INTERNATION CONFRENCE ON ENGINEERING DESIGN
ICED 05 MELBOURNE, AUGUST 15-18, 2005

A SITUATED QUESTION-DRIVEN AND MODEL-BASED APPROACH TO DESIGN REASONING

Dr. Ulf Sellgren

Abstract

A complex system is a system with many components and interconnections, interactions, or interdependencies that are difficult to describe, understand, predict, manage, design, or change. Complex technical systems are characterized by the level of complexity and uncertainty. Uncertainty appears mainly in system behaviors and the ways different aspects of a system change with time. The uncertainty, which can be seen as a direct consequence of the complexity and it increases with the scale and the scope of the system, will normally decrease as we gain insight and knowledge about the system.

Engineering design of complex technical systems depends much more heavily on computer simulation than design of smaller and less complex products. The main objectives for simulations is to assist design reasoning by helping to identify unintended and accidental interactions between components and to assess and verify the system in early design stages. The special character of product development and engineering design require that this reasoning process is situated and that the model definitions are based on "systems thinking". This paper elaborates on a formal basis for model-based design reasoning and presents a formal knowledge framework for model-based design reasoning.

Keywords: Design question, knowledge management, model-based design reasoning

1 Introduction

It is widely agreed that technical systems increase in size, scope, and complexity as a result of globalization, new technologies, increasing customer expectations, and tougher social requirements. A complex system is a system with many components and interconnections, interactions, or interdependencies that are difficult to describe, understand, predict, manage, design, or change. A technical system, which is a system designed by humans, is a complex set of physical components and interfaces. An interface may be defined as "a boundary or interconnection between systems or their components that define or support interrelationships; interfaces may be intended or unintended" [1]. Interface specification and design are concerned with making the components interoperable, including making the interaction between human(s) and technical system intuitive and unambiguous. Components are parts of a system relative to that system. A component can be a system too if it contains other components (and interfaces). The purpose and the intended physical behavior of the individual components of a technical system are often well-defined (although possibly poorly understood), but the interactions, which take place at the interfaces between the components, are frequently difficult to describe and understand, thus, making the behavior of the system difficult to predict and, consequently,

difficult to design. The two most characteristic aspects of a complex technical system are complexity and uncertainty [1]. Uncertainty appears mainly in system behaviors and the ways different aspects of a system change with time. A system can be classified *as behaviorally complex* if its behavior is difficult to predict, analyze, describe, or manage. If the number of parts of a system is large and the interconnections between its parts is hard to describe briefly, the system is *structurally complex*. The uncertainty, which also can be seen as a measure of the complexity, increase with the scale and the scope of the system. The uncertainty will normally decrease as we gain insight and knowledge about the system through (virtual and physical) prototyping and use.

Aristotle's postulated that our knowledge resides in the questions we ask and the answers we can provide [2]. Most of the published question-research (education, artificial intelligence, cognitive psychology) has been focused on convergent thinking, where the questioner is assumed to converge on "the facts". By studying questions that are raised in design situations, [3] identified an important class of questions, which he labeled "generative design questions". These questions are characteristic of divergent thinking, where the questioner is attempting to diverge away from the facts to the possibilities that can be generated from them. "Engineering is question intensive" [3] and, consequently, question-asking is a fundamental cognitive mechanism in design reasoning and it can be treated as a process.

A (computer-based) model may be viewed as a complex object composed of explicit and implicit knowledge. Anyone with access to a model can pose a question to the model (e.g., on the behavior and the performance of the modeled object) and hopefully formulate an answer to that question. Modeling is a cognitive act of structuring and simplifying information. Computer-based models are consequently cognitive constructs that assist us to manage technical complexity and reduce uncertainty. A model is thus a cognitive tool for design reasoning.

Each model must be targeted for a specific purpose [4]. The purpose of a model is defined by the engineering question and its context. Furthermore, models are related to each other in one or several ways (e.g., causally related) and they thus form a structure of knowledge. Design reasoning does not only transform questions into answers, which represent new knowledge, but also transforms questions into questions. Further more, in engineering design an answer may exist but the question is not known.

A research challenge is to define a formal process to enable and assist model-based design reasoning. The special character of product development and engineering design require that this process is situated [5] and that the definition of the models are based on "systems thinking". This paper discusses the formal basis and presents a formal knowledge framework for question-driven and model-based design reasoning.

2 Background to systems engineering

Systems thinking involves the use of various techniques to study systems of many kinds. It includes studying things in a holistic way. It aims to gain insight into the whole by understanding the linkages, interactions, and processes between the elements that comprise the whole system. The goal of *systems engineering* is to transform mission, operational requirements or remediation requirements into system architecture, performance parameters, and design detail [6]. A systems view implies that the components and the interfaces of a system are treated in a similar way.

Whenever the engineering task must reconcile technical considerations with non-technical design criteria, it seems to be necessary to use diverse strategies and techniques [7]. This is in contrast to the prescriptive design methodology view of applying a uniform, technically-based decomposition approach (e.g., conceptual design with a "function structure" in [8]. Because most of the actual design work in industry also can be characterized as adaptive design, many practitioners in industry seriously question the role of the function concept as a useful entity in product development.

