings are reduced. The latter can be accomplished by a careful ordering of design decisions and hierarchical ordering of requirements, as illustrated below. For ease of manipulation and visualization, we describe the functions using Matrix Algebra constructs. Thus, the relationship between the requirements (R) and the design principles (DPs) in the lower level can be characterized mathematically by expressing them in terms of vectors:

[1] A possible fourth type "mappings from many to one" is not considered realistic. In intent specifications, the nature of the languages used along the intent hierarchical abstraction leads to the multiplication of the associated structures as the system is defined at the lower levels of an intent specification.

(1) {R} = [A] {DP}

where {R} is a vector of m requirements, {DP} is a vector of n design principles, and [A] is the traceability matrix. The values of the traceability matrix will be either * or 0, where * indicates a mapping exists between the corresponding vector components while 0 signifies no mapping.

For intent specification mappings between level 1 and level 2, the vector {R} can itself be divided into a vector of high-level functional requirements {FR} and a vector of non-functional requirements {NFR} (constraints, limitations, environmental assumptions). The {DP} vector itself becomes {SDP}, the minimum set of implementation-independent system design principles required to specify the design.

For a level 1 to level 2 mapping, consider the case of a square traceability matrix where m=n=3. The simplest and most desirable design, an uncoupled design, is one in which all the non-diagonal elements of the traceability matrix are zero. That is:

$$(2)\ [A] = \begin{bmatrix} A_{11} & A_{21} & A_{22} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} * & 0 & 0 \\ 0 & * & 0 \\ 0 & 0 & * \end{bmatrix}$$

where the {R} has simplified to be the set of functional high-level requirements {FR}

(3) {FR} = [A] {SDP}

In the uncoupled design, each of the FRs can be satisfied independently from the others. If an element FR_i of the {FR} vector changes, all one needs to do is to adjust the corresponding system design principle SDPi. Note that in software, the traceability matrix will almost always not be square, that is, there will be multiple SDPs for each requirement.

A coupled design is one in which elements on both sides of the diagonal are non-zero, as in:

$$(4)\ \begin{pmatrix} FR_1 \\ FR_2 \\ FR_3 \end{pmatrix} = \begin{bmatrix} * & * & 0 \\ 0 & * & * \\ * & * & * \end{bmatrix} \begin{pmatrix} SDP_1 \\ SDP_2 \\ SDP_3 \end{pmatrix}$$

In this case, a change in any of the requirements is likely to produce a cascading effect, forcing all three SDPs to be adjusted. In general, any requirements changes in a coupled design can require a highly complex, iterative redesign of the system. This process can be extremely expensive, particularly if the changes take place far into the detailed design or implementation phase.

The third case is the loosely coupled design, where the traceability matrix is lower or upper triangular:

$$(5)\ \begin{pmatrix} FR_1 \\ FR_2 \\ FR_3 \end{pmatrix} = \begin{bmatrix} * & 0 & 0 \\ * & * & 0 \\ * & * & * \end{bmatrix} \begin{pmatrix} SDP_1 \\ SDP_2 \\ SDP_3 \end{pmatrix}$$

In a loosely coupled design, even though the requirements are not strictly independent, the impact of potential changes can be reduced by adjusting the SDPs in the proper order. Thus, as stated above, while total independence or uncoupling is almost always not achievable for real systems, the effects of changes in a loosely coupled design can be reduced by a careful ordering of design decisions and hierarchical ordering of requirements. Our goal then, is to create designs that are loosely coupled and to order the design decisions such that changes will have minimal impact. This process above is oversimplified because different types of requirements will affect the design in different ways. Recall that, in an intent specification, Level 1 contains three types of “requirements”: functional requirements, environmental assumptions, and design constraints[2].

4.2 Semantic Coupling in Level 1 to Level 2 Mappings

Figure 3 shows an example of a traceability matrix illustrating how the general concept of semantic coupling is applied to a specific mapping (Level 1 to Level 2). Level 1 includes three kinds of requirements: high-level functional requirements (FR), environmental assumptions (EA), and design/safety constraints (DC or SC). Level 2 includes the system design principles (SDP) used to implement the requirements in Level 1. Note that all these elements are to be developed in a hierarchical structure. In this way, a refinement of the FRs, SDPs, etc yield tree-like constructs that are denoted in the following fashion: FR1.x.x, SDP1.x.x, etc. The dependencies or links between the elements of the two levels are marked with a cross “X” in the traceability matrix.

		System Design principles		
		SDP 1	SDP 2	SDP 3
High-level FRs	FR 1	X		
	FR 2		X	
Environmental Assumptions	EA 1	X	X	
	EA 2			X
Design Constraints	DC1		X	
	SC1	X		

Figure 3: Example Traceability Matrix

As expected, the traceability matrix is not square. Nevertheless, this is not an obstacle to analyzing the coupling in the design. Before going any further, it is important to understand the nature of information provided by each of the sub-matrices in which the overall traceability matrix can be divided. Each sub-matrix provides a representation of the dependency that exists between the design parameters and the corresponding functional or non-functional requirements. Each mapping will not have the same semantic content:

- *FR-to-SDP submatrix*. The links in this case indicate which system design parameters (SDPs) satisfy / implement the high-level functional requirements.
- *EA-to-SDP submatrix*. These links indicate those SDPs that depend on assumptions about the behavior of components in the system environment or physical interface. This dependency can be a simple data dependency (format, range, or time properties) but can also include behavioral information (the external system is expected to accomplish certain tasks, etc.).
- *DC-to-SDP submatrix*. These links identify those SDPs that can violate a constraint if not properly defined. That is, it specifies which constraints are applicable to each SDP. The mappings from design constraints (DCs) to SDPs have a different meaning than those linking FRs and SDPs. The SDPs must not violate the DCs, but they do not necessarily

[2] We do not separate performance requirements from functional requirements. In a real-time system, this separation is not possible and when attempted, as often occurs, leads to many of the problems that arise in building such systems. The reason behind this claim is beyond the scope of this paper. See Jaffe [7] for an explanation.

implement a safety function nor does the function necessarily need to be modified to accommodate a change in the constraint.

