# Elements of Systems Engineering

Jeremy Ziegler

First Edition, 2012

ISBN 978-81-323-3046-2

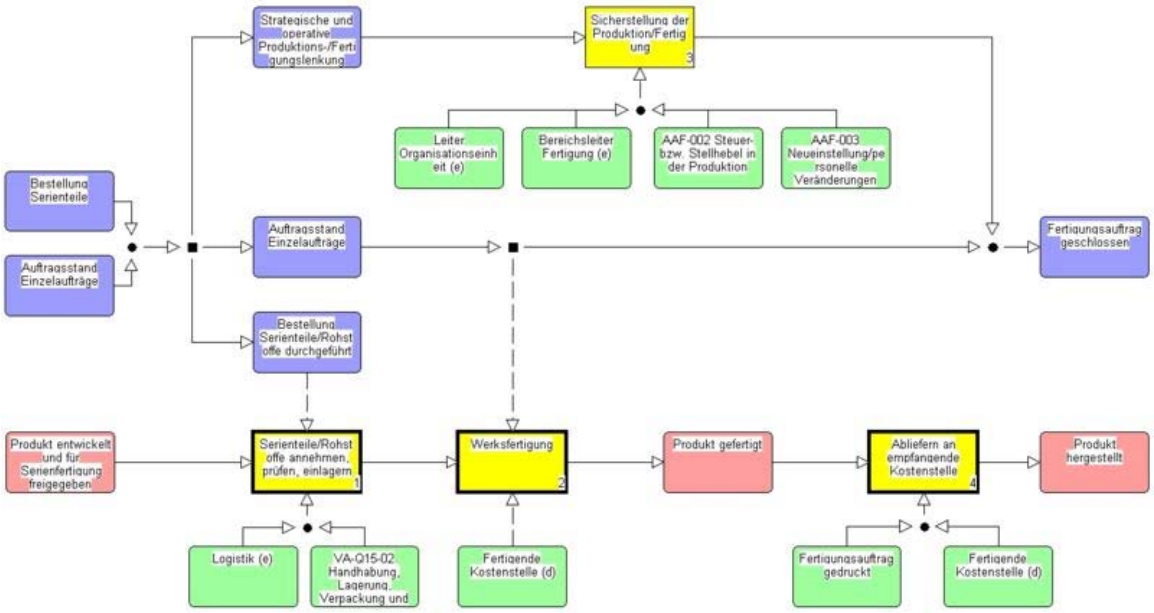


*Published by:*
**Research World**
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

# Table of Contents

**Chapter 1**

# Integrated Enterprise Modeling



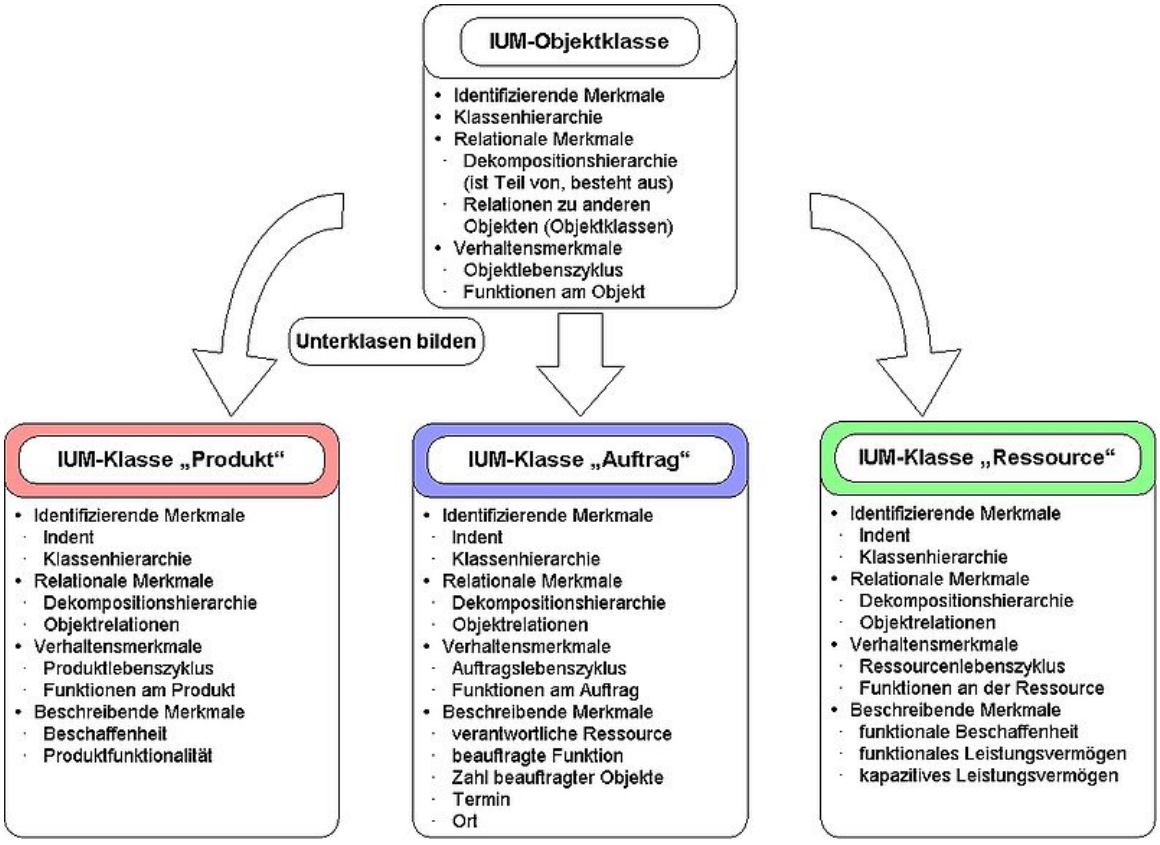

Model example

The **Integrated Enterprise Modeling** (IEM) is an enterprise modeling method used for the admission and for the reengineering of processes both in producing enterprises and in the public area and service providers. In the Integrated Enterprise Modeling different aspects as functions and data become described in one model. Furthermore the method supports analyses of business processes independently of the available organizational structure.

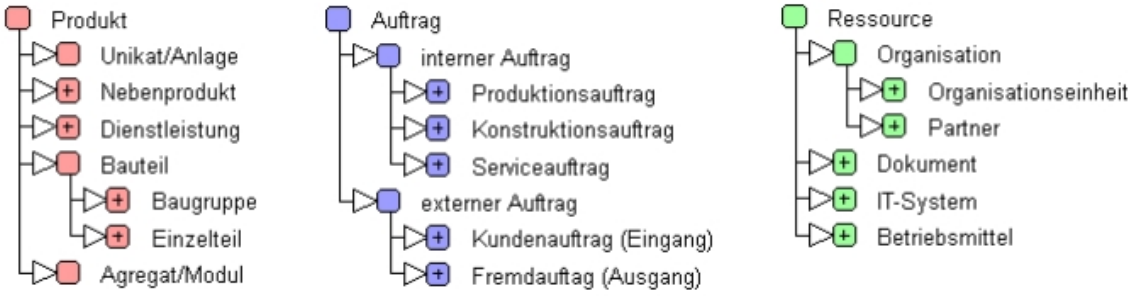
The Integrated Enterprise Modeling is developed at the *Fraunhofer Institute for Production Systems and Design Technology (German: IPK) Berlin, Germany*.

## *Integrated Enterprise Modeling Topics*

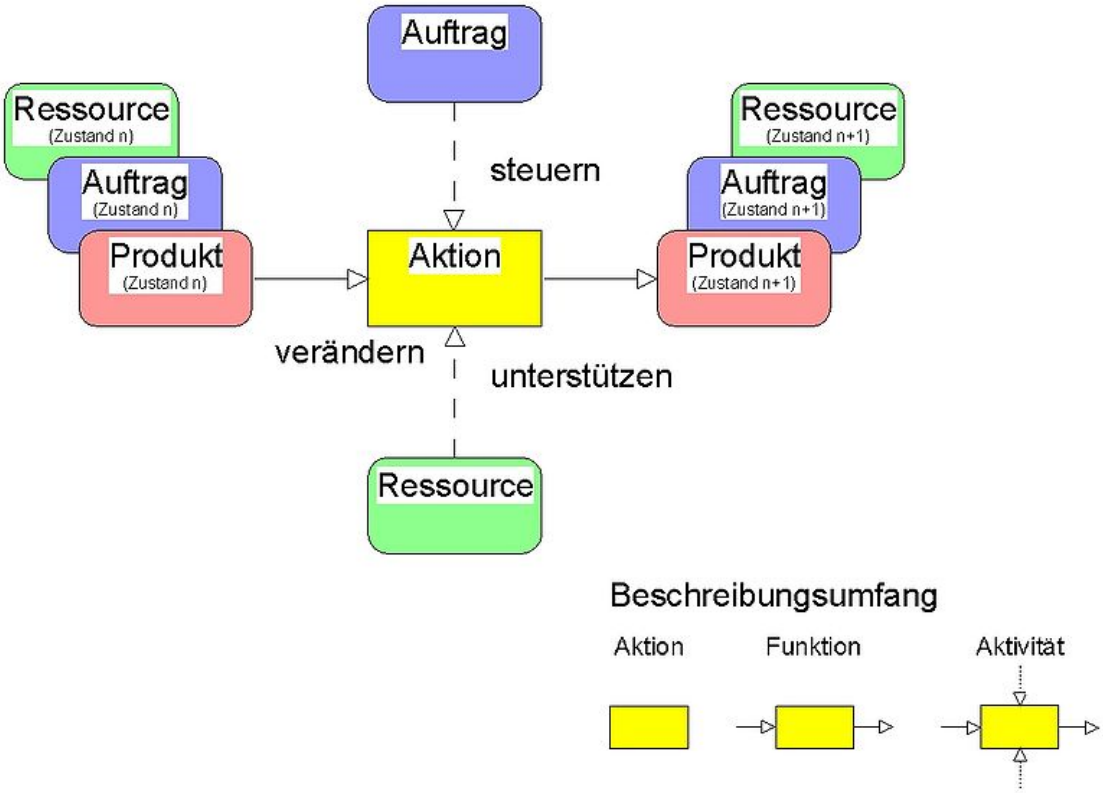
### Base constructs




Generic object classes of the IEM




Overview of the object classes product, order and resource




Generic activity model

The Integrated Enterprise Modeling (IEM) method uses an object-oriented approach and adapts this for the enterprise description. An application-oriented division of all elements of an enterprise forms the core of the method in generic object classes *"product"*, *"resource"* and *"order"*.