Nevertheless, the functionality of an artifact describes and represents a part of the designer's intention or design rationale [9]. Based on an extensive classification study of highly complex natural systems (e.g., biological systems and cosmic systems) and engineering systems (i.e., systems that are human designed and having both significant human and technical complexity) [10] found that function type as originally proposed in [11], [8], and [12] is the only technical attribute able to differentiate among engineering systems. Function is consequently a characteristic product attribute that captures important knowledge about an existing product or component whenever a task involves adaptive design, re-design, and/or design by analogy.

The FBS framework [13] is a formal representation of the processes of designing. It represents the developing design in different states. The basic assumption is the existence of three classes of variables: function, behavior, and structure. They are linked together by eight classes of processes, which transform one class into another (see figure 1).

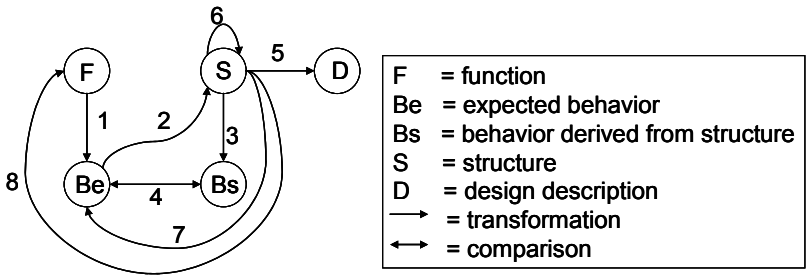

Figure 1. The FBS-framework, after Gero [13]

The eight processes in the FBS framework that are shown in figure 1 are claimed [13] to be fundamental for all designing are ; *formulation* (1), s*ynthesis* (2), a*nalysis* (3), *evaluation* (4), *documentation* (5), *reformulation type 1* (6), *reformulation type 2* (7), and *reformulation type 3* (8).

Qian and Gero [14] defined two designs to be analogous if they have a similar function or similar behavior. Two analogous designs may or may not have similar structures. By defining associations between function, behavior, and (product) structure, they presented a function-behavior-structure (FBS), or a causal knowledge, formalism of design knowledge representation and utilized it in a framework for analogy-based creative design (ABD).

Prabhakar and Goel [15] introduced the idea of function as resulting from the interaction between a device and its environment, and developed a functional representation scheme, referred to as the Environmentally-bound Structure-Behavior-Function or ESBF model, and a processing strategy, called Environmentally-driven Adaptive Modeling (EAM), for adapting existing design to new environments. The ESBF model views a device function as an abstraction of the interactions of the device with its external environment. This is in contrast to most functional representation schemes, including the SBF-model, where function is viewed as an abstraction of the internal behavior of the device.

To represent designing in a dynamic world, Gero and Kannengiesser [16] included the environment and reconstructed the eight fundamental processes in the original FBS framework and proposed the situated FBS-framework.

2 The function-behavior-implementation-environment (FBIE) model

The *purpose* of presenting an adapted systems model of components and interfaces is to enable design, communication, and verification of the components of technical systems *and* the interfaces between the components and between the technical system, its environment, and the human operator(s)/user(s). This paper is a contribution to the development of an integrated, logically rigorous representation and management of complex technical systems and the system models that are used for performing design reasoning.

The chosen *approach* to model management is to develop a framework that is based on a modified and extended FBS-model. In order to have consistent taxonomy for the aspects relevant to systems engineering, the term structure (S in the FBS-model) is replaced with the term implementation (I). Further more, to acknowledge the situated nature of product development, the environment domain (E) is included. Consequently, we are further on referring to the *FBIE domain-model*. Figure 2 shows the four primary FBEI-domains and the associated (the original) customer domain that represents who the product is intended for and all the requirements that are directly related to the customer in a broad sense (i.e., the end-user(s), the realizing company, and the society through its legislation). The requirements on the FBIE-model is to enable components and interfaces between components in mechanical, electronics, hydraulics, and software systems to be treated in a consistent way, and to allow environment-machine and human-machine interaction aspects to be treated in a similar way. The FBIE-model is intended to enable efficient interface definition, representation, and use in engineering in general and modeling and simulation of product behavior and performance in particular.

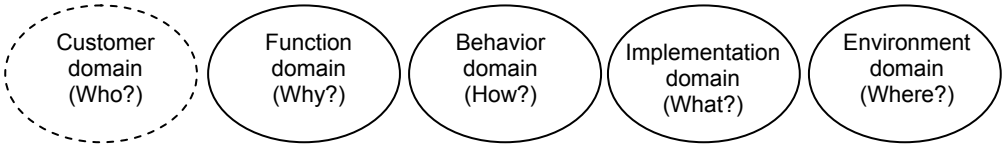

Figure 2. The customer domain and the four primary domains in the FBIE-model

By using functional reasoning, a desired function can be mapped onto its implementation structures straightforward or via (intended) behaviors which are taken as the bridge between the function and the implementation (structure). *Function* is regarded here as the intended use of an artifact. *Intended behavior* is considered as the means by which (or how) a function is achieved, and it expresses physical state transitions or physical phenomena (principles). *Implementation* refers to the physical components or forms and structures that are utilized to achieve the intended behavior. *Actual behavior* is normally the behavior that can be observed when an implementation structure is influenced by its environment. This type of behavior can be classified as an exogenous or indirect behavior [14]. Sometimes the actual behavior can be derived directly from the implementation structure without any external effect. In this case the actual behavior can be characterized as structural or direct [14]. Actual behavior can be decomposed into *intended behavior*, *unintended behavior* (side effects), and *accidental behavior* (faults or errors). Function, behavior and implementation (structure) are referred to as product, component, and interface characteristics. Effects and properties that are external to the product but have a significant

influence on the actual behavior of the product are referred to as environment characteristics. End-users and other human agents that interact with a product are interactive part(s) of the products environment. If the scope of an investigation is the interaction between two components in a product, the other components and interfaces of the product may be regarded as environment, i.e. objects that are external to the focused system.