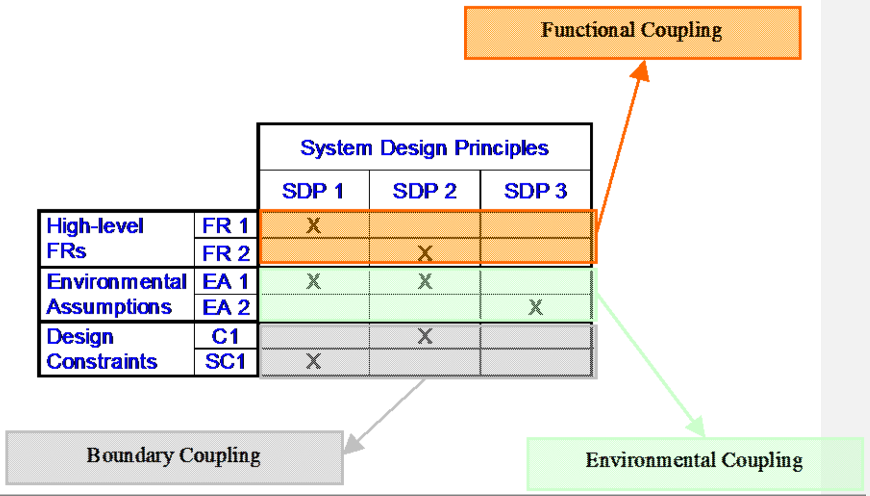


Figure 4: Coupling Types in a Level 1 to Level 2 Mapping

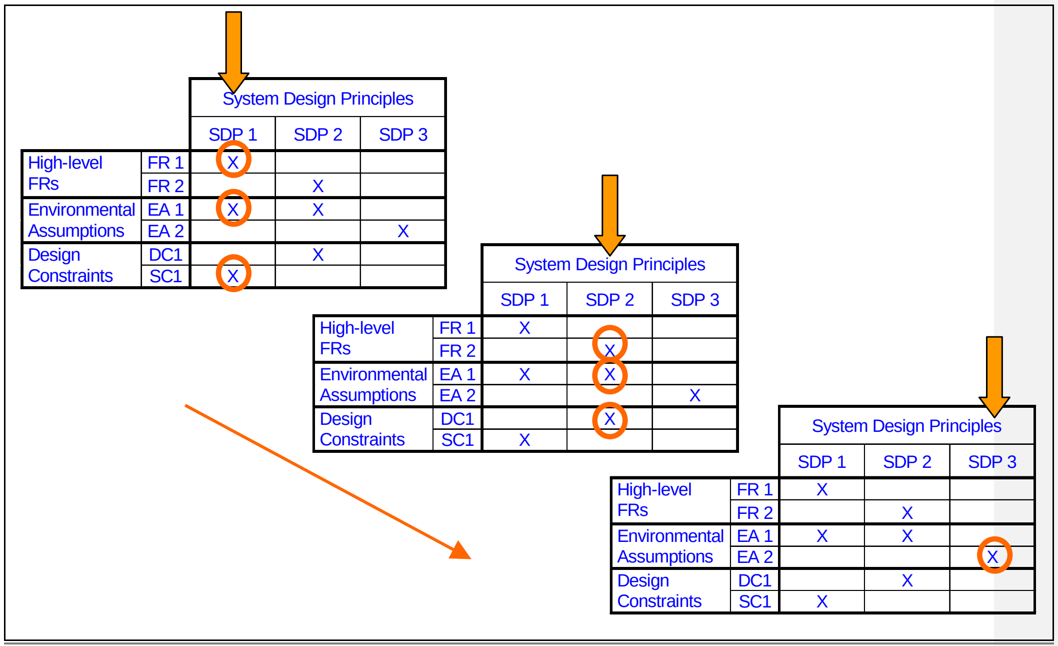
Following from the previous discussion, three types of couplings can be identified as shown in Figure 4:

- *Functional Coupling* represents the core of the interdependency problem and is the subject of the general semantic coupling discussion in section 4.1. For each SDP, an attempt is made to minimize the dependencies outside its hierarchical structure. In the ideal case, an *uncoupled matrix* (a diagonal matrix that indicates requirements can be independently satisfied from each other) is obtained

- *Environmental Coupling* represents the dependencies between the system design and the assumptions made about its physical environment or interfacing systems. This coupling will not be nearly as strong as the previous one because, in this case, the SDPs do not have a relation with the implemention of a goal/requirement but with the use of external information. In the ideal case, a *null matrix* (denoting no dependency on external information) is obtained.

- *Boundary Coupling*. Design and safety constraints will typically define a boundary of acceptable system behaviors. As long as these constraints only indicate *what* hazards or limitations the system must respect, no functional coupling results, thus minimizing the sensitivity of the design with respect to changes in these constraints. Therefore, it is imperative they be formulated in terms of “what” hazards/limitations are to be respected and not on “how” the system design must go about respecting those hazards/limitations. Nevertheless, any change to an SDP linked to a safety constraint must trigger an analysis to ensure safety has not been compromised. In the ideal case, a *null matrix* (impling that no possible behavior of the system can violate the prescribed constraints) results.

Whenever the environmental assumptions or the design constraints prescribe a behavior to be followed, they must be rewritten as high-level functional requirements or be treated as such within the semantic coupling assessment.



		System Design Principles		
		SDP 1	SDP 2	SDP 3
High-level FRs	FR 1	X		
	FR 2		X	
Environmental Assumptions	EA 1	X	X	
	EA 2			X
Design Constraints	DC1		X	
	SC1	X		

		System Design Principles		
		SDP 1	SDP 2	SDP 3
High-level FRs	FR 1	X		
	FR 2		X	
Environmental Assumptions	EA 1	X	X	
	EA 2			X
Design Constraints	DC1		X	
	SC1	X		

		System Design Principles		
		SDP 1	SDP 2	SDP 3
High-level FRs	FR 1	X		
	FR 2		X	
Environmental Assumptions	EA 1	X	X	
	EA 2			X
Design Constraints	DC1		X	
	SC1	X		

Figure 5: Semantic Coupling Assessment

The process to assess the degree of semantic coupling in the design entails a review of how each system design parameter depends on the high-level functional requirements defined in the intent level above (see Figure 5). This process is repeated along the refinement hierarchy to cover the entire set of system design parameters (Figure 6). In practice, this process will be iterative as design parameters and traceability links are continuously corrected and updated. The method is presented here for a sample abstract matrix; the application of this method to a real-world example is given in Section 5.

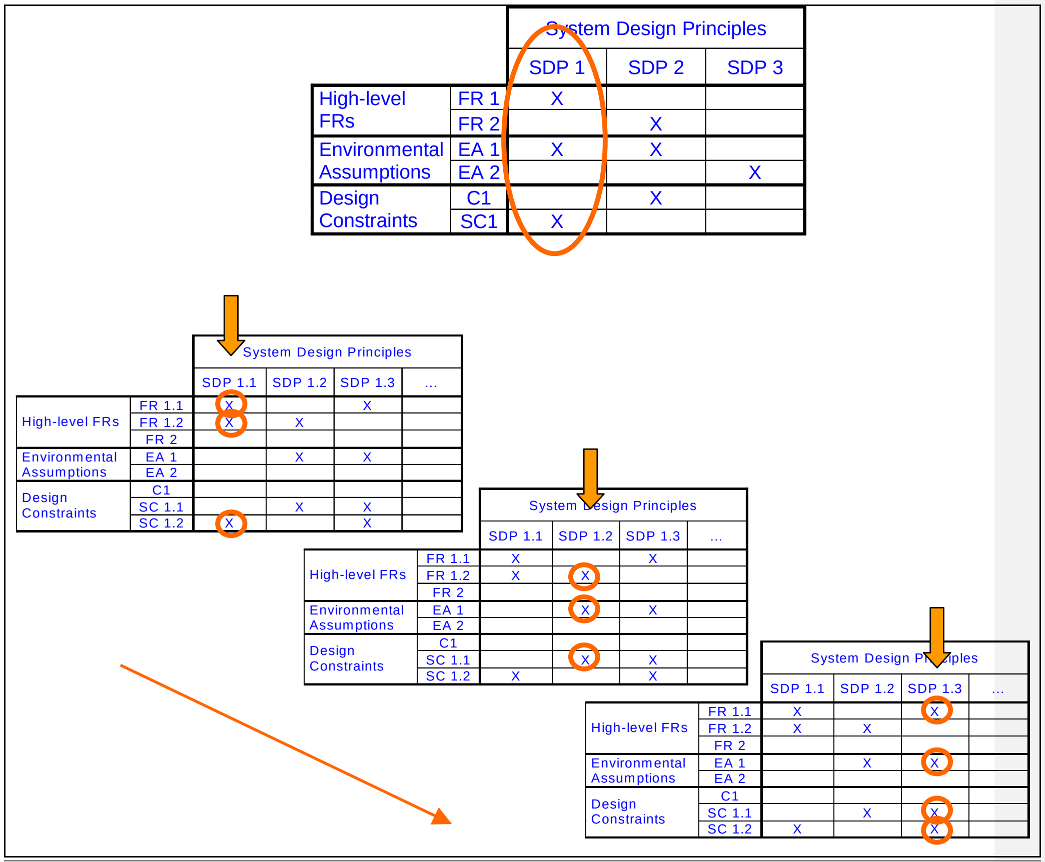
Using the sample matrix in Figure 3, we search through the columns of the traceability matrix as depicted in Figure 5. From this process, we determine that:

Column 1:	SDP1 implements FR1	SDP1 uses EA1	SDP1 is bound by SC1
Column 2:	SDP2 implements FR2	SDP2 uses EA1	SDP2 is bound by DC1
Column 3:		SDP3 uses EA3	

In this case, we have an uncoupled system since SDP1 and SDP2 satisfy independently FR1 and FR2, respectively. (Its traceability matrix is $\begin{bmatrix} * & 0 \\ 0 & * \end{bmatrix}$).