Product

The object class "product" represents all objects whose production and sale are the aim of the looked-at-enterprise as well as all objects which flow into the end product. Raw materials, intermediate products, components and end products as well as services and the describing data are included.

Order

The object class "order" describes all types of a commissioning in the enterprise. The objects of the class "order" represent the information which is relevant from the point of view of planning, control and supervision of the enterprise processes. One understands by it what, when, at which objects, in whose responsibility and with which resources it will be executed.

Resource

The IEM class "resource" contains all necessary key players which are required in the enterprise for the execution or support of activities. Among other things these are employees, business partner, all kinds of documents as well as information systems or operating supplies.

The classes "product", "order", and "resource" can gradually be given full particulars and specified. Through this it is possible to show both line of business typical and enterprise specific product, order and resource subclasses. Structures (e.g. parts lists or organisation charts) can be shown as relational features of the classes with the help of being-part-of- and consists-of-relations between different subclasses.

Action

The activities which are necessary for the production of products and to the provision of services can be described as follows: an activity is the purposeful change of objects. The aim orientation of the activities causes an explicit or implicit planning and control. The execution of the activities is incumbent by the capable key players. From these considerations the definitions can be derived for the following constructs:

- An *action* is an object neutral description of activities: a verbal description of a work task, a lawsuit or proceeding;
- A *function* describes the change of state of a defined status into another defined one of objects of a class by using an action;
- An *activity* specifies necessary resources for the state transformation of objects of a class the controlling order described by a function and these for the execution of this transformation in the enterprise, in each case represented by an object state description.

## Views

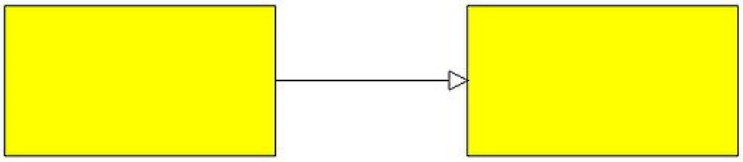
All modeled data of the looked-at-enterprise are recorded in the model core of an Integrated Enterprise Modeling (IEM) model in two main views:

- the "information model" and
- the "business process model".

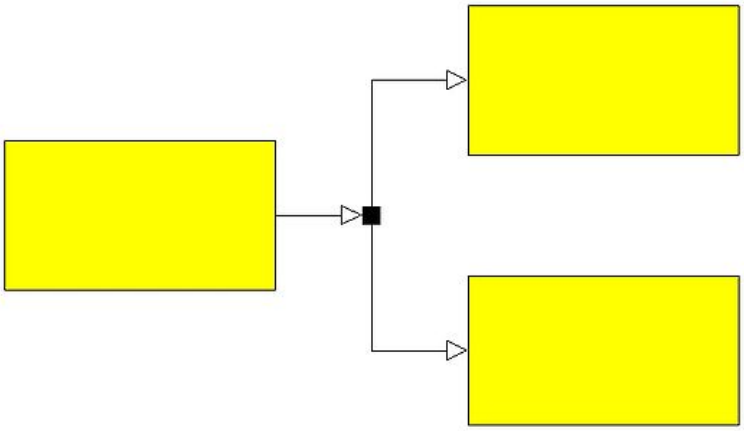
All relevant objects of an enterprise, their qualities and relations are shown in the "information model". It is class trees of the object classes "product", "order" and "resource" here. The "business process model" represents enterprise processes and their relations to each other. Activities are shown in their interaction with the objects.

## Process modeling

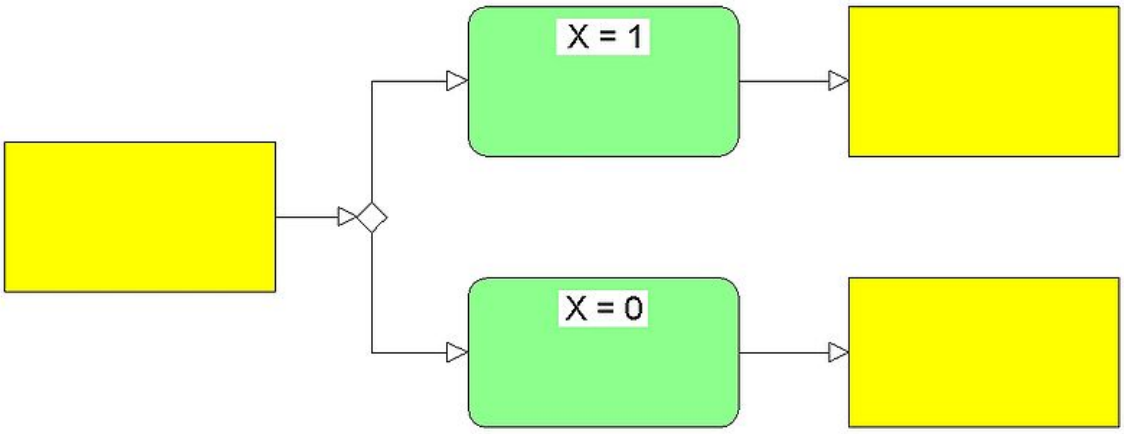
The structuring of the enterprise processes in Integrated Enterprise Modeling (IEM) is reached by its hierarchical subdivision with the help of the decomposition. Decomposition means the reduction of a system in a partial system which respectively contains components which are in a logical cohesion. The process modeling is a partitioning of processes into its threads. Every thread describes a task completed into itself. The decomposition of single processes can be carried out long enough until the threads are manageable, i.e. appropriately small. They may turn out also not too rudimentary because a high number of detailed processes increases the complexity of a business process model. A process modeling person therefore have to find a balance between the effort complexity degree of the model and possible detailed description of the enterprise processes. A model depth generally recommends itself with at most three to four decomposition levels (model levels).



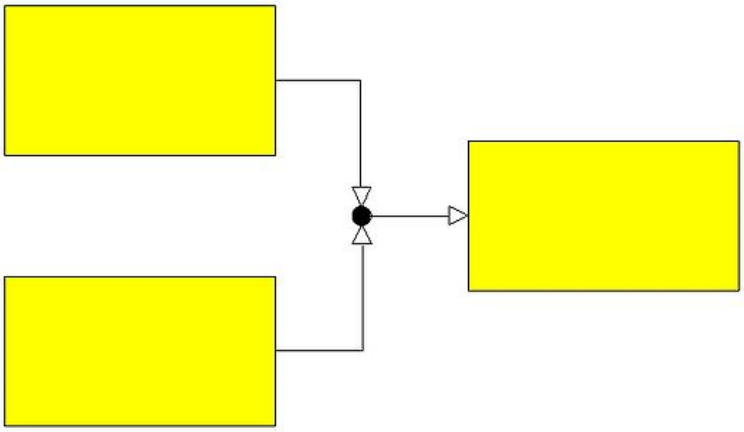
Sequential order



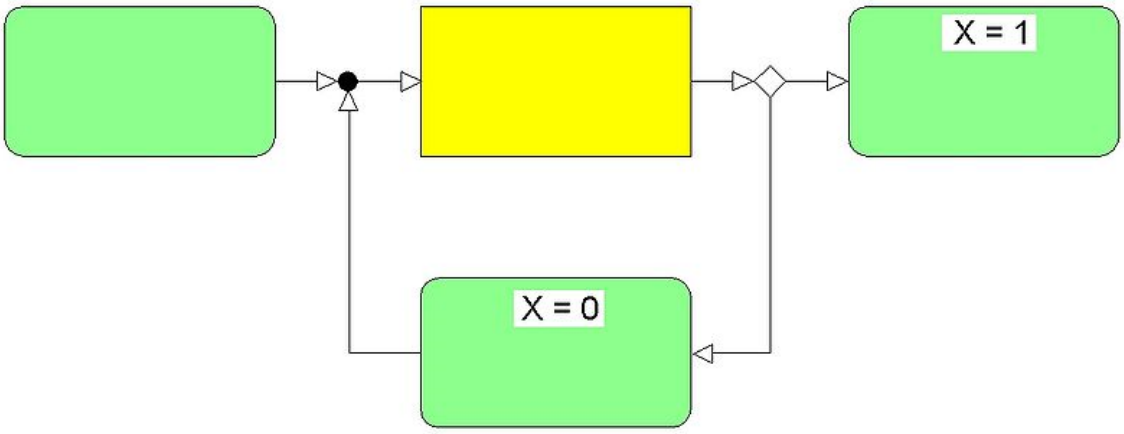
Parallel branching




Case distinction



Uniting




Loop

On a model level business process flows are represented with the aid of illustrated combination elements. There are these five basic types of combinations between the activities:

- Sequential orde: At a sequential order the activities are executed after each other.
- Parallel branching: A parallel branching means that all parallel branched activities to be executed have to be completed before the following activity can be started with. It is not necessary that the parallel activities are executed at the same time. They can be deferredly carried out, too.
- Case distinction: Decision either or. The case distinction is a branching in alternative processes depending on definition of the subsequent conditions.
- Uniting: The end of a parallel as the case may be alternative execution or also an integration of process chains is indicated by the uniting.
- Loop: A repatriation (loop, cycle) is represented by means of case distinction and uniting. The activities included in the loop are executed as long as the condition for the continuation is given.

## Modeling proceeding

The modeling procedure for the illustration of business processes in IEM covers the following steps:

- System delimitation,
- Modeling,
- Model evaluation and use,
- Model change.