2.1 Function domain

The function tells what the design is intended to do (i.e., the purpose) and is often used for specifying design requirements, sometimes referred to as functional requirements (e.g, [17]). This purpose can be defined as relationships between inputs and outputs of energy, mass, and information, or as a change in the fluxes thereof (e.g. [14]). Manipulation of flows (or fluxes) involves actions. Function can, thus, be characterized by two kinds of variables – action and flow – that can be classified into two taxonomy hierarchies. The action taxonomy represents a hierarchy of verbs (e.g., store, transport). In the flow taxonomy, the flow superclass is categorized into material, energy, and information subclasses. These three subclasses can be categorized further. For example, energy may be mechanical, which may be further classified as kinetic or potential, and so on.

Actions and flows can be decomposed in many different ways, but some decompositions are more convenient to use than others. In [18], Little et al. refer to their set of actions and flows as a basis set. The mathematical definition of basis requires that the set spans the space and the components of the set are linearly independent. For example, in many engineering situations the eigenvectors of a dynamic is a convenient and thus attractive basis set. The set of functions and flows proposed in [18] and further elaborated on by Stone [19] is a basis set in a qualitative sense. It is further on referred to as the *Little function base set* and it is used as if it was a base set in a strict sense

Figure 3 shows a qualitative causal relation graph of the function domain and its relations to the other domains of the FBIE-model. *F1* is a required function that can be decomposed in an interactive subfunction $f11_i$ and a technical subfunction $f12_t$. The technical subfunction is realized by the intended behavior $b1_i$ and implemented in *i1*. The interactive subfunction defines a user aspect *e1* of the environment and its relation to the implementation *i1*. The implemented structure *i1* and the environment *e1* induces an actual behavior $b1_a$ that is decomposed has to be evaluated against the intended behavior $b1_i$,. This evaluation also shows an unintended component of the actual behavior, which requires a counter-measure. This counter-measure is defined by the new function *f2* that is realized by a new intended behavior $b2_i$ and implemented in the new and more detailed structure *i2*.

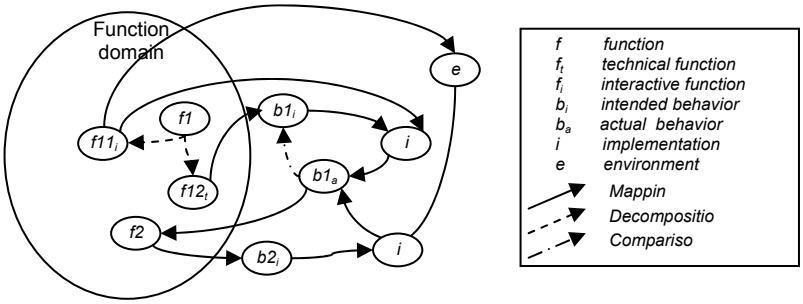

Figure 3. The function domain as a qualitative causal relation graph

The actual function objects in the functional domain are typically results of analysis of end-user, corporate and regulatory requirements. The objective is to define the purpose of the product and to use functional decomposition as a means to reduce complexity by breaking the development task into smaller and manageable pieces.

The preferred approach to create, verify, validate, decompose, and compare required functions is to have function attributes that are pointers to the context (e.g., the project task description) and the list of stated customer requirements.

2.2 Behavior domain

Multiple functions and multiple side effects, such as vibration, friction, wear, heat, fatigue, and crack growth, are fundamental characteristics of many technical products [20]. Behavior is defined here as a change of state of an object or of the relation between the object and the external world. From a functional perspective, behavior manifests functionality and state changes of the implemented structure [14]. This aspect of behavior, which is strongly related to intention, or function, reveals the meaning of the implementation with or without the relations to external effects, is referred to as the intended behavior and it is causally derived from the required function.

In the FBIE-model, a distinction is made between three categories of behavior: intended behavior, unintended behavior, and accidental behavior. *Intended behavior* is thus the means by which a function is achieved. *Unintended behavior* is a side-effect that may require an additional sub-function to counteract, eliminate, or reduce the undesired side-effect. Unintended behavior is normally caused by physical interactions between the system components, i.e. they are focused at the interfaces between the components (e.g., friction-induced thermal effects between surfaces in contact), An *accidental behavior* is an unintended behavior that is caused by an accidental relation/interaction between product pieces (e.g., a cable placed too close to a hot component such as a motor block). An accidental behavior is a design error. Engineering design is much about maximizing the intended behavior of the evolving product, minimizing the unintended behavior, and avoiding the accidental behavior.