Also note how SDP1 and SDP2 are dependent on the environmental assumption EA1. Thus this system is coupled with respect to the environmental assumptions; changes in this assumption could have a large impact on the design. Appropriate design changes or tradeoff decisions should also take this coupling into account in order to reduce its effects.

A significant side advantage of the process described previously is that engineers are prompted to engage closely with the traceability matrices as part of the design process. Because traceability links are unlikely to be correct the first time around, engineers will engage in an iterative process to complete and correct the dependency links available. The use of a graphical representation (traceability matrix) supports the abstract reasoning required to perform this task.



		System Design Principles		
		SDP 1	SDP 2	SDP 3
High-level FRs	FR 1	X		
	FR 2		X	
Environmental Assumptions	EA 1	X	X	
	EA 2			X
Design Constraints	C1		X	
	SC1	X		

		System Design Principles			
		SDP 1.1	SDP 1.2	SDP 1.3	...
High-level FRs	FR 1.1	X		X	
	FR 1.2	X	X		
	FR 2				
Environmental Assumptions	EA 1		X	X	
	EA 2				
Design Constraints	C1				
	SC 1.1		X	X	
	SC 1.2	X		X	

		System Design Principles			
		SDP 1.1	SDP 1.2	SDP 1.3	...
High-level FRs	FR 1.1	X		X	
	FR 1.2	X	X		
	FR 2				
Environmental Assumptions	EA 1		X	X	
	EA 2				
Design Constraints	C1				
	SC 1.1		X	X	
	SC 1.2	X		X	

		System Design Principles			
		SDP 1.1	SDP 1.2	SDP 1.3	...
High-level FRs	FR 1.1	X		X	
	FR 1.2	X	X		
	FR 2				
Environmental Assumptions	EA 1		X	X	
	EA 2				
Design Constraints	C1				
	SC 1.1		X	X	
	SC 1.2	X		X	

Figure 6: Refinement in Semantic Coupling Assessment

The assessment is continued along the refinement axis. Implementing the refinement of SDP1 into SDP1.1, SDP1.2 and SDP1.3 (along with minor refinements in the requirements), we obtain (Figure 6):

Column 1: SDP1.1 implements FR1.1 and FR1.2
Column 2: SDP1.2 implements FR1.2
Column 3: SDP1.3 implements FR1.1

Its associated traceability matrix is $\begin{bmatrix} * & 0 & * \\ * & * & 0 \end{bmatrix}$, which indicates we have a loosely uncoupled design at this stage. When this is the case, the analysts need to identify the appropriate ordering of design decisions required to unlock the coupling. Those design parameters that involve a higher degree of coupling (in this case, SDP1.1 has to satisfy two requirements) must be managed first. In our example:

FR1.1 = SDP1.1 + SDP1.3
FR1.2 = SDP1.1 + SDP1.2

Once SDP1.1 has been set, SDP1.2 and SDP 1.3 can be adjusted independently to satisfy the FRs. Note that the actual ordering of the SDPs is irrelevant.

5. CASE STUDY: MAPS

The software chosen for the case study is the Mobility and Positioning Software (MAPS), part of the control software for a CMU/NASA robot [12]. The robot was designed to inspect and waterproof, between Space Shuttel flights, each of the 17,000 silica tiles that protect the Space Shuttle underside. Because there are so many tiles, the robot divides its work area into uniform workspaces inspecting tiles in each area with as little overlap between work spaces as possible.

Before each inspection shift, a supervisor enters instructions and information about shuttle position and inspection sequence via an off-board computer, the Workcell Controller. The Workcell Controller workstation is used to create jobs and update other NASA databases after the robot uploads data gathered during the course of the shift. This data includes tile images, records of tiles injected and inspected, and other pertinent job data. In addition, robot status data is used to monitor robot operation.

At the beginning of the shift, a job is downloaded to the robot. A job consists of a series of files describing the locations, sequences, target IDs, orbiter parking measurements, etc. The robot then uses a rotating laser to position itself under the shuttle, and the robot's camera locates the exact tile to be inspected. Because the shuttle's belly is not flat, the robot must customize its upward movement to each tile: Two vertical beams on either side of the robot raise the manipulator arm, which holds the injection tools and camera. A smaller lifting device raises the arm the rest of the way. By comparing the current state of each tile with the state of the tile at previous inspections, the robot characterizes anomalies in tiles as cracks, scratches, gouges, discoloring, or erosion. The robot also indicates when it is unsure what is wrong with a tile, so the supervisor can re-analyze the tile on the screen of the Workcell Controller. At the end of a shift, the robot's updated tile information is entered into existing NASA databases.

On board the robot, a computer controls the robot's high-level processing tasks while a low-level controller and amplifiers direct arm and wheel motions. Two more computers control the robot's vision and injection systems. If anything goes wrong, such as rising compartment temperatures or low battery level, safety circuits will shut down the robot.

MAPS is in charge of issuing low-level commands to the Motor Controller based on the inputs received either from the Planner (AI-based software) or a human operator. The Planner controls robot movement and positioning by providing MAPS with a specification of the destination and route. The operator controls robot movement and positioning using a hand-held joystick. The robot is unstable when the manipulator arm is extended, so stabilizer legs are used to provide stability. These legs must be retracted when the robot is in motion. MAPS is responsible for controlling the stabilizer legs. MAPS also monitors and controls several other robot subsystems and is responsible for most safety-related functions.

Level 1 of the MAPS intent specification contains the system goals, high-level functional requirements, limitations, constraints, and hazard analysis. It also includes the assumptions MAPS makes about the other robot subsystems and relevant external factors. Level 2 contains the MAPS design principles. We have completed the top three levels of the MAPS intent specification as well as the traceability matrices. However, they are too large to be included here. Instead, we describe in detail two parts of the specification to illustrate how the concept of semantic coupling can be applied to a real system. The information in the actual system requirements and design principles have been simplified in this paper for space and conciseness purposes. Downward links are also omitted as they are not relevant for the examples.

5.1 MAPS High-level Functionality

The first example focuses on the definition of the different modes under which MAPS can function. Following an initial assessment of the system goals and safety concerns, the designers identified the need to make the robot controllable both manually and automatically. These requirements originate in the need to provide a balance between the performance goals and the safety constraints. The FR subset corresponding to these requirements is:

MAPS Level 1 High-level Functional Requirements

MAPS-1.1.1: Computer-Controlled Operations

MAPS shall be able to operate automatically through control of the Planner alone.