The *system delimitation* is the base of an efficient modeling. Starting out from a conceptual formulation the area of the real system to be shown is selected and interfaces will be defined to an environment. In addition, the detail depth of the model is also

determined, i.e. the depth of the hierarchical decomposition relations in the view "business process model".

The delimited real system is convicted with help of the IEM method in an abstract model. IEM is the construction of the two main positions "information model" and "business process model". The "information model" is made by the specification of the object classes to be modeled for "product", "order" and "resource" with the class structures as well as descriptive and relational features. By identification and description of functions, activities and its combination to processes the "business process model" is formed. As a general rule the construction of the "information model" follows first in which the modeling person can go back to available reference class structures. The reference classes which do not correspond to the real system or were not found to be relevant at the system delimitation are deleted. The missing relevant classes are inserted. After the object base is fixed, the activities and functions are joined together at the objects according to the "generic activity model" and with the help of combination elements to business processes. A model is made which can be analysed and changed if it's required. It often happens, that during the construction of the "business process model" new relevant object classes are identified so that the class trees getting completed. The construction of the two positions is therefore an iterative process.

Afterwards weak points and improvement potentials can be identified in the course of the *model evaluation*. This can cause the *model changes* whose realization should clear the weak points and make use of the improvement potentials in the real system.

## Modeling tool MO²GO

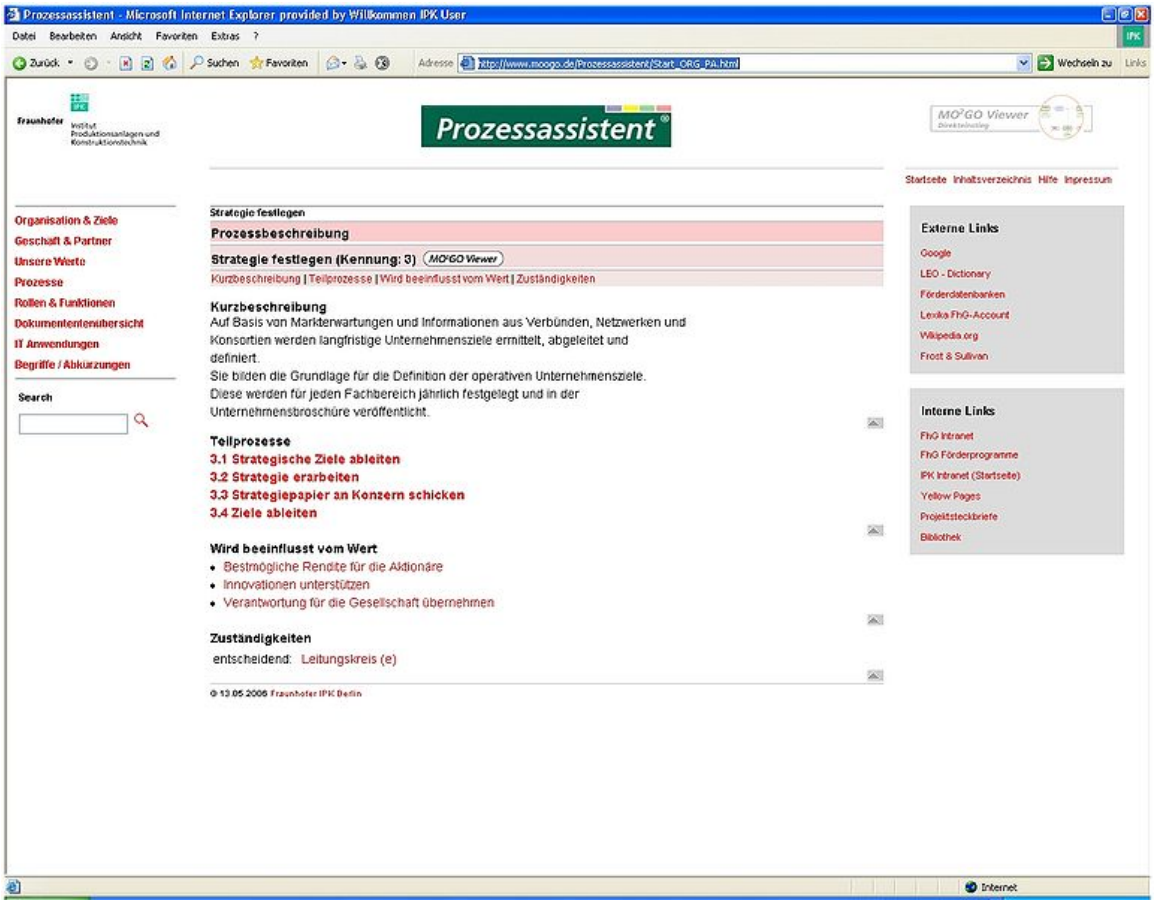
The software tool MO²GO (method for an object-oriented business process optimization) supports the modeling process based on the integrated enterprise modeling (IEM). Different analyses of a given model are available like the planning and implementation of information systems. The MO²GO system is expandable easily and makes a high-speed modeling approach possible.

The currently used MO²GO system consists of the following components:

- MO²GO version 2.4: This component offers modeling functions for class structures, process chains and mechanism for analysis of IEM.
- MO²GO Macro editor version 2.1 : The macro editor supports the outline of MO²GO macros for user-defined evaluation procedures.
- MO²GO Viewer version 1.07 : The Java based and licence free MO²GO Viewer is a user interface to be used easily to navigate process chains through MO²GO.
- MO²GO XML converter version 1.0 : Nowadays the IT implementation works mainly with UML diagrams. MO²GO supports a component for a model based XML file which can be imported in UML tools.
- MO²GO Web publisher version 2.0 : The web Publisher is a mechanism of analysis to be started directly out of MO²GO 2.4. A process assistant is the result of the evaluation of the model contents based on texture and hyperlink

representation. To be able to adapt the process assistant to the user requirements flexibly, the web Publisher contains a configuration component.

## MO²GO Process assistant




Prozess assistant

The IEM business process models contain much information which can not only be used by system analysts but also be helpful for the employees at their daily work. To provide this model information for the staff and to enable the participation of the employees for the results of the modeling, a special tool was developed at the Fraunhofer IPK. This is a web based process assistant whose contents are generated automatically from the IEM business process model of the enterprise. The process assistant provides all users the information of the business process model in a HTML based form by intranet of the enterprise. For its implementation no special methods or tool knowledge is required besides the basic EDP and Internet experiences.

The process assistant has been developed so that the employees can find answers to the questions fast and precisely: e.g.

- What are the processes in the enterprise?
- In which way are they structured as?

- Who and with which responsibility is involved in the certain process?
- Which documents and application systems are used?

Or also:

- A certain organisation unit is involved at which processes?
- Or in which processes a certain document or an application system is used?

To make an informative process assistant from the business process model, certain modeling rules must be followed. The means e.g. that the individual actions must be deposited with its descriptions, the responsibility of the organisation units must be indicated explicitly or the paths also must be entered to the documents in the class tree. The fulfilment of these conditions means an additional time expenditure at the modeling, if these conditions are met, all employees are able to *"surf"* online through the intranet with the help of the process assistant by an informative enterprise documentation. They have the possibility between a graphic view and a texture based description according to their preferences and methodical previous knowledge. The graphic view is provided by the MO²GO Viewer, a viewer tool for MO²GO models. The process assistant and the MO²GO Viewer are connected so that the graphic representation of the process looked at can be accessed context sensitively from the process assistant.

Users can call on all templates, specifications and documents for the working sequence both from the process assistant and from the MO²GO Viewer online. Therefore the process assistant cannot only be employed for the tracing of the modeling results but also in the daily business for the training of new employees as well as execution of process steps. To improve the usability in the daily routine, the process assistant can be adapted to the needs of the users flexibility. This customization can be carried out both concerning the layout and concerning the main content emphases of the process assistant.

## *Areas of application of the IEM*

Knowledge is used in organisations as a resource to render services for customers. The service preparation performs along actions which are described as processes or business processes. The analysis and improvement in dealing with knowledge presupposes a common idea about this context. An explicit description of the processes therefore is required because they represent the context for the respective knowledge contents. The process modeling represents a powerful instrument for the design and a conversion of a process-oriented knowledge management. In the context of the method of the business process-oriented knowledge management (GPO KM) developed at the Fraunhofer IPK the method of the "integrated enterprise modeling" (IEM) is accessed. It makes it possible to be able to show, to describe, to analyse and to form organisational processes. The IEM features few object classes, is ascertainable easily understandable and fast. Furthermore the object orientation of the IEM opens up the possibility of showing knowledge as an object class. For the knowledge-oriented modeling of the business processes according to the IEM method the relevant knowledge contents have to be specified after knowledge