Figure 4 shows a qualitative causal relation graph of the behavior domain and its relations to the other domains of the FBIE-model. $B1_i$ is the intended behavior required to realize function *f1*. It is implemented in *i1*. $B1_i$ is decomposed into sub-behaviors $b11_i$ and $b12_i$. The implemented structured *i1* and the environment *e1* induces an actual behavior $b11_a$ that is decomposed into an intended actual behavior $b11_{ai}$ that has to be evaluated against $b11_i$, an unintended behavior $b11_{au}$, and an accidental behavior $b11_{aa}$. The accidental behavior requires a modification in *i1*. The unintended behavior is decided unacceptable and it thus requires that a new function (*f2*) is introduced. This function is realized by a new intended behavior $b2_i$ and implemented in the new and more detailed structure *i2*.

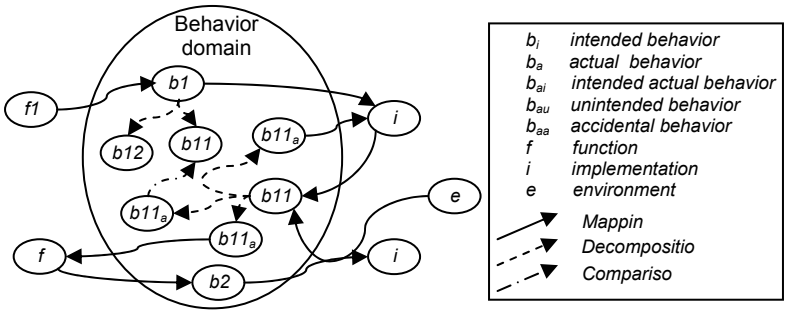

Figure 4. The behavior domain as a qualitative causal relation graph

The actual behavior objects in the behavior domain are typically results of physical tests or simulations. Planning a test or creating a simulation model is a combined act of science and art. To be able to assist decision-making in product development projects with tight time constraints, a model should for example be optimized for the targeted problem, e.g., evaluate the eigenfrequencies below 10 Hz, or determine how much of the product life time that is consumed after 100 cycles of normal operation. The scientific part of the task is to identify and represent the physical phenomena that are relevant for the problem. The art is to create a representation that is as simple as possible, but not too simple. A reasonable level of detail will most likely show an "actual" behavior at a reasonable level of accuracy. Sometimes, a physical phenomenom is considered irrelevant for the targeted behavior and it is thus not included or monitored in the physical or virtual tests. A more thorough study would perhaps show unforeseen interplay between physical phenomena and even between physical components. It is important to remember that models and test pieces are cognitive tools with the purpose to help people understand and ultimately manage complexity. Sometimes, highly coupled phenomena and/or components in an implementation are decoupled on purpose and then coupled in a subsequent study, just to make us understand what is going on and why it is happening.

The preferred approach to create, verify, validate, decompose, and compare "actual" behavior is to have behavior attributes that are pointers to the context (e.g., the product model *as_is*) target problem (or more precisely the engineering question), and the stated model requirements.

2.3 Implementation domain

The ultimate design goal is achieved by conscious ordering and planning of components. A component is either physical or logical and the relations between the components are physical connections or logical links [14], or physical and logical interfaces, respectively. The implementation system structure specifies what components the design is composed of, what the attributes of the components are, and how they relate.

Any implementation system is composed of *components*. A component can be either a physical entity (e.g., a gearbox) or a logical entity (e.g., a numerical array). A basic component is single material body (e.g., a screw) that cannot be disassembled. Some components group together and form a sub-system or a module. For example, a keyboard is a module of a computer and a gearbox is a module of a drive-train in a vehicle. In a higher order system, a sub-system is a component of that system. A component has many *attributes* or properties. An attribute is characterized by an attribute variable (e.g., shape or color) and a value (e.g., cube or yellow), which may be continuous (e.g., a cube with variable dimensions) or discrete (e.g., a selectable set.of cubes). The *relations* between the components of an implementation structure can be physical or logical. A part-of relation between component and sub-system is a logical relation.

An implementation structure can be defined as static, dynamic, or hybrid. A *static structure* has components, attributes, and relations that are fixed in time. If the components, attributes, and/or their relations can be changed, the structure can be defined as *dynamic*. For example, a fuel injection system with valves than can open and close is a dynamic system.

Figure 5 shows a qualitative causal relation graph of the implementation domain and its relations to the other domains of the FBIE-model. *I1* is a direct implementation of function *f11* and an indirect implementation of function *f12* (via the intended behavior $b1_i$). Due to effects from the environment *e1*, the implemented structure i1 (decomposable into components and interfaces) has

an actual behavior $b11_a$ that has to be analyzed and checked against the intended behavior. The actual behavior shows some accidental behavior which require a direct correction of a design error in i1 and an unintended behavior or side-effect which require a new subfunction *f2*. This counter-measure is realized by a new intended behavior $b2_i$ and implemented in the new and more detailed structure *i2*.

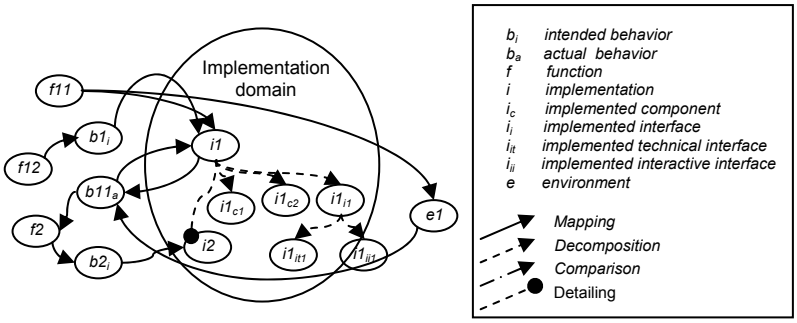

Figure 5. The implementation domain as a qualitative causal relation graph

The preferred approach to create, verify, validate, decompose, configure, and compare implementations is to treat an implementation as a composition of components and (technical and interactive) interfaces and to have implementation attributes that are pointers to the context (e.g., the list of customer requirements) and the set of required functions and/or intended behaviors.