Rationale: Human control of MAPS throughout the long and tedious tile servicing process (which takes several weeks) is impractical.

MAPS-1.1.2: Operator-Controlled Operations

MAPS shall be able to be operated under direct manual control.

Rationale: Sufficient confidence cannot be obtained in the automated implementation of some safety-related robot operations (like detecting a passerby or an unexpected obstacle in the path of the robot), and therefore human movement control will be used during some limited but particularly hazardous operations.

These FRs result in the definition of a series of modes and associated transitions in level 2 of the intent specification. However, these FRs are not adequate to produce a complete design. Assumptions about the robot's other components and about the way in which MAPS is supposed to interact with them must be known. For this example, the engineers need some basic assumptions about the Planner (PL), the Motor Controller (MC), and the operator joystick interface (JOY):

MAPS Level 1 Environmental Assumptions

PL1.2: The Planner will provide both a route of travel and a destination to MAPS in world coordinates.
MC3: The Motor Controller can be operated in position (relative displacement) mode.
MC4: The Motor Controller can be operated in velocity mode.
MC6: The Motor Controller is able to stop the motion of the robot within 0.2 seconds of receiving a 'stop' command.
JOY1: The operator will be able to drive the robot by deflecting a joystick in the direction the operator would like the robot to travel

Level 1 of the intent specification also includes a preliminary hazard analysis from which a set of safety constraints (SC) is derived. The SCs limit the way in which the functionality required by the FRs can be implemented. Two SCs related to the example are:

MAPS Level 1 Safety Constraints

SC1: The robot mobile base shall move only when commanded.
SC2: The robot mobile base shall stop when commanded.

During the design process, system engineers must define a functional structure out of all the Level 1 information. This synthesis process is quite challenging; it requires designers to take on a multi-disciplinary approach where the optimization of functional requirements may conflict with the safety constraints, limitations, environmental assumptions, etc. It is an iterative process by its very nature.

The level 2 *Design Principles* subset corresponding to the current example is:

MAPS Level 2 Design Principles

MAPS-2.2.1: Initialization Precondition

MAPS does not accept any motion commands until system initialization mode is complete [↑SC1].

<u>Rationale</u>: The normal operation of the system relies on the correct performance of the subsystems initialized during startup.

MAPS-2.2.2: Control Mode Selection Principles
At any time, MAPS is in one and only one of the following three modes: safety mode, computer mode, or joystick mode. Mode selection is based on the following principles: […]. [↑MAPS 1.1, ↑MAPS 1.1.1, ↑MAPS 1.1.2,↑SC2]

MAPS-2.2.3: Safety Mode
In Safety Mode, MAPS stops the robot, prevents further motions, notifies the operator of the problem that has occurred, and executes a set of recovery procedures. [↑SC2, ↑MC6]
<u>Rationale</u>: This mode provides MAPS with the ability to handle unsafe conditions and to return the robot to a safe state if possible.

MAPS-2.2.4: Computer-Controlled Operations
When operating in Computer mode, MAPS accepts routes from the Planner and generates all the necessary movement commands for the motor controller to move the robot along that route. Relative displacement mode is used. Motor controller commands are generated by calculating […] [↑MAPS-1.1.1, ↑MAPS-1.1.1.1, ↑MC3, ↑PL1.2, ↑SC1]
<u>Rationale</u>: This mode permits MAPS to position the robot very precisely and allows the optimization of the waterproofing operations.

MAPS-2.2.5: Operator-Controlled Operations
When operating in joystick mode, MAPS accepts movement commands from the operator via the joystick and issues all necessary motor controller commands to move the robot as commanded by the operator. Velocity mode is used when operating in joystick mode. In velocity mode, the kinematics on the body relative velocity are computed, and the appropriate wheel velocities are set. The computations used for the kinematics are […] [↑MAPS-1.1.2, ↑MC4, ↑Joystick1, ↑SC1]
<u>Rationale</u>: Velocity commands best match an operator's mental model of how robot base motions are controlled, i.e., they are the easiest for humans to understand and therefore monitor.

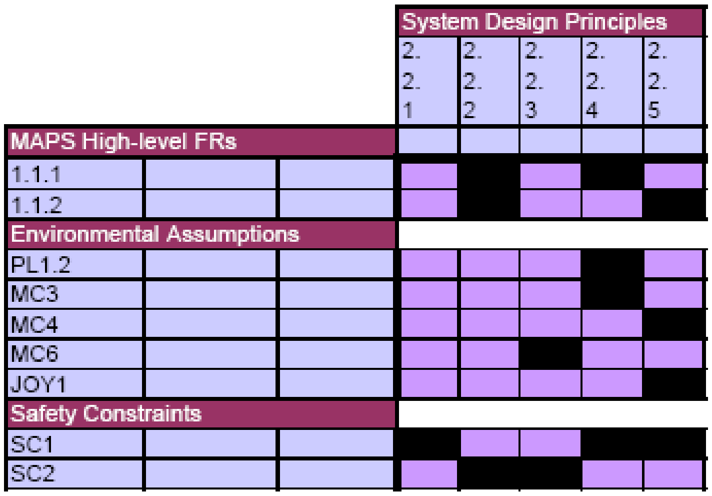
The corresponding overall traceability matrix is given in Figure 7.

Note how most of the level 2 design principles include the rationale behind design decisions as well as traceability information. This information can prove extremely useful in system redesigns and during maintenance by providing insight into the reasoning and assumptions of the original designers. The traceability links are denoted by up-arrows (↑), providing an explicit way to indicate a dependency or satisfying relationship between different items of an intent specification. They also provide rationale through structural means rather than requiring extra explanation. Our automated tools to support intent specifications use hyperlinks to implement traceability links.

Creation of the FR to SDP traceability submatrix for this example is immediate from the links:

(6)

$$\begin{pmatrix} FR_{1.1.1} \\ FR_{1.1.2} \end{pmatrix} = \begin{bmatrix} * & * & 0 \\ * & 0 & * \end{bmatrix} \begin{pmatrix} SDP_{2.2.2} \\ SDP_{2.2.4} \\ SDP_{2.2.5} \end{pmatrix}$$



			System Design Principles				
			2.2.1	2.2.2	2.2.3	2.2.4	2.2.5
MAPS High-level FRs							
1.1.1			🟪	⬛	🟪	⬛	🟪
1.1.2			🟪	⬛	🟪	🟪	⬛
Environmental Assumptions							
PL1.2			🟪	🟪	🟪	⬛	🟪
MC3			🟪	🟪	🟪	⬛	🟪
MC4			🟪	🟪	🟪	🟪	⬛
MC6			🟪	🟪	⬛	🟪	🟪
JOY1			🟪	🟪	🟪	🟪	⬛
Safety Constraints							
SC1			⬛	🟪	🟪	⬛	⬛
SC2			🟪	⬛	⬛	🟪	🟪

Figure 7: MAPS High-Level Functionality Traceability Matrix

As expected, the traceability matrix is not square. Equation (6) provides important information regarding the coupling in the system. $FR_{1.1.1}$ is only satisfied by $SDP_{2.2.2}$ and $SDP_{2.2.4}$. Similarly, $FR_{1.1.2}$ is satisfied by $SDP_{2.2.2}$ and $SDP_{2.2.5}$. This means that the implementation of each type of operation is accomplished by separate control modes, each independent of each other. The only SDP that is common to both functional requirements is the one defining the mode selection logic. This is clearly a case of a loosely coupled design. This point can be seen easily by expanding the matrix:

(7) $FR_{1.1.1} = SDP_{2.2.2} + SDP_{2.2.4}$
$FR_{1.1.2} = SDP_{2.2.2} + SDP_{2.2.5}$

If the designer can fix $SDP_{2.2.2}$ before the others, then $SDP_{2.2.4}$ and $SDP_{2.2.5}$ can be adjusted independently to satisfy their corresponding functional requirements. In practice, this means that the mode transitions should be defined apart from the mode functionality itself. In addition, the implementation of each mode must be unaware of the existence of other modes. In this way, if one of the two modes needs to be changed or removed altogether, the remaining mode can be left unchanged. The traceability matrices can be used to identify subtle interactions and dependencies between the modes.