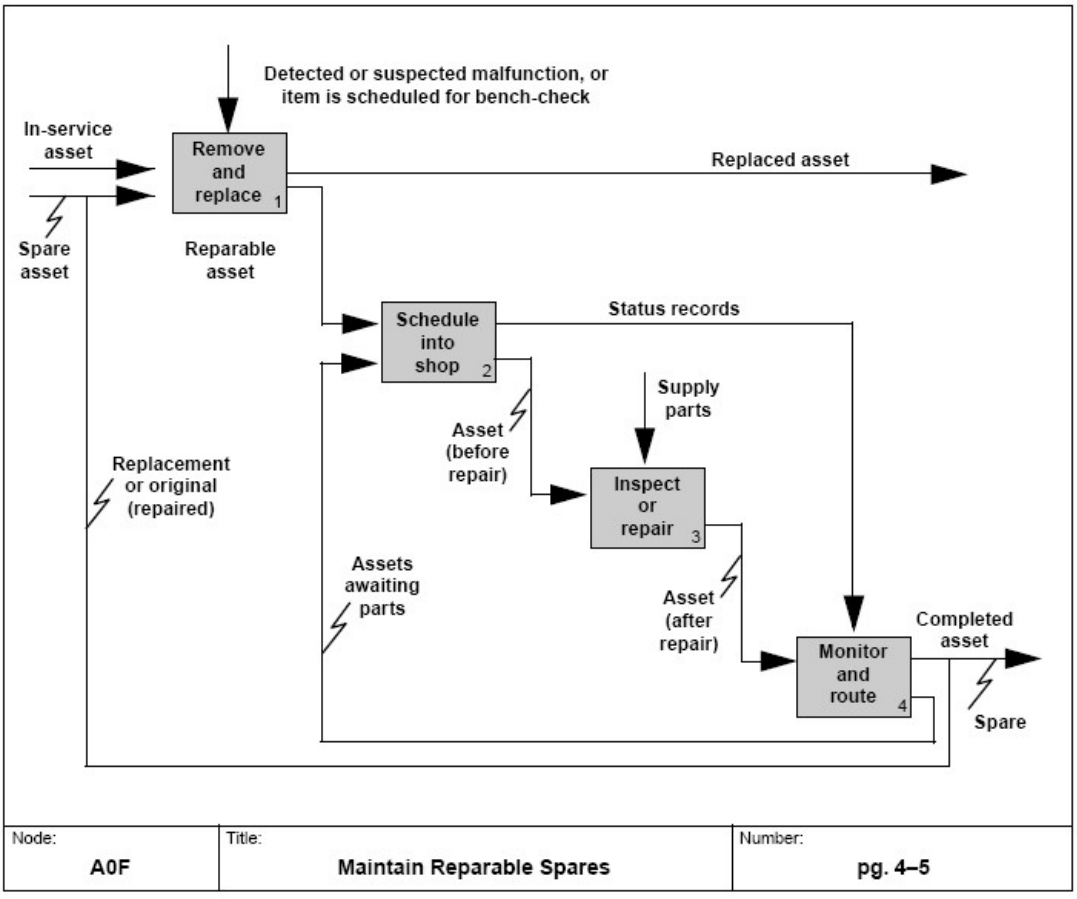
domains and know-how bearers and represented as resources in the business process model.

In further applications IEM is used to create models across organisations (e.g. companies)to archive a common understanding between the involved stakeholders and derive services (create software and define the ASP). In this context the object oriented basis of IEM has been used to create a common semantic across the single company models and to archive compliant enterprise models (predefined classes - terminology, model templates, etc.). The reason is that the terminology used within a model has to be understandable independent of the modeling language.

**Chapter 2**

# Function Model

A **function model** or *functional model* in systems engineering and software engineering is a structured representation of the functions (activities, actions, processes, operations) within the modeled system or subject area.




Example of a function model of the process of "Maintain Reparable Spares" in IDEF0 notation.

A function model, also called an activity model or process model, is a graphical representation of an enterprise's function within a defined scope. The purposes of the function model are to describe the functions and processes, assist with discovery of information needs, help identify opportunities, and establish a basis for determining product and service costs.

## *History*

The function model originates in the 1950s, after in the first half of the 20th century other types of management diagrams had already been developed. The first known Gantt Chart was developed in 1896 by Karol Adamiecki, who called it a *harmonogram*. Because Adamiecki did not publish his chart until 1931 - and in any case his works were published in either Polish or Russian, languages not popular in the West - the chart now bears the name of Henry Gantt (1861–1919), who designed his chart around the years 1910-1915 and popularized it in the West. The first structured method for documenting process flow, the flow process chart, was introduced by Frank Gilbreth to members of American Society of Mechanical Engineers (ASMR) in 1921 as the presentation “Process Charts—First Steps in Finding the One Best Way”. Gilbreth's tools quickly found their way into industrial engineering curricula.

One of the first well defined function models, was the Functional Flow Block Diagram (FFBD) developed by the defense-related TRW Incorporated in the 1950s. In the 1960s it was exploited by the NASA to visualize the time sequence of events in a space systems and flight missions. It is further widely used in classical systems engineering to show the order of execution of system functions.

## *Functional modeling topics*

### Functional perspective

In systems engineering and software engineering a function model is created with a functional modeling perspective. The functional perspective is one of the perspectives possible in business process modelling, other perspecifives are for example behavioural, organisational or informational.

A functional modeling perspective concentrates on describing the dynamic process. The main concept in this modeling perspective is the process, this could be a function, transformation, activity, action, task etc. A well-known example of a modeling language employing this perspective is data flow diagrams.

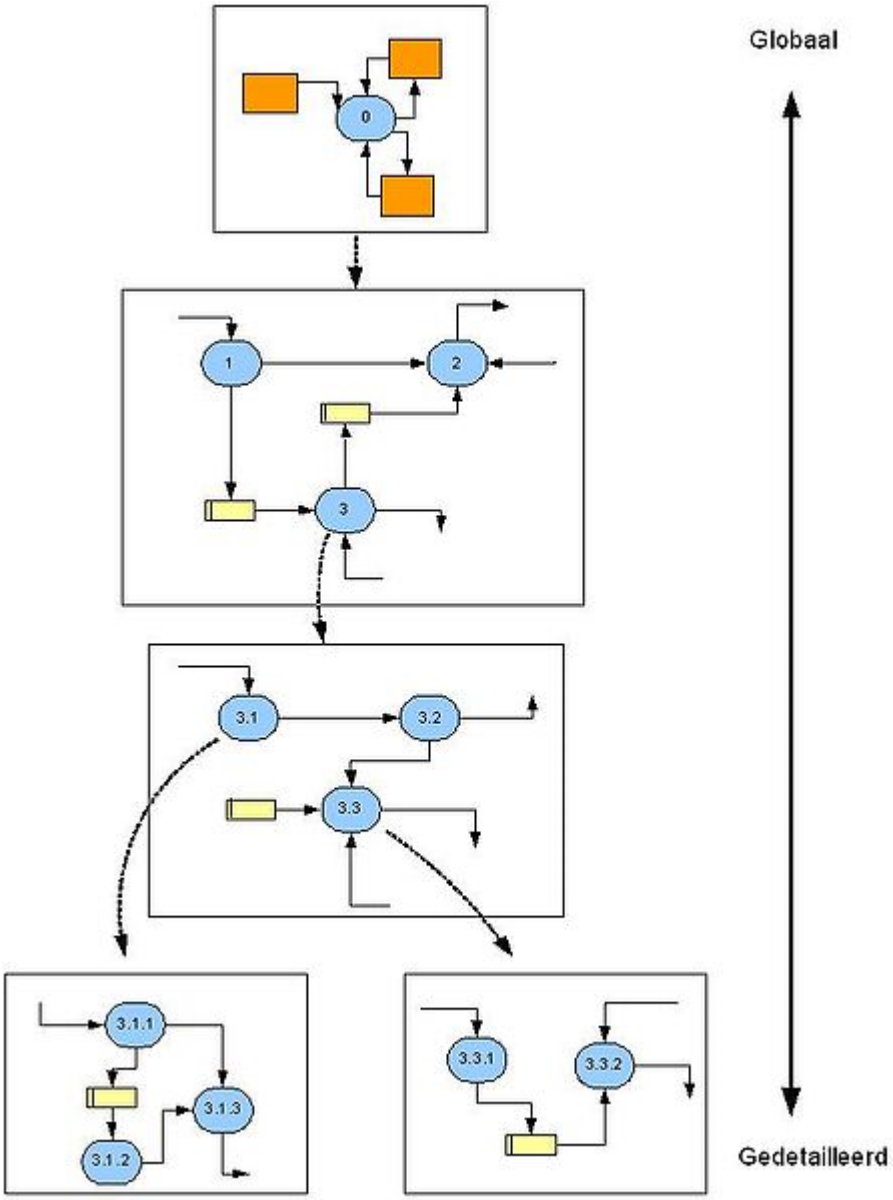
The perspective uses four symbols to describe a process, these being:

- Process: Illustrates transformation from input to output.
- Store: Data-collection or some sort of material.
- Flow: Movement of data or material in the process.
- External Entity: External to the modeled system, but interacts with it.

Now, with these symbols, a process can be represented as a network of these symbols. This decomposed process is a DFD, data flow diagram.

In Dynamic Enterprise Modeling a division is made in the Control model, Function Model, Process model and Organizational model.

## Functional decomposition




Example of functional decomposition in a systems analysis.

Functional decomposition refers broadly to the process of resolving a functional relationship into its constituent parts in such a way that the original function can be reconstructed from those parts by function composition. In general, this process of decomposition is undertaken either for the purpose of gaining insight into the identity of

the constituent components, or for the purpose of obtaining a compressed representation of the global function, a task which is feasible only when the constituent processes possess a certain level of *modularity*.

Functional decomposition has a prominent role in computer programming, where a major goal is to *modularize* processes to the greatest extent possible. For example, a library management system may be broken up into an inventory module, a patron information module, and a fee assessment module. In the early decades of computer programming, this was manifested as the "art of subroutining," as it was called by some prominent practitioners.

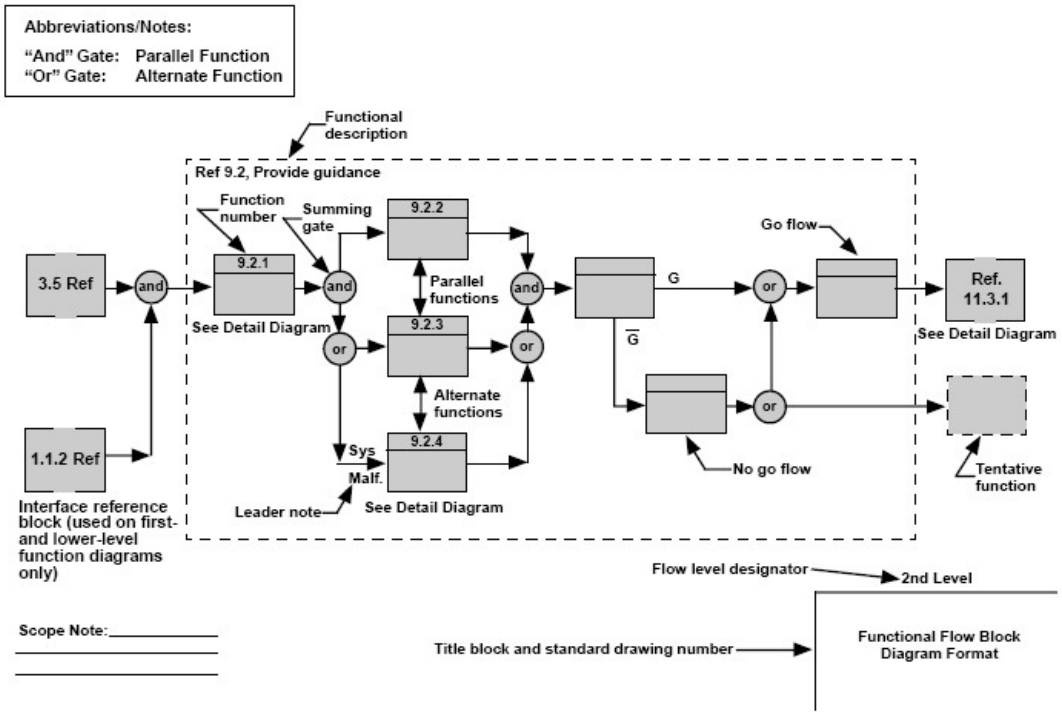
Functional decomposition of engineering systems is a method for analyzing engineered systems. The basic idea is to try to divide a system in such a way that each block of the block diagram can be described without an "and" or "or" in the description.