2.4 Environment domain

Environment can be defined as an object imposed onto the implementation system (e.g., cooling water flowing in a car radiator), an environment around the system (e.g., the temperature and velocity of the cooling air stream around a car radiator), or an operation interacting with the product (e.g., an operator pushing a button that activates a function in the car interior climate system). This definition of environment is equivalent to the term *external effects* used in [14]. Further on, the three environment subcategories defined above are referred to as *functional environment*, *operational environment*, and *natural environment*, respectively.

Figure 6 shows a qualitative causal relation graph of the environment domain and its relations to the other domains of the FBIE-model. An environment *e1* is initially defined in the customer requirements and reformulated as a required function *f11*. The effect from the environment is defined by the interaction between *e1* and the implementation *i1* at a specific interface and assessed as an actual behavior $b11_a$.

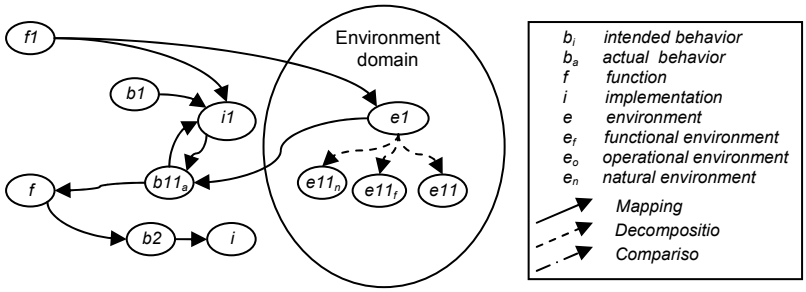

Figure 6. The environment domain as a qualitative causal relation graph

The preferred approach to create, verify, validate, decompose, configure, and compare active environments is to treat the environment as an entity that can be decomposed into sub-entities (functional environment, operational environment, and natural environment) and to have environment attributes that are pointers to the context (e.g., the list of customer requirements) and the set of defining functions, and the derived actual behaviors.

3 An example of FBIE-assisted design reasoning

Take the design of a new truck seat (see figure 7) as an example that can be used to demonstrate the FBIE-approach to management of behavior models in general and interface models in particular. The seat will be developed as an optional module in a heavy truck family and it will be used by a variety of drivers with different characteristic properties, such as length, weight, preferred sitting position, etc. Assume that there is a subtask to design the interface between the new truck seat and the truck floor platform.

Figure 7. Truck and driver can be viewed as truck set environment objects.

The perhaps most important requirement on a set-truck interface is that it must firmly fixed at the defined location. With the taxonomy defined by *the Little base set* this requirement may be formulated as the main function *support seat* (*f1*). Further more, *f1* can be decomposed into three sub-functions; *position seat* (*f11*), *secure seat* (*f12*), and *branch seat* (*f13*). *Position seat* is the required function to orient and align the seat at the target position (*u*). S*ecure seat* is the function to attach and hold the seat within a position interval ($\pm\delta u$). Finally, *branch seat* is the requirement to allow the seat to be disconnected and reconnected for maintenance, service, and upgrade reasons.

The main function (*f1*) may for example be solved, realized, or implemented with a *screw joint concept* (*i1*). The *intended (target) behavior* ($b1_i$) to accomplish *f1* is to keep *the seat-floor interface position within the target position* ($u\pm\delta u$) *for a set of external effects* (*e1*, *e2*) and within a geometric design space (*e3*). The *external effects*, or environment objects, *e1* and *e2* are defined by functions *f2* and *f3*, respectively. The function *f2* defines the required stiffness, or alternatively the allowed flexibility, of the interface, with respect to allowed movements and the frequencies of the complete system. Function *f3* is a safety related crash requirement. Environment *e3*, which is the boundary of the geometric envelope of the targeted truck family cabins, is defined by *f4*. The objects *e1*, *e2*, and *e3* are environment representations of *f2*, *f3*, and *f4*. An environment object has an interactive relation to one (or several) spatial feature(s) in the implementation system (e.g., a pressure acting on a surface).

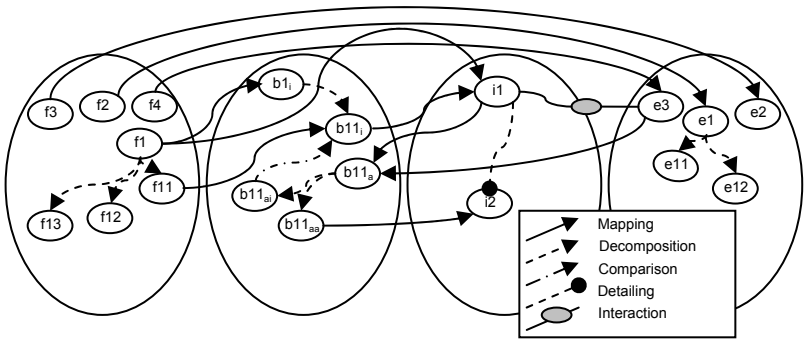

Figure 8. Causal paths (sep 1) between four domains (FBIE).