The environment assumptions and safety constraints affecting the SDPs are shown in Figure 7. Note how the environmental assumptions corresponding to $SDP_{2.2.4}$ and $SDP_{2.2.5}$ were successfully encapsulated: they do not have any common assumptions.

Figure 7 also contains SDPs that are not linked to any level 1 FR. For example, the establishment of a Safety Mode ($SDP_{2.2.3}$) does not answer the needs defined by any of the system goals but ensures that certain safety properties are preserved during the operation of the robot. This is a case of a SC that results directly in the establishment of a level 2 function. $SDP_{2.2.3}$ also serves to isolate the assumption MC6 that would otherwise affect $SDP_{2.2.4}$ and $SDP_{2.2.5}$.

5.2 Computer-Controlled Operations

This section includes provides an analysis of the functionality of the Computer mode. Rather than again providing a walkthrough of the design process, the relevant semantic coupling issues are presented directly.

The high-level functional requirements corresponding to this mode are:

MAPS-1.1.1.1:
MAPS shall generate appropriate commands to the Motor Controller to traverse the route provided by the Planner.

MAPS-1.1.1.1.1:
MAPS shall inform the Planner of the success or failure of the route traversal and the reason for any failure.

MAPS-1.1.1.1.2:
While in Computer Mode, all joystick deflections shall be ignored and the operator shall be informed when this occurs.

MAPS-1.1.1.2:
While in Computer Mode, MAPS shall maintain information about the position of the robot.
Rationale: MAPS will use the position information to correct the robot trajectory and to determine when the final position has been reached.

MAPS-1.1.1.3:
In Computer Mode, MAPS shall determine when the stabilizer legs should be deployed (or retracted) and issue the appropriate commands.

5.2.1 Position Determination

The first example analyzes the coupling created between the robot base motion control function and the position-finding function. The level 2 SDPs that define the latter function are:

MAPS-2.2.4.6: Scanner Query. MAPS obtains position information from the laser scanner. Position is calculated by […] [↑MAPS-1.1.1.2, ↑LS2]

MAPS-2.2.4.6.1: Position determination occurs prior to the beginning of each route segment when operating in Computer Mode. [↑MAPS-1.1.1.1, ↑MAPS-1.1.1.2, ↑LS2, ↑LS2.1, ↑LS3]
Rationale: MAPS must know the robot position before it can send a new relative displacement command to the Motor Controller.

MAPS-2.2.4.6.2: Position determination occurs following the completion of each route segment when operating in Computer Mode. [↑MAPS-1.1.1.1, ↑MAPS-1.1.1.2, ↑LS2, ↑LS2.1, ↑LS3]
Rationale: MAPS must know where the previous position command took the robot base. This information is used to detect errors and to provide feedback on previous actions in order to detect errors in carrying them out.

The location system environmental assumptions used for the previous SDPs are:

LS 2: Upon request, the laser scanner will provide the current position of the robot in the form of global location coordinates with an accuracy of TBD.
LS 2.1: The laser scanner can only take position readings while the robot base is immobile.
LS3: The bar code targets remain within the line of sight of the scanner at all times.

The corresponding overall traceability matrix is shown in Figure 8.

The traceability submatrix derived from the FR-SDP links follows:

(8)

$$\begin{pmatrix} FR_{1.1.1.1} \\ FR_{1.1.1.2} \end{pmatrix} = \begin{bmatrix} 0 & * & * \\ * & * & * \end{bmatrix} \begin{pmatrix} SDP_{2.2.4.6} \\ SDP_{2.2.4.6.1} \\ SDP_{2.2.4.6.2} \end{pmatrix}$$

Equation (8) clearly represents the case of a coupled design. The origin of this coupling can be traced back to LS2.1. Because the scanner can only take position readings after the robot base has stopped, the motion and position finding functions are interdependent. Changes in some of the environmental assumptions can make the situation worse. For example, assume that LS3 does not hold anymore. This change would mean that in some cases, the scanner would not be able to find the bar code targets needed to perform its function. MAPS software engineers might be tempted to include additional motion controlling functionality to tackle this shortcoming. Unfortunately, the problem is nontrivial because it is hard to direct the robot to another position if there is a high degree of uncertainty about its current position. Despite having successfully isolated the LS environmental assumptions from the rest of the design (see Figure 8), the semantic coupling identified in equation (8) makes the entire system susceptible to changes in them.

A possible solution to this problem is the substitution of the laser scanner by a Local Positioning System (LPS). This change would uncoupled the design and thus create a more robust system. It is unlikely that the decision to modify the robot would be based solely on this consideration; other factors (cost, equipment availability) are likely to play an important role as well. The key is to provide engineers with the information necessary to identify and evaluate the trade-offs of crucial importance to the long-term success of their system.

5.2.2 Robot Base Stabilization

The final example involves the responsibilities of MAPS for robot base stabilization. The related part of the MAPS intent specification follows:

MC1: When commanded to do so, the motor controller will provide power to the motor, which will drive the robot wheels

GUI1: The GUI will provide the operator with enough information about the status of the robot and the work area that the operator is able to avoid hazards. This information includes movements commanded by the Planner.

PL1: All automatic robot operations will be directed and coordinated by the Planner

SL1: The stiff legs provide the robot base with the stability needed to perform all the currently anticipated robot arm operations. The stiff legs will be able to stabilize the robot during tile servicing (manipulator motion*).*