This exercise forces each part of the system to have a pure function. When a system is composed of pure functions, they can be reused, or replaced. A usual side effect is that the interfaces between blocks become simple and generic. Since the interfaces usually become simple, it is easier to replace a pure function with a related, similar function.

## *Functional modeling methods*

The functional approach is extended in multiple diagrammic techniques and modeling notations.

## Functional Flow Block Diagram

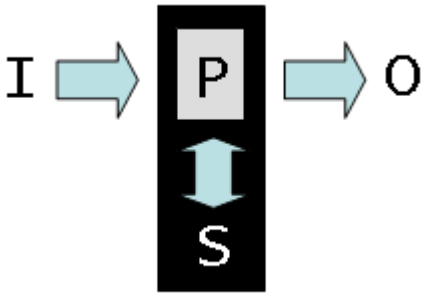



Functional Flow Block Diagram Format.

The Functional flow block diagram (FFBD) is a multi-tier, time-sequenced, step-by-step flow diagram of the system's functional flow. The diagram is developed in the 1950s and widely used in classical systems engineering. The Functional Flow Block Diagram is also referred to as *Functional Flow Diagram*, *functional block diagram*, and *functional flow*.

Functional Flow Block Diagrams (FFBD) usually define the detailed, step-by-step operational and support sequences for systems, but they are also used effectively to define processes in developing and producing systems. The software development processes also use FFBDs extensively. In the system context, the functional flow steps may include combinations of hardware, software, personnel, facilities, and/or procedures. In the FFBD method, the functions are organized and depicted by their logical order of execution. Each function is shown with respect to its logical relationship to the execution and completion of other functions. A node labeled with the function name depicts each function. Arrows from left to right show the order of execution of the functions. Logic symbols represent sequential or parallel execution of functions.

## HIPO and IPO




An expanded IPO Model.

HIPO for *hierarchical input process output* is a popular 1970s systems analysis design aid and documentation technique for representing the modules of a system as a hierarchy and for documenting each module.

It was used to develop requirements, construct the design, and support implementation of an expert system to demonstrate automated rendezvous. Verification was then conducted systematically because of the method of design and implementation.

The overall design of the system is documented using HIPO charts or structure charts. The structure chart is similar in appearance to an organizational chart, but has been modified to show additional detail. Structure charts can be usedto display several types of information, but are used most commonly to diagram either data structures or code structures.

## N2 Chart

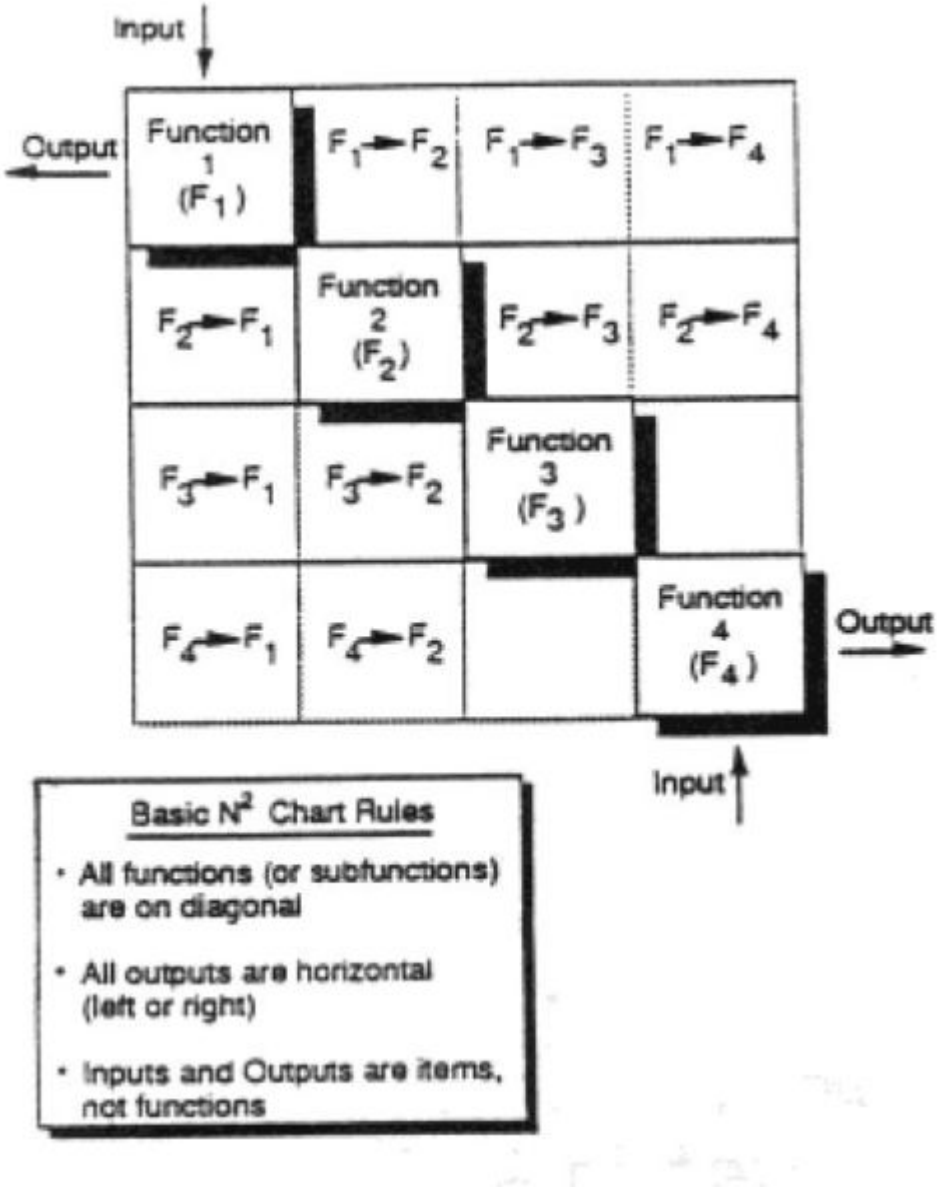
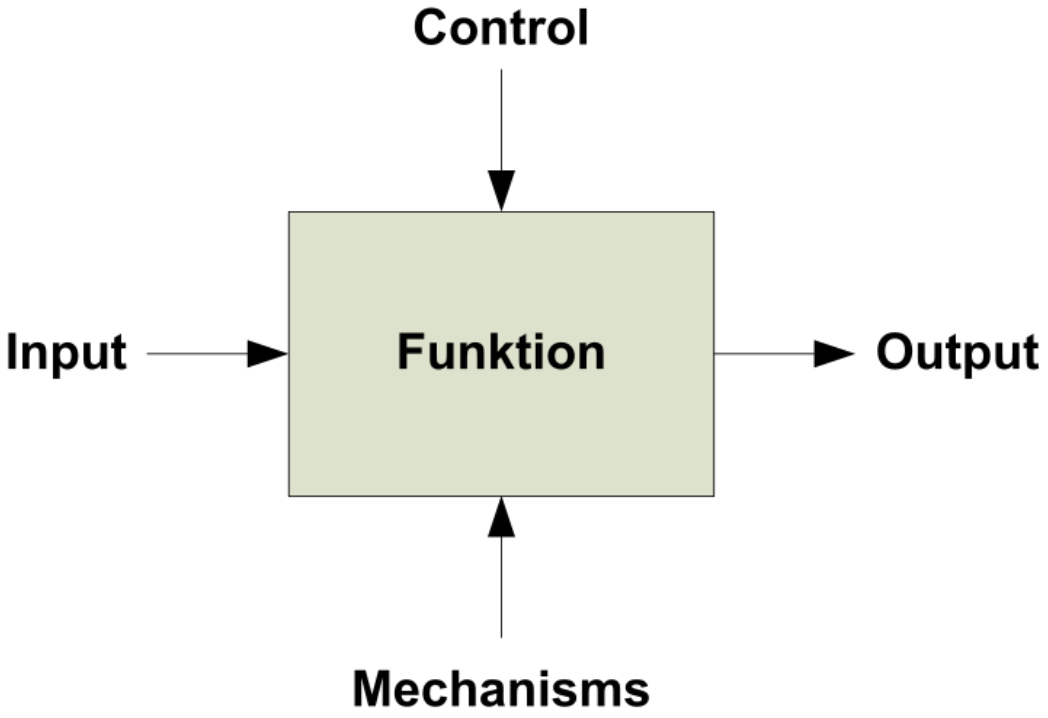



Figure 2. N2 chart definition.

The N2 Chart is a diagram in the shape of a matrix, representing functional or physical interfaces between system elements. It is used to systematically identify, define, tabulate, design, and analyze functional and physical interfaces. It applies to system interfaces and hardware and/or software interfaces.