The main *engineering question* is how the implemented concept (*i1*) fulfills the technical function *f1* and environment functions *f2*, *f3*, and *f4* (i.e., what is the *real behavior* $b1_a$ of the implemented structure *i1*). For obvious reasons, it is beneficial to decompose this question into several subquestions, each with a more limited scope. Figure 8 shows how *b1* is decomposed into sub-behaviors that are related to the subfunctions *f11* and *f12*. Figure 8 indicates that the positioning problem related to *f11* is addressed first. At a first glance, the *position seat* function (*f11*) is realized by an actual behavior (*b11a*), which can be viewed is a direct behavior [14] that is given by the spatial dimension and orientation parameters, and it can be visualized and quantitatively obtained by performing CAD assembly orientations, such as mate and/or align) between pairs of mating surfaces on the implementing system. In our case, the correct position (i.e., $b11_i$) is probably not only that the mating features of the seat and the truck floor fit together (i.e., a direct behavior) but also a spatial relation between the seat and other features of the environment, such as door, steering wheel, and pedals (i.e., an indirect behavior), This justifies that the interaction between *i1* and *e3* is also considered when the actual positioning of the seat ($b11_a$) is evaluated. This actual (nominal) position ($b11_a$) must be compared to the intended nominal position. If we assume that we have detected a difference between the target position $b11_i$ and the actual position bll_a, this difference can be interpreted as an accidental behavior $b11_{aa}$, which can be removed by creating a modified implementation *i2*. In our example, *I1* is a representation of nominal geometry. In more advanced situations, the effect of geometric tolerances (a range) is also a significant issue. In that case, $b1_a$ is a range and $b1_i$ must also define a target position and the acceptable deviation. This can either be addressed by extending *f11* with a position range or by formulating the range as a subfunction to *f11* (e.g., *f111*)

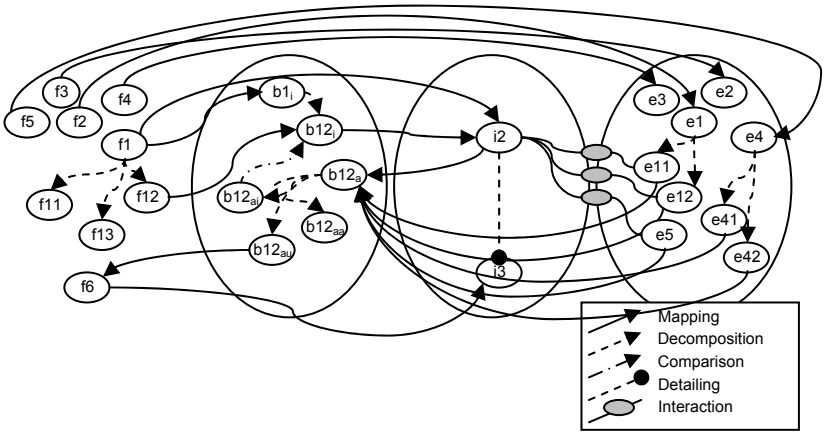

Figure 9. Causal paths (step 2) between four domains (FBIE).

At this stage, which is shown in figure 9), a realistic *engineering question* is how the new implemented concept (*i2*) fulfills the technical subfunction *f12* and the environment function *f2* (i.e., what is the *real behavior* $b12_a$ of the implemented structure). The *secure seat* function (*f12*) is related to the stiffness and strength of the interface. The actual behavior is determined by the interaction between *i2* and e1, which can be decomposed into a *static gravity field in the vertical direction* (*e11*) *and transient acceleration field* (*e12*) *in the longitudinal direction*. The environment object *e11* interacts with *i2* in the entire material volume of *i2*, i.e. at a *body interface*, which is a relation between the implementation and an external object such as the environment or the user/operator. The actual stiffness of the truck floor (*e41*) and the seat (*e42*), the weight of the driver (e5), and the prescribed static gravity (e11) and transient acceleration field (*e12*) cause an actual behavior ($b12_a$) of the screw joint (*i2*). The perhaps most appropriate method to estimate the actual behavior, is to scrutinize the motions, forces, and stress results from a simulation with an FE-model. Thermoelastic effects caused by microslip phenomena between the interacting surfaces in the interface may cause excessive wear which is an unintended behavior or side-effect ($b12_{au}$). A side-effect may be decided to be acceptable and ignored or alternatively it can be decided that it must be removed or reduced: This can, for example, be accomplished by introducing a new function, which is represented by *f6* in figure 9 and implemented in *i3*.Too high stresses in the screws or too high forces transferred to the floor may be classified as an accidental behavior ($b12_{aa}$). An accidental behavior may be regarded as a design error or fault and it requires a modification of *i2* or a new implementation (*i3*), and so on until the design is released for production

4 The behavior models in the FBIE approach

The relations between the behavior domain and the other domains in the FBEI approach are described in chapter 2. It is important to observe that the behavior objects in the FBIE-graphs shown above are explicit representations of intended and actual behavior. In model-based product development, the actual behavior is typically transformed results from a simulation experiment performed with a computer model or results from a physical test.