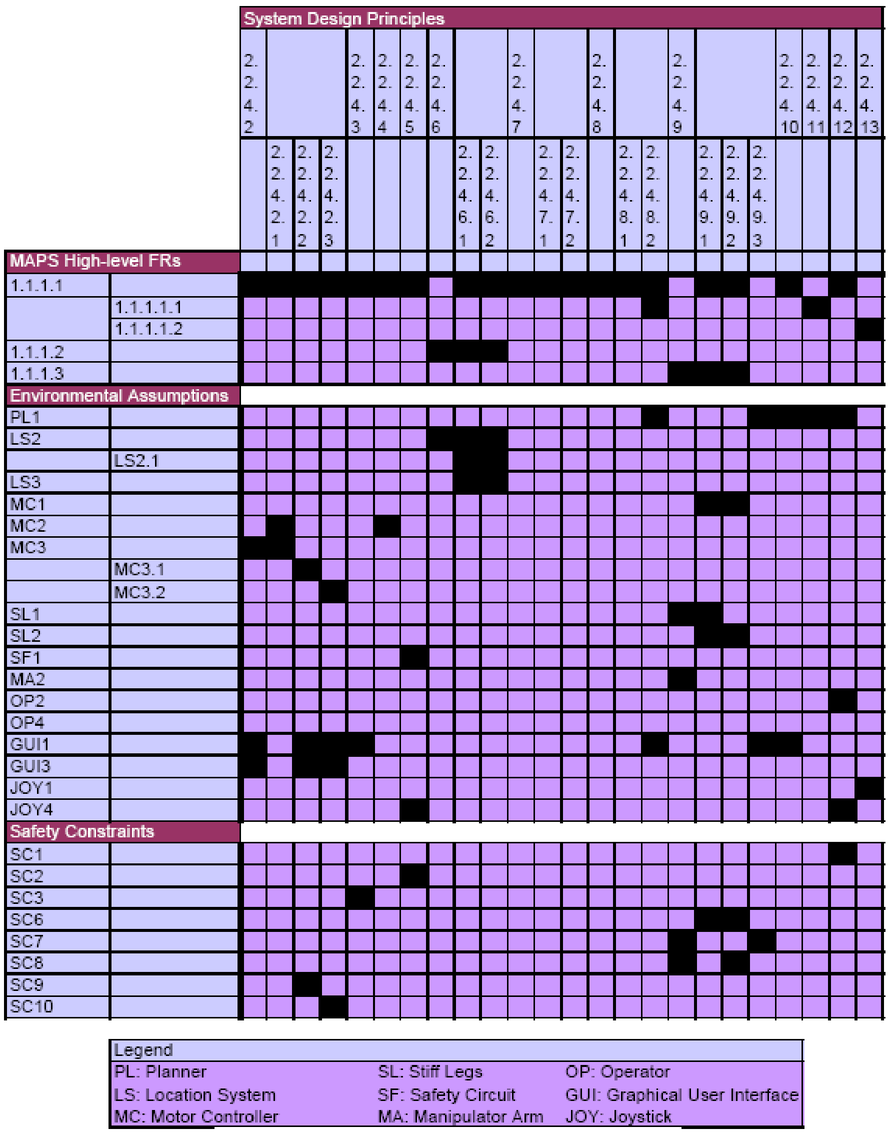
SL2: The stiff legs are not able to stabilize the robot while in motion.

MA2: The manipulator arm controller will provide information about the position of the arm directly to MAPS.

SC6: The mobile base must not move when the stabilizer legs are extended.
Rationale: Damage can occur to the robot if movement is attempted with the legs extended.

SC7: The manipulator arm must not be extended when the stabilizers are retracted.

SC8: Stabilizer legs must not be retracted until the manipulator arm is fully stowed.



		System Design Principles																							
		2.2.4.2				2.2.4.3	2.2.4.4	2.2.4.5	2.2.4.6			2.2.4.7			2.2.4.8			2.2.4.9				2.2.4.10	2.2.4.11	2.2.4.12	2.2.4.13
			2.2.4.2.1	2.2.4.2.2	2.2.4.2.3					2.2.4.6.1	2.2.4.6.2		2.2.4.7.1	2.2.4.7.2		2.2.4.8.1	2.2.4.8.2		2.2.4.9.1	2.2.4.9.2	2.2.4.9.3				
MAPS High-level FRs																									
1.1.1.1		■	■	■	■	■	■	■		■	■	■	■	■	■	■	■		■	■		■		■	
	1.1.1.1.1																■						■		
	1.1.1.1.2																								■
1.1.1.2									■	■	■														
1.1.1.3																		■	■	■					
Environmental Assumptions																									
PL1																	■				■	■	■	■	
LS2									■	■	■														
	LS2.1									■	■														
LS3										■	■														
MC1																			■	■					
MC2			■				■																		
MC3		■	■																						
	MC3.1			■																					
	MC3.2				■																				
SL1																		■	■						
SL2																			■	■					
SF1								■																	
MA2																		■							
OP2																								■	
OP4																									
GUI1		■		■	■	■											■				■	■			
GUI3		■		■	■																				
JOY1																									■
JOY4								■																■	
Safety Constraints																									
SC1																								■	
SC2								■																	
SC3						■																			
SC6																			■	■					
SC7																		■			■				
SC8																		■		■					
SC9				■																					
SC10					■																				

Legend		
PL: Planner	SL: Stiff Legs	OP: Operator
LS: Location System	SF: Safety Circuit	GUI: Graphical User Interface
MC: Motor Controller	MA: Manipulator Arm	JOY: Joystick

Figure 8: Computer Control Mode Traceability Matrix

MAPS-2.2.4.9: Leg Deployment and Retraction. Upon reaching the final destination, MAPS deploys the stabilizer legs and later retracts them (upon the Planner's request) once the manipulator arm is stowed. [↑MAPS-1.1.1.3, ↑SL1, ↑MA2, ↑SC7, ↑SC8]

MAPS-2.2.4.9.1: After a move that has correctly positioned the robot base in the work area, MAPS turns off the wheel motors and then deploys the legs. [↑MAPS-1.1.1.1, ↑MAPS-1.1.1.3, ↑SL1, ↑SL2, ↑MC1, ↑SC6]

<u>Rationale</u>: The motor is turned off so no accidental motion command sets the base in motion with the stabilizers legs deployed.

MAPS-2.2.4.9.2: When a move is commanded, MAPS retracts the stabilizer legs before turning on the wheel motors. [↑MAPS-1.1.1.1, ↑MAPS-1.1.1.3, ↑SL2, ↑MC1, ↑SC6, ↑SC8]

<u>Rationale</u>: see MAPS-2-2.4.9.1

MAPS-2.2.4.9.3: In the event the stabilizer retraction or deployment fails, MAPS notifies the operator and the Planner of the failure. [↑GUI1, ↑PL1, ↑SC7]

<u>Rationale</u>: The Planner is informed about the error so it does not send any arm motion commands. The operator is informed about the error so the failure can be diagnosed and repaired.

As before, the traceability submatrix can be derived directly from the FR-SDP links:

(9)

$$\begin{pmatrix} FR_{1.1.1.1} \\ FR_{1.1.1.3} \end{pmatrix} = \begin{bmatrix} 0 & * & * \\ * & * & * \end{bmatrix} \begin{pmatrix} SDP_{2.2.4.9} \\ SDP_{2.2.4.9.1} \\ SDP_{2.2.4.9.2} \end{pmatrix}$$

As in the previous case, equation (9) indicates the design is coupled. The origin of this coupling is the need to satisfy safety constraints related to the instability of the robot base. Although on the surface the software logic appears to be relatively simple here, it can become quite complex due to the need to keep track of and control several unrelated subsystems and the related timing and fault tolerance issues. Changes to this logic could cause major disruptions to a project or rippling effects that delay deliverables or lead to budget overruns. Changes in this early system design phase could prevent these problems, and the semantic coupling information can assist the engineers in making the necessary tradeoff decisions. For example, a first step in decoupling this design might be to use a power interlock so the energy supply to the Motor Controller is shut off automatically whenever the manipulator arm power is turned on, thus reducing the safety responsibilities of the software.

6. Semantic Coupling Contributions and Comparison with Other Approaches

The concept of semantic coupling presented has similarities with the concepts of module coupling, design structure matrices, and others. All use basic principles from matrix algebra and apply them to different stages and representations of a system or software design process. However, it is important to underline how semantic coupling differs from a theoretical and practical point of view.

The definition of semantic coupling within the intent specifications framework provides the ability to record different mappings between the six levels. This provides the potential to to analyze the different conceptual processes that take place as the designers go from the very preliminary high-level requirements (level 1) until they reach the production of actual software artifacts (level 5) or even to the system operational use (level 6). Thus, intent specifications are able to support mappings to assess the three kinds of modularity defined by Baldwin and Clark [5]: modularity-in-design, modularity-in-use and modularity-in-production. In contrast, most previous efforts (module coupling and DSM as applied to software so far) focus on studying the internal couplings of an already complete software artifact at the very source-code level. This approach produces very useful information regarding the need to define appropriate coding practices, establish modules, or even computer languages. However, these applications are not where the real problem is today; as stated previously, the remaining great challenge lies in the requirement definition phase. Therefore, it is important to apply decoupling at the earliest stages of a design and follow through to implementation. Current approaches produce diagnosis data but too late: applying retroactive decoupling measures in a complex software system design is not a viable option.