The N2 diagram has been used extensively to develop data interfaces, primarily in the software areas. However, it can also be used to develop hardware interfaces. The basic N2 chart is shown in Figure 2. The system functions are placed on the diagonal; the remainder of the squares in the N x N matrix represent the interface inputs and outputs.

## Structured Analysis and Design Technique

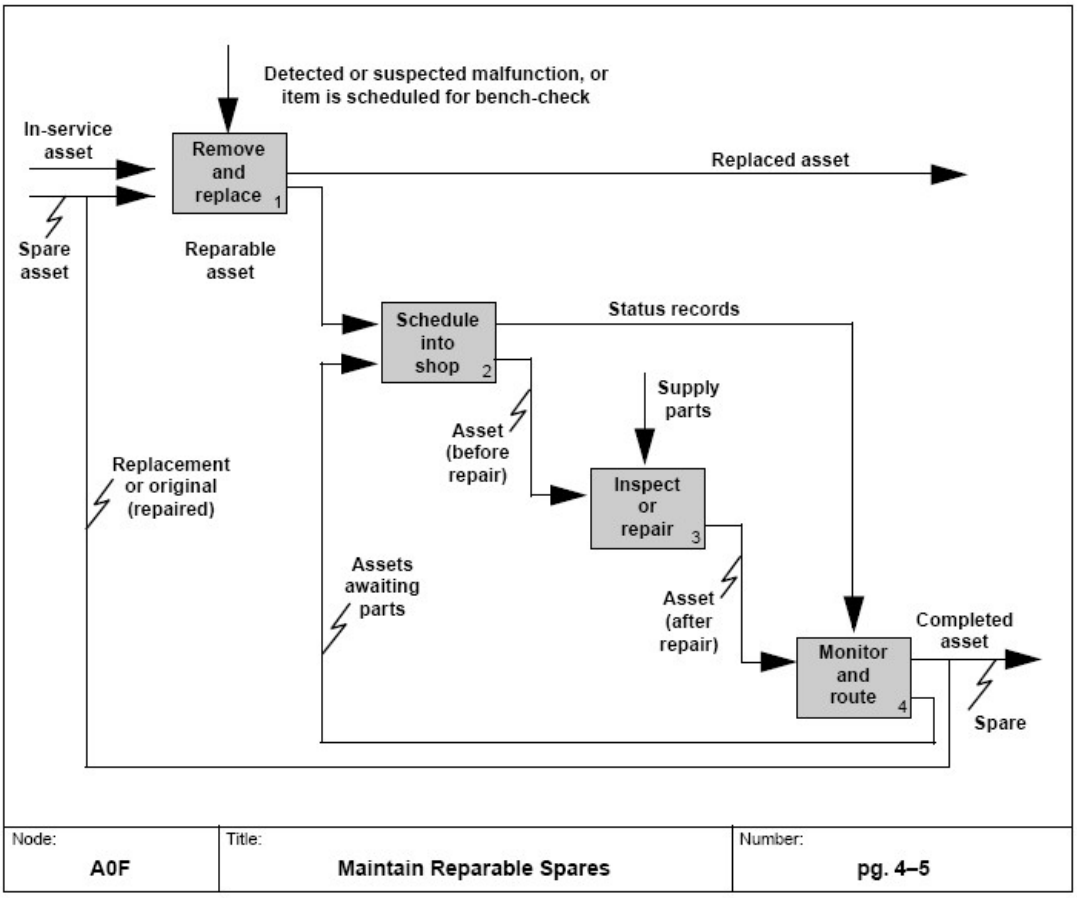



SADT basis element.

Structured Analysis and Design Technique (SADT) is a software engineering methodology for describing systems as a hierarchy of functions, a diagrammatic notation for constructing a sketch for a software application. It offers building blocks to represent entities and activities, and a variety of arrows to relate boxes. These boxes and arrows have an associated informal semantics. SADT can be used as a functional analysis tool of a given process, using successive levels of details. The SADT method allows to define user needs for IT developments, which is used in industrial Information Systems, but also to explain and to present an activity's manufacturing processes, procedures.

The SADT supplies a specific functional view of any enterprise by describing the functions and their relationships in a company. These functions fulfill the objectives of a company, such as sales, order planning, product design, part manufacturing, and human resource management. The SADT can depict simple functional relationships and can reflect data and control flow relationships between different functions. The IDEF0 formalism is based on SADT, developed by Douglas T. Ross in 1985.

## IDEF0

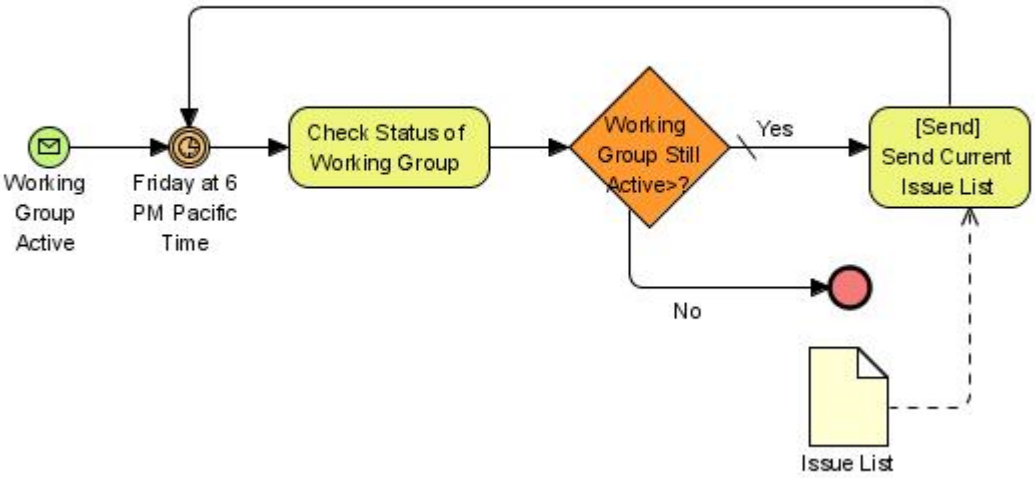



IDEF0 Diagram Example

IDEF0 is a function modeling methodology for describing manufacturing functions, which offers a functional modeling language for the analysis, development, re-engineering, and integration of information systems; business processes; or software engineering analysis. It is part of the IDEF family of modeling languages in the field of software engineering, and is built on the functional modeling language building SADT.

The IDEF0 Functional Modeling method is designed to model the decisions, actions, and activities of an organization or system. It was derived from the established graphic modeling language Structured Analysis and Design Technique (SADT) developed by Douglas T. Ross and SofTech, Inc.. In its original form, IDEF0 includes both a definition of a graphical modeling language (syntax and semantics) and a description of a comprehensive methodology for developing models. The US Air Force commissioned the SADT developers to develop a function model method for analyzing and communicating the functional perspective of a system. IDEF0 should assist in organizing system analysis and promote effective communication between the analyst and the customer through simplified graphical devices.

## Business Process Modeling Notation




Business Process Modeling Notation Example.

Business Process Modeling Notation (BPMN) is a graphical representation for specifying business processes in a workflow. BPMN was developed by Business Process Management Initiative (BPMI), and is currently maintained by the Object Management Group since the two organizations merged in 2005. The current version of BPMN is 1.1, and a major revision process for BPMN 2.0 is in progress.

The Business Process Modeling Notation (BPMN) specification provides a graphical notation for specifying business processes in a Business Process Diagram (BPD). The objective of BPMN is to support business process management for both technical users and business users by providing a notation that is intuitive to business users yet able to represent complex process semantics. The BPMN specification also provides a mapping between the graphics of the notation to the underlying constructs of execution languages, particularly BPEL4WS.

## Axiomatic Design

Axiomatic design is a top down hierarchical functional decomposition process used as a solution synthesis framework for the analysis, development, re-engineering, and integration of products, information systems, business processes or software engineering solutions. Its structure is suited mathematically to analyze coupling between functions in order to optimize the architectural robustness of potential functional solution models.

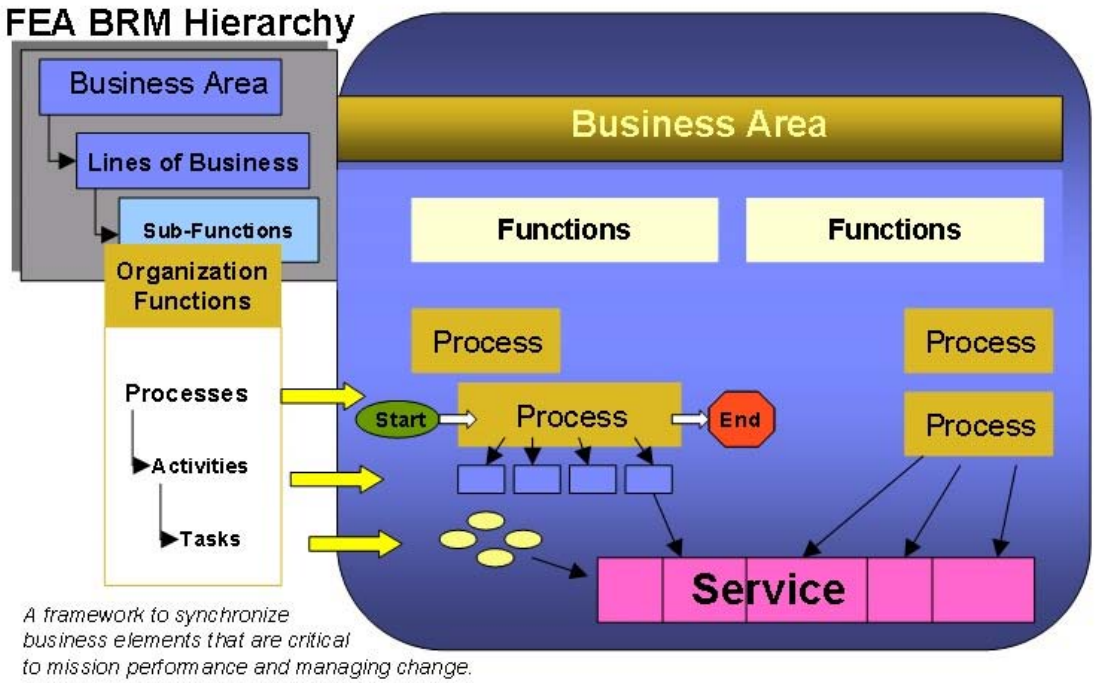
## *Other types of function models*

In the field of systems and software engineering numerous specific function and functional models have been defined. Here only a few general types will be explained.