Question-driven modeling (QDM) [21] is a structured sequence of activities for situated model-based product development. The purpose of QDM is to enable flexible knowledge acquisition, generation, and reuse in model-based design. Engineering questions are frequently related to a state of knowledge deficit. In QDM, a knowledge deficit is defined as a problem (required knowledge ≠ available knowledge) and formulated as a question, which is addressed to a model. Problem solving can thus be represented as a situated question-driven process. The model which is required to address the question must consequently be targetted/optimized for its specific purpose.

A QDM scenario [21] is modeled as a recursive workflow (see the left-hand side of figure 33) with six distinctive steps or scenes:

1. Define the context-dependent engineering problem and reformulate it as one or several question(s). The context may for example be a stored product model of an artifact.
2. Analyze each question and specify the requirements for a target model.
3. Synthesize (i.e., configure) a specific systems model that available knowledge suggests will satisfy the requirements.

4. Perform the simulation, or alternatively a set of simulations in probabilistic design, and store the result as an aggregated information object.
5. Analyze the simulation result to verify and validate (V&V) the model. Store as information object.
6. Synthesize an answer from the analyzed simulation results (and identify new questions).

Each activity in a question-driven modeling scenario needs input information, creates output information, is controlled by information, and is performed by an engineer with the aid of a mechanism or tool. The right-hand side of figure 10 shows an IDEF0- or SADT-based representation of the six steps, or scenes, in a basic scenario workflow. Each of the scenario objects (i.e., the question, model requirements, model simulation result, and answer) has an owner attribute, a state attribute, and a causality relation to the object that was created in the immediately preceding activity in the scenario workflow.

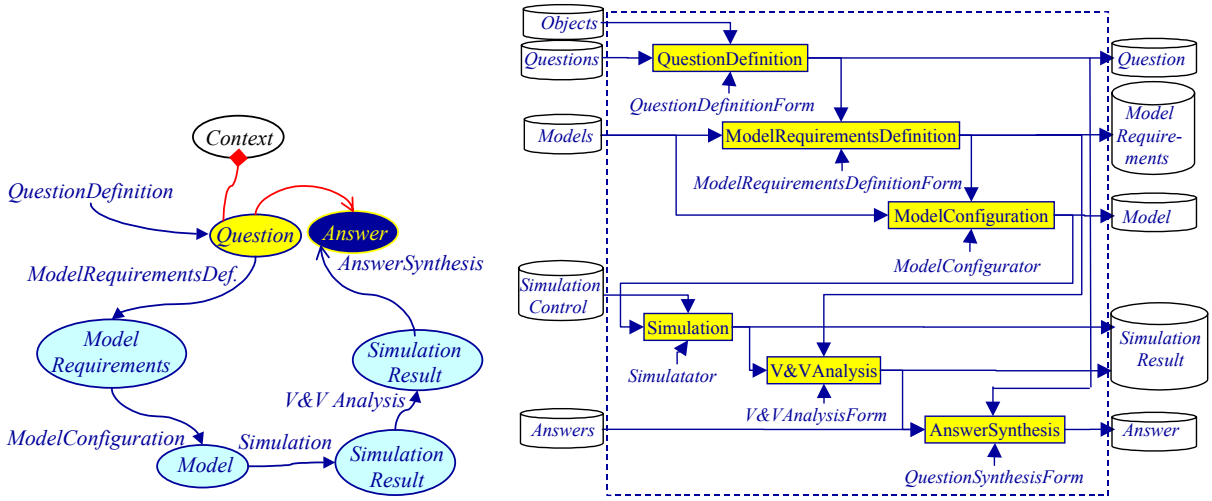

Figure 10. Question-driven modeling (left) and an activity view of a scenario loop (right), from [21]

The mechanisms in figure 10 are currently web-based forms, (e.g., one form for question definition and another for model requirements definition), specialized configuration tools, and industry-standard simulator tools (e.g., Ansys for finite element simulation and Adams for multibody dynamic simulations).

Figure 11 shows the casual paths in QDM adapted to the FBIE-structure. A question of how the actual implementation will behave is formulated. This question has a context relation to the intended behavior. Then the question is analyzed the requirements for the target behavior model is specified. Based on the model requirements, a systems model that available knowledge suggest will satisfy the requirements is synthesized (i.e., configured). The simulation, or alternatively the set of simulations in probabilistic design, is performed and the native results are stored. The results are then analyzed and transformed to the targeted actual behavior and possibly further decomposed into the as intended, unintended and accidental components. The actual behavior is then used to formulate an answer to the original question.

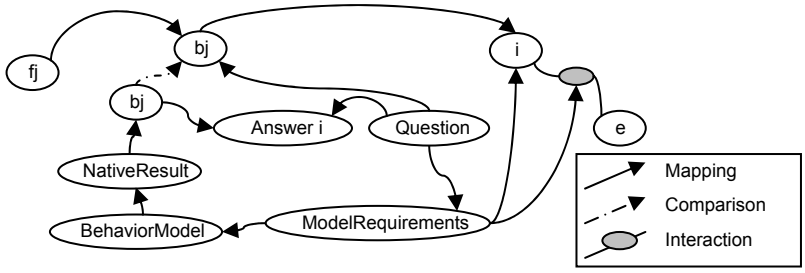

Figure 11. Causal paths between the FBIE domains and the behavior models required to assess the actual behavior of an implementation.