Another differences with the DSM approach is that the matrices obtained from an intent specification will not be square in general. Rather than a theoretical inconvenience, this is a fair reflection of the design processes that take place as a complex system is implemented. Moreover, this is not an obstacle to assessing the level of coupling in a system: the case study in the previous section demonstrated this fact. Note that the hierarchical organization used in intent specifications provides significant support to structure the tasks required to assess the feasibility of decoupling a given system design.

One of the main objectives of the semantic coupling approach is to enable software and system engineers to think about their design properties in terms of ideas that are close to them. For this reason, we focus on the use of the traceability matrix, which is an artifact available already in most projects of any significant size. The creation of unfamiliar constructs, such as the DSM, is likely to face difficulties in being accepted by mainstream software system engineering. The use of a natural language in the first two levels of the intent specification further enhances the ability of engineers to use their domain knowledge in finding ways to uncouple the system design.

Another difficulty is raised by the use of design constraints in the Design Rule Theory. While this is a fully viable solution, it again forces the developers to perform unfamiliar tasks and does not provide a resolution for the identified couplings because there is no certainty all the derived design constraints can be satisfied. For this reason, our semantic coupling approach emphasizes the need to review the mappings between two levels of an intent specification: undesired couplings are readily identified and designers are challenged to provide design solutions that solve the undesired coupling in the lower level. The results of this process, either positive or negative, are then recorded in the rationale field of each design parameter.

By placing the traceability matrices at the core of the design process, system or software engineers are also encouraged to think carefully about what traceability is and what it means with respect to their system. This means that the natural design iterations that take place will not only improve the system design itself but will also optimize the completeness and correctness of the traceability matrices. Improved traceability is an added value in itself as discussed in Section 2.

Work on semantic coupling has not so far considered the use of quantitative measures of coupling or the inclusion of formal / semi-formal methods to search the design space for uncoupled solutions. The possibility of adapting such techniques to the lower levels (levels 3, 4, 5) of an intent specification canl be the subject of future research.

Despite all the differences, the semantic coupling and DSM approaches share the same objective: minimizing the interdependencies in a given system. The vast literature in DSM supporting the idea of decoupling (in theory and in real-life examples) as a important system metric and the already successful use of intent specifications in safety-critical systems provide reassurance of the importance of the semantic coupling approach to the requirements change problem.

Overall, semantic coupling, as applied to safety-critical software systems, promotes in a novel fashion, three valuable system properties:

- *Changeability,* by minimizing the repercussions associated to requirement changes
- *Traceability,* by supporting the complex design assurance processes used in many certification activities (e.g.: DO-178B and DO-254 in the aviation industry)

- *Safety*, by reducing unnecessary interdependencies between different parts of a system, thus limiting the potential for unsafe interactions among them [27].

7. CONCLUSIONS

The need for software engineering derives from the need to manage complexity. Complexity can be managed intellectually by partitioning, establishing hierarchies, and maximizing independence among components. The goal of intent specifications is to increase intellectual manageability by applying the concepts found by cognitive psychologists to be key to success in complex problem-solving activities. This paper has introduced and defined a concept of semantic coupling and shown how it can be operationalized using traceability matrices. Semantic coupling is related to the difficulty of changing system requirements and reducing coupling should reduce the rippling effects associated with requirements changes.

Semantic coupling at the higher system levels translates to structural coupling and cohesion at the software design and implementation levels and should also ease the structural decoupling process by decoupling functions or identifying coupled functionality at the higher levels. Any such decoupling involves design tradeoffs, but making these tradeoffs and their implications clear will be helpful to those generating system and software designs.

For small projects, the advantage of the approach lies in the intellectual processes that the domain experts must follow to produce the traceability matrices. For larger and more complex projects, the traceability matrices themselves play an important role. As the number of developers increases, creating a common convention to communicate coupling characteristics becomes increasingly important. During maintenance the need is even greater. Used in an intent specification framework, traceability matrices provide traditional functional traceability, but they also assist in system design and tradeoff evaluation, designing for safety and safety evaluation and assessment, interface design, etc.

Providing techniques and tools to support the creation and navigation of large traceability matrices is a future research goal as is the more general goal of increasing our understanding of how to structure complex systems to allow us to stretch the limits of complexity of the systems we can build and maintain successfully.

8. REFERENCES

[1] Allen, Edward B, Taghi M. Khosgoftaar, and Ye Chen. Measuring Coupling and Complexity of Software Modules: An Information Theory Approach, *7th Int. Software Metrics Symposium*, 2001, p. 124.

[2] Aoki, M., *Towards a Comparative Institutional Analysis*, Chapter 4, MIT Press, 2001.

[3] E. Arisholm, L. Briand, and A. Foyen, Dynamic Coupling Measurement for Object-Oriented Software, *IEEE Trans. on Software Engineering*, 30(8): 491-506, 2004.

[4] Baldwin, C.Y., and Clark, K. B. *Design Rules, Vol. 1: The Power of Modularity*. The MIT Press, 2000.

[5] Baldwin, Carliss Y. and Kim B. Clark. "Modularity in the Design of Complex Engineering Systems." *In Complex Engineered Systems: Science Meets Technology*, edited by Ali Minai, Dan Braha and Yaneer Bar Yam. New England Complex Systems Institute Series on Complexity. N.Y.: Springer-Verlag, 2006.

Field Code Changed

[6] Cai, Y. and Sullivam K. J., “Simon: Modeling and Analysis of Design Space Structures”, *Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering,* pp. 329-332, 2005.

[7] Dahlstedt, Asa G., and Persson, A., "Requirements Interdependencies- Moulding the State of Research into a Research Agenda", *Ninth International Workshop on Requirements Engineering: Foundation for Software Quality* (REFSQ '03).

[8] Dardenne, A. van Lamsweerde, A., and Fickas, S., “Goal-Directed Requirements Acquisition”, *Science of Computer Programming*, Vol. 20, 1993, 3-50.

[9] Darimont, R., Delor, E., Massonet, P., and van Lamsweerde, A., “GRAIL/KAOS: An Environment for Goal-Driven Requirements Engineering”, *Proc. ICSE’98 - 20th Intl. Conf. on Software Engineering*, Kyoto, April 1998, vol. 2, 58-62.

[10] DePauw, W., D. Kimmelman, et al, Visualizing Object-Oriented Software Execution. *Software Visualization*, J. Stasko. Cambridge Massachusetts and London England, MIT Press: 329-367, 1997.

[11] Dick, J., Rich Traceability, *in Proceedings of the Automated Software Engineering*, *Traceability Workshop*, 2002.

[12] Dowling, K.R., Bennett, R., Blackwell, M.,Graham, T., Gatrall, S., O’Toole, R., and Schempf, H. A Mobile Robot System for Ground Servicing Operations on the Space Shuttle. Cooperative Intelligent Robotics in Space III, *Proceedings of the Meeting, Boston, MA, Nov. 16–18, 1992 (A93-29101 10-54)*. Bellingham, WA, Society of Photo-Optical Instrumentation Engineers, 1992, p. 98–09.

[13] Dulac, Nicolas, Thomas Viguier, Nancy Leveson, and Margaret-Anne Storey, On the Use of Visualization in Formal Requirements Specification, *Int. Conference on Requirements Engineering*, Essen Germany, 2002.