### Business function model

A *Business Function Model* (BFM) is a general description or category of operations performed routinely to carry out an organization's mission. It can show the critical business processes in the context of the business area functions. The processes in the business function model must be consistent with the processes in the value chain models. Processes are a group of related business activities performed to produce an end product or to provide a service. Unlike business functions that are performed on a continual basis, processes are characterized by the fact that they have a specific beginning and an end point marked by the delivery of a desired output. The figure on the right depicts the relationship between the business processes, business functions, and the business area's business reference model.

### Business reference model




This FEA Business reference model depicts the relationship between the business processes, business functions, and the business area's business reference model.

A Business reference model is a reference model, concentrating on the functional and organizational aspects of the core business of an enterprise, service organization or government agency. In enterprise engineering a business reference model is part of an

Enterprise Architecture Framework or *Architecture Framework*, which defines how to organize the structure and views associated with an Enterprise Architecture.

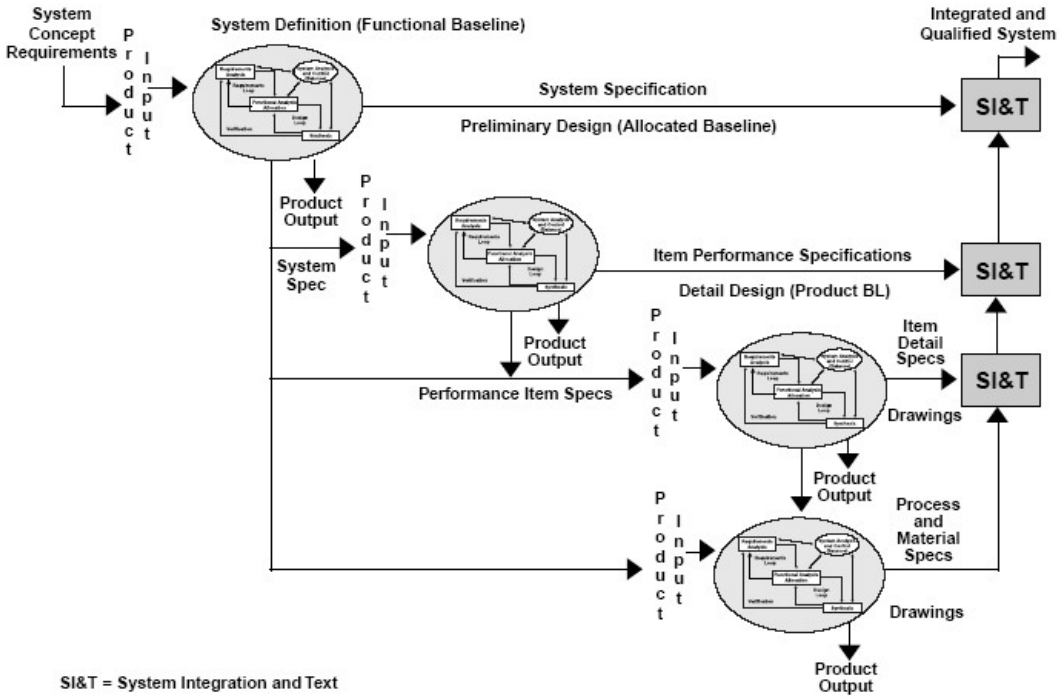
A reference model in general is a model of something that embodies the basic goal or idea of something and can then be looked at as a reference for various purposes. A business reference model is a means to describe the business operations of an organization, independent of the organizational structure that perform them. Other types of business reference model can also depict the relationship between the business processes, business functions, and the business area's business reference model. These reference model can be constructed in layers, and offer a foundation for the analysis of service components, technology, data, and performance.

## Operator function model

The *Operator Function Model* (OFM) is proposed as an alternative to traditional task analysis techniques used by human factors engineers. An operator function model attempts to represent in mathematical form how an operator might decompose a complex system into simpler parts and coordinate control actions and system configurations so that acceptable overall system performance is achieved. The model represents basic issues of knowledge representation, information flow, and decision making in complex systems. Miller (1985) suggests that the network structure can be thought of as a possible representation of an operator's internal model of the system plus a control structure which specifies how the model is used to solve the decision problems that comprise operator control functions.

Chapter 3

# Functional Specification




Systems engineering model of Specification and Levels of Development. During system development a series of specifications are generated to describe the system at different levels of detail. These program unique specifications form the core of the configuration baselines. As shown here, in addition to referring to different levels within the system hierarchy, these baselines are defined at different phases of the design process.

A **functional specification** (also, *functional spec*, *specs*, *functional specifications document (FSD)*, or *Program specification*) in systems engineering and software development is the documentation that describes the requested behavior of an

engineering system. The documentation typically describes what is needed by the system user as well as requested properties of inputs and outputs (e.g. of the software system).

## *Overview*

In systems engineering a specification is a document that clearly and accurately describes the essential technical requirements for items, materials, or services including the procedures by which it can be determined that the requirements have been met. Specifications help avoid duplication and inconsistencies, allow for accurate estimates of necessary work and resources, act as a negotiation and reference document for engineering changes, provide documentation of configuration, and allow for consistent communication among those responsible for the eight primary functions of Systems Engineering. They provide a precise idea of the problem to be solved so that they can efficiently design the system and estimate the cost of design alternatives. They provide guidance to testers for verification (qualification) of each technical requirement.

A functional specification does not define the inner workings of the proposed system; it does not include the specification how the system function will be implemented. Instead, it focuses on what various outside agents (people using the program, computer peripherals, or other computers, for example) might "observe" when interacting with the system. A typical functional specification might state the following:

*When the user clicks the OK button, the dialog is closed and the focus is returned to the main window in the state it was in before this dialog was displayed.*

Such a requirement describes an interaction between an external agent (the user) and the software system. When the user provides input to the system by clicking the OK button, the program responds (or should respond) by closing the dialog window containing the OK button.

It can be *informal*, in which case it can be considered as a blueprint or user manual from a developer point of view, or *formal*, in which case it has a definite meaning defined in mathematical or programmatic terms. In practice, most successful specifications are written to understand and fine-tune applications that were already well-developed, although safety-critical software systems are often carefully specified prior to application development. Specifications are most important for external interfaces that must remain stable.

## *Functional specification topics*

### Purpose

There are many purposes for functional specifications. One of the primary purposes on team projects is to achieve some form of team consensus on what the program is to achieve before making the more time-consuming effort of writing source code and test cases, followed by a period of debugging. Typically, such consensus is reached after one

or more reviews by the stakeholders on the project at hand after having negotiated a cost-effective way to achieve the requirements the software needs to fulfill.

### Process

In the ordered industrial software engineering life-cycle (waterfall model), functional specification describes what has to be implemented. The next system specification document describes how the functions will be realized using a chosen software environment. In not industrial, prototypical systems development, functional specifications are typically written after or as part of requirements analysis.

When the team agrees that functional specification consensus is reached, the functional spec is typically declared "complete" or "signed off". After this, typically the software development and testing team write source code and test cases using the functional specification as the reference. While testing is performed the behavior of the program is compared against the expected behavior as defined in the functional specification.

## *Types of software development specifications*

- Advanced Microcontroller Bus Architecture
- Bit specification
- Design specification
- Diagnostic design specification
- Multiboot Specification
- Product design specification
- Real-time specification for Java
- Software Requirements Specification

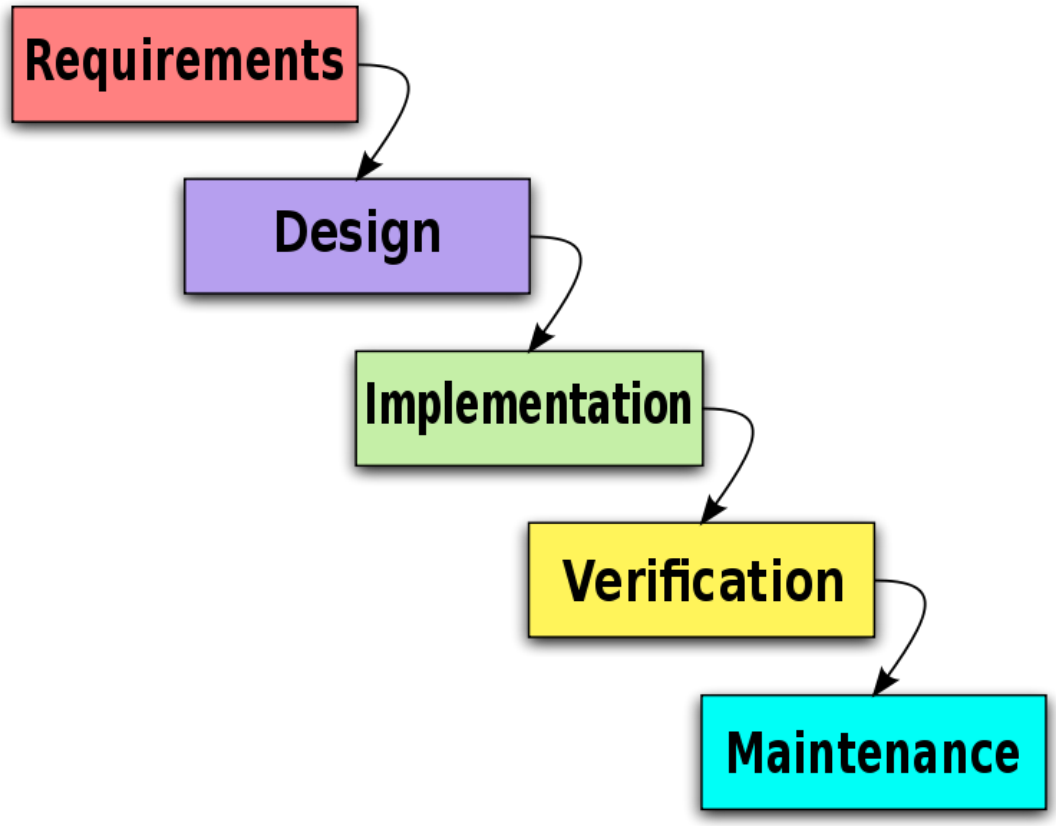
Chapter 4

# Dual Vee Model

The **Dual Vee Model** builds on the V-Model to cleanly depict the complexity associated with designing and developing systems. In systems engineering it defines a uniform procedure for product or project development. The model depicts concurrent development of a system's architecture as one Vee with each entity of that architecture as another Vee that intersects the architecture Vee. This clearly shows interactions and sequences in developing a complex system and a system of systems.