5 Conclusions

A complex system is a system with many components and interconnections, interactions, or interdependencies that are difficult to describe, understand, predict, manage, design, or change. Engineering design and redesign of complex technical systems involves managing of complexity and reduction of uncertainty and it, consequently depends much more heavily on computer simulation than design of smaller and less complex products. The main objective for simulations is to help reduce uncertainty by enabling identification of unintended and accidental interactions between components and to assess and verify various features of the system in early design stages.

This paper examines design reasoning and proposes a situated design knowledge management framework, based on the Function-Behavior-Implementation-Environment model, with the aim of supporting designing and redesigning and engineering of complex technical systems.

The framework integrates:

- A general modular modeling principle for technical systems [4].
- The concept of situatedness [5].
- The Function-Behavior-Structure (FBS) [16].
- The question-driven modeling (QDM) workflow model [21].

The presented framework, thus, extends the FBS-approach with the active environment including the human user/operator and includes the reasoning process by adding casual paths between the QDM reasoning and modeling process with the FBIE-domain objects.

The structure of the proposed framework has been conceptually described by elaborating on a simple but not oversimplified industrial case.

7 Acknowledgements

The work presented here was performed in the Interface project and financially supported by the Swedish Strategic Research Foundation.

References

[1] ESD, "ESD symposium committee overview: engineering systems research and practice", Engineering Systems Division MIT, http://esd.mit.edu/ESD_Internal_Symposium_Docs/WPS/ESD-WP-2003-01.20-ESD_Internal Symposium.pdf, May 2002.

[2] Aristotele, "Posterior analytics", Translated by G.R.G. Mure. http://www.philosophy.ru/library/aristotle/post_an/00.html, 350BC.

[3] Eris, O., "Asking generative design questions: a fundamental cognitive mechanism in design thinking", Proc. International Conference on Engineering Design ICED 03, August 19-21, Stockholm, Sweden, 2003.

[4] Sellgren, U., "Architecting models of technical systems for non-routine simulations," International Conference on Engineering Design ICED 03, Stockholm, 2003.

[5] Gero, J. , "Situated design computing: introduction and implications", Proc. of the 8th International Conference DESIGN 2004, pp. 27-34, Dubrovnik, Croatia, 2004..

[6] DOE, "Systems engineering and interface management", Rev E, June, 2003, http://oecm.energy.gov/Portals/2/SystemsEngineeringInterfaceMgmt.pdf, Program and Project Management, U.S. Department of Energy, 2003.

[7] Koopman, Jr, P.H., "A taxonomy of decomposition strategies based on structures, behaviors, and goals", Design Theory & Methodology Conference, September, 1995.

[8] Pahl, G. and Beitz, W., "Engineering design a systematic approach", 2nd edition, Springer, 1996.

[9] Kitamura, Y, Sano, T., Namba, K., and Mizoguchi, R., "A functional concept ontology and its application to automatic identification of functional structures", Advance Engineering Informatics, Vol. 16, pp. 145-163, Elsevier Science Ltd., 2002.

[10] Magee, C.L. and de Weck, O.L., "Complex system classification", Fourteenth Annual International Symposium of the International Council On Systems Engineering (INCOSE), June 20-24, 2004.

[11] Hubka, V. and Eder, W.E., "Theory of technical systems", Springer-Verlag, Berlin, 1988.

[12] Van Wyk, R.J., "A standard framework for product protocols", in Khalil (ed.), Management of Technology, Geneva, pp.93-99, 1988.

[13] Gero, J.S., "Design prototypes: a knowledge representation scheme for design", AI Magazine, Vol 11 (4), pp. 26-36, 1990.

[14] Qian, L. and Gero, J.S., "Function-behavior-structure paths and their role in analogy-based design", Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 10, pp. 289-312, 1996.

[15] Prabhakar, S and Goel, A.K., "Functional modeling for enabling adaptive design of devices for new environments", Artificial Intelligence in Engineering, Vol. 12, pp. 417-444, Elsevier Science Limited, 1998.

[16] Gero, J.S. and Kannengiesser, U, "The situated function-behaviour-structure framework", in Gero, J.S (ed.), Artificial Intelligence in Design'02, pp. 89-104, Kluwer Academic Publishers, Dordrecht, the Netherlands, 2002.

[17] Suh, N.P., "The principles of design", Oxford University Press, Inc., New York, USA, 1990.

[18] Little, A., Wood, K., and McAdams, D., "Functional analysis: a fundamental empirical study for reverse engineering, benchmarking and redesign", Proc. of the 1997 Design Engineering Technical Conference 97-DETC/DTM-3879, Sacramento, CA, USA, 1997.

[19] Stone, R., "Towards a theory of modular design", Doc. Thesis, The University of Texas at Austin, Texas, USA, 1997..

[20] Whitney, D.E., "Why mechanical design cannot be like VLSI design", Research in Engineering Design, Vol. 8, 1996, pp.125-128, 1996.

[21] Sellgren, U., "Question-driven modeling", Proc. of the 8th International Conference DESIGN 2004, pp.503-510, Dubrovnik, Croatia, May 18-21, 2004.

Ulf Sellgren
Royal Institute of Technology
KTH - Machine Design
SE-100 44 Stockholm
Sweden
Phone: +46 – 8 790 73 87
Fax: +46 – 8 723 17 30
E-mail: ulfs@md.kth.se