[14] Egyed, Alexander, A Scenario-Driven Approach to Trace Dependency Analysis, *IEEE Trans. on Software Engineering*, 29(2): pp. 116-132, Feb. 2003.

[15] Eppinger, S., D. Whitney, et al. “A Model-base Method for Organizing Tasks in Product Development.” *Research In Engineering Design* 6: 1-13, 1994.

[16] Garud, R. and Kumaraswamy, A., Technological and Organizational Designs to Achieve Economics of Substitution, *Strategic Management Journal*, Vol.17, pp.63-65, 1995.

[17] Glaser, R., Chi, M.T.H., and Farr, M.J. *The Nature of Expertise*. Erlbaum, Hillsdale, New Jersey, 1988.

[18] Gotel, O. and Finkelstein, A., “Extended Requirements Traceability: Results of an Industrial Case Study”, *In Proc. 3rd International Symposium on Requirements Engineering (RE97),* IEEE Computer Society Press, p. 169-178, 1997.

[19] Gunter, Carl A. Abstracting Dependencies Between Software Configuration Items, *ACM Trans. on Software Engineering Methodology* (TOSEM), 9(1):94-131, Jan. 2000.

[20] Holland, John H., *Adaptation in Natural and Artificial Systems*, 2nd Ed. MIT Press, Cambridge, MA, 1992.

[21] Jaffe, M.S., Completeness, Robustness, and Safety in Real-Time Software. Ph.D. Dissertation, University of California Irvine, 1988.

[22] Kelleher, Justin. A Reusable Traceability Framework Using Patterns, *Proc. 3rd Int. Workshop on Traceability in Emerging Forms of Software Engineering*, Long Beach, CA, Nov. 8, 2005.

[23] Knight, J., Advances in Software Technology since 1992 and a Modest Proposal for their Incorporation into Certification, presentation to the *National Software and Airborne Electronic Hardware Conference*, Denver, CO. August 20-21, 2008.

[24] Kotonya, G., and Sommerville, I., *Requirements Engineering: Processes and Techniques*, John Wiley & Sons, 1998.

[25] Leveson, Nancy G., *Safeware: System Safety and Computers*, Boston: Addision-Wesley, 1995.

[26] Leveson, N. G. Intent Specifications: An Approach to Building Human-Centered Specifications. *IEEE Transactions on Software Engineering*, Vol. 26, No. 1, pp. 15–35, January 2000.

[27] Leveson N.G. Completeness in Formal Specification Language Design for Process-Control Systems. *ACM Formal Methods in Software Practice*, Portland, August 2000.

[28] Leveson, N.G., Reese, J.D. TCAS II Intent Specification. http://sunnyday.mit.edu/papers/intent.pdf

[29] Leveson, N.G., "A New Accident Model for Engineering Safer Systems," *Safety Science*, Vol. 42, No. 4, pp. 237-270, 2004.

[30] Lopes, C.V. and Bajracharya S.K, "An Analysis of Modularity in Aspect Oriented Design", *In 4th International Conference on Aspect-Oriented Software Development*, pp. 15-26, New York, NY, USA, 2005.

[31] MacCormack, A., Rusnak, J., and Baldwin, C. Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code. Harvard Business School Working Paper Number 05-016

[32] Marcus, Andrian and Jonathan Maletic, Recovering Documentation-to-Source-Code Traceability Links Using Latent Semantic Indexing, *International Conference on Software Engineering* (ICSE), 2003.

[33] Myers, G. J., *Reliable Software through Composite Design*, Petrocelli / Charter, 1975.

[34] Myers, G. J. *Composite/Structured Design*, Van Nostrand Reinhold Company, New York, NY, 1978.

[35] Offutt, Jeffrey, Mary Jean Harrold, and Priyadarshan Kolte. A Software Metric System for Module Coupling, *Journal of Systems and Software*, 20(3): 295-308, March 1993.

[36] Owens, Brandon D., Margaret Stringfellow Herring, Nicolas Dulac, Nancy Leveson, Michel Ingham, and Kathryn A. Weiss. Application of a Safety-Driven Design Methodology to an Outer Planet Exploration Mission, *IEEE Aerospace Conference*, Big Sky, Montana, March 2008.

[37] Parnas, D. L., "On the criteria to be used in decomposing system into modules," *Communications of the ACM*, Vol. 15, No. 12, December 1972 pp. 1053-1058.

Field Code Changed

[38] Pohl, K, *Process-Centered Requirements Engineering*, John Wiley & Sons Inc, 1996.

[39] Pimmler, T. U. and Eppinger, S., Integration Analysis of Product Decompositions. *Proceedings of the ASME DETC Design Theory and Methodology Conference*, Minneapolis, Minnesota, 1994.

[40] Ramesh, B. and Jarke, M., “Toward Reference Models for Requirements Traceability”, *IEEE Transactions on Software Engineering*, Vol.27, no1., p. 58-93, 2001.

[41] Rasmussen, J., “The Role of hierarchical knowledge representation in decision making and system management”, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-15, no. 2, March/April 1985.

[42] Rotenstreich, S. Toward Measuring Potential Coupling, *Software Engineering Journal*, 9(2):83-90, March 1994

[43] Sangal, N., Jordan, E., Sinha, V. , and Jackson, D. Using Dependency Models to Manage Complex Software Architecture*. Proceedings of the 20th annual ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages, and Applications*, pp. 167-176, 2005.

[44] Smith, R. P. and S. Eppinger, “Identifying Controlling Features of Engineering Design Iteration.” *Management Scie*nce 43(3), 1997.

[45] Spanoulakis, G, et.al., Rule-Based Generation of Requirements Traceability Relations, *Journal of Systems and Software*, 72(2): 105-127, 2004.

[46] Stevens, W. P., Myers, G. J. and Constantine, L. L., "Structured Design," *IBM Systems Journal*, Vol. 13, No. 2, May 1974.

[47] Steward, D. V. “The design structure system: a method for managing the design of complex systems.” *IEEE Transactions on Engineering Management*, EM-28(3), pp.71-74, 1981.

[48] Sullivan, K., Cai, Y., Hallen, B., and Griswold, W. G., “The Structure and Value of Modularity in Software Design”, *SIGSOFT Software Engineering Notes*, 26(5):99-108, Sept. 2001.

[49] Sullivan, K., Griswold, W., Song, Y. et al. Information Hiding Interfaces for Aspect-Oriented Design. *In ESEC/FSE '05*, Sept 2005

[50] van Lamsweerde, A., Goal-oriented requirements engineering: a guided tour, *Proceedings. Fifth IEEE International Symposium on Requirements Engineering*, pp.249-262, 2001.

[51] Vicente, K.J., Christoffersen, K., and Pereklit, A..Supporting operator problem solving through ecological interface design. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(4):529--545, 1995.

[52] Wilkie, F.G. and B.A. Kitchenham, Coupling Measures and Change Ripples in C++ Application Software, *Journal of Systems and Software*, 52(2-3):157-164, June 2000.

[53] Yu, E., “Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering”, *Proc. RE-97 - 3rd Int. Symp. on Requirements Engineering*, Annapolis, 1997, 226-235.

[54] Yue, K., “What Does It Mean to Say that a Specification is Complete?”, *Proc. IWSSD-4, Fourth International Workshop on Software Specification and Design*, Monterey, 1987.

[55] Yourdon, Edward, and Constantine, Larry, *Structured Design*, Prentice Hall, Englewood Cliffs, N.J., 1979.