## *Background*

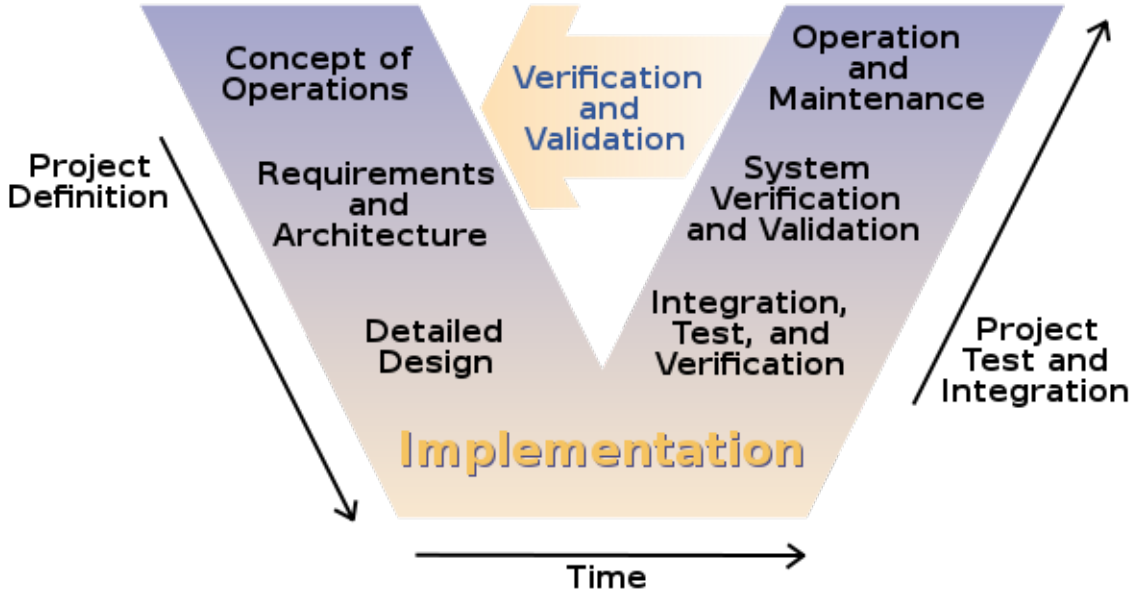
### Waterfall Model



The unmodified "waterfall model". Progress flows from the top to the bottom, like a waterfall.

The Waterfall Model breaks up the development process into development phases. The name implies that design decisions flow from the requirements, implementation decisions flow from the design, etc. On a large project, many different people only work on each part. So a designer may ask, "What am I trying to design?" and the response would be, "You're trying to design something that will satisfy the product requirements." The builder may ask, "What am I trying to build?" and the response would be, "You're trying to build what was designed." etc.

## Vee Model



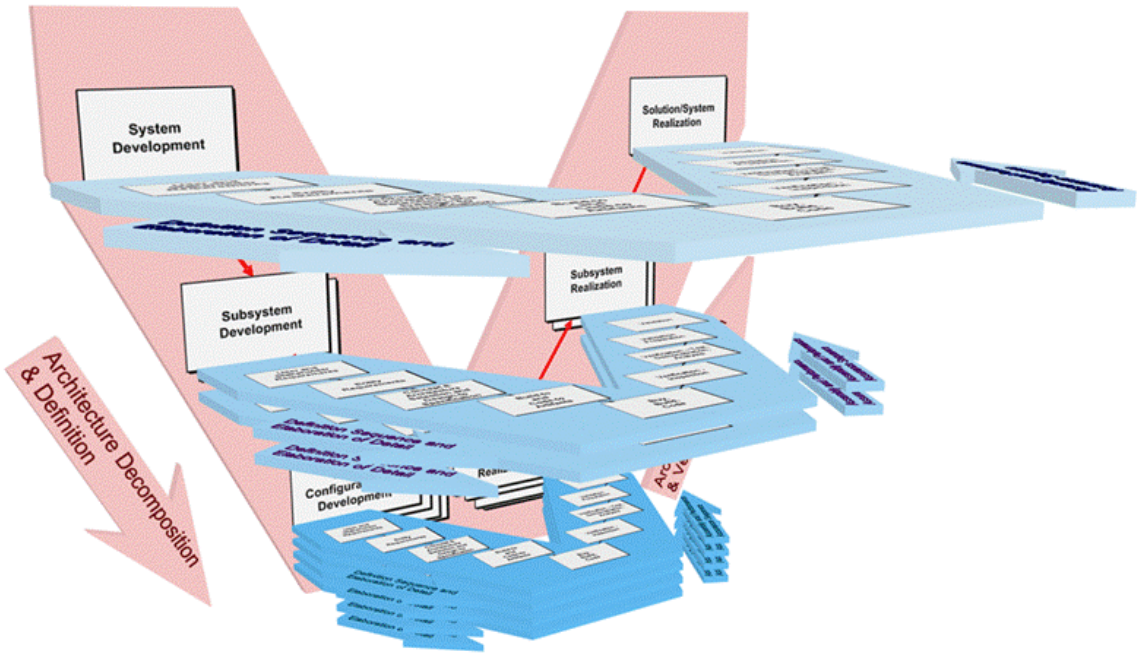

The V-model of the Systems Engineering Process.

The Vee Model organizes development phases into levels of complexity with the most complex item on top and least complex item on bottom (a.k.a. Lowest Configuration Item). This places the requirements at the beginning next to the product's operation at the end, and the design next to verification. This makes sense because when developer delivers a product to the customer, the customer may ask, "Why should I accept this product?" and the developer should answer, "Because it meets your (the customer's) requirements." The requirements are connect to the operation. When performing product testing, the test engineer may ask, "What tests should I conduct?" and the designer should answer, "You should conduct tests to show the product that was built matches the design." Verification is connected to the design.

The Vee Model can be expanded in several ways to meet system requirements. It can include the seven INCOSE layers of system complexity (i.e. system, element, subsystem,

assembly, subassembly, component, and part). It can include decomposition, definition, integration, and verification. It may also include user/stakeholder participation, opportunity and risk management, and problem resolution.

## Dual Vee Model



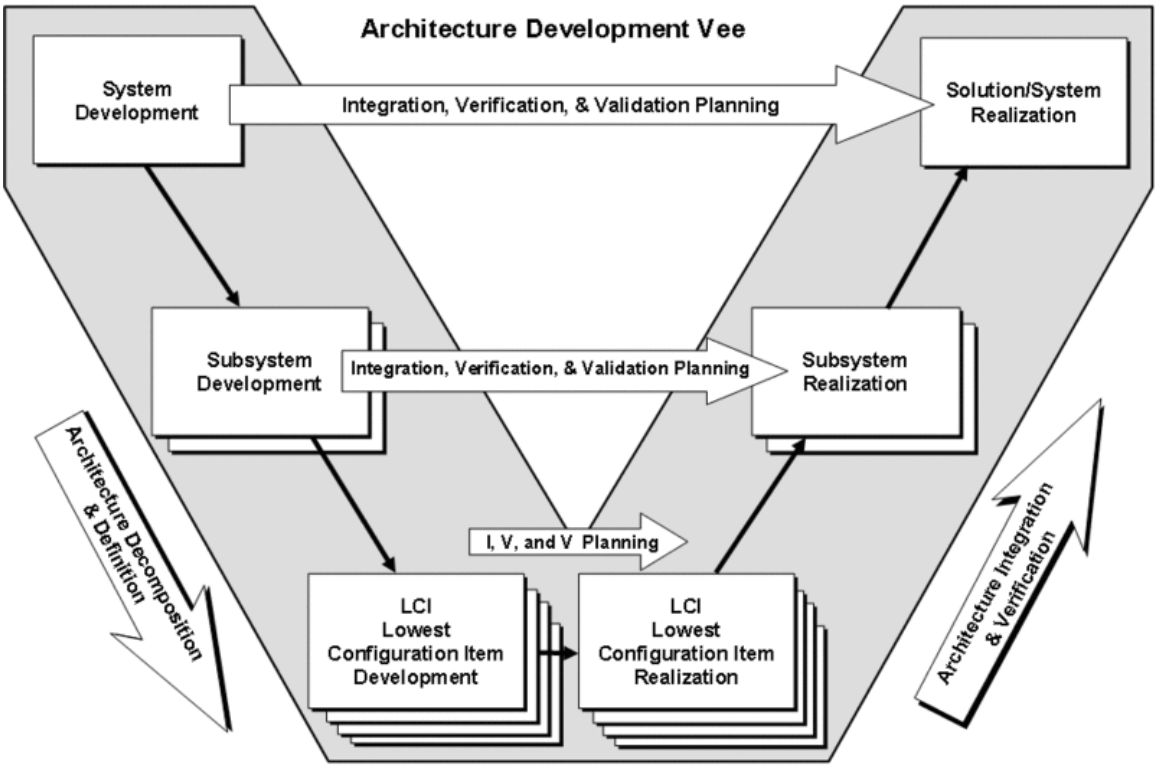

Architecture and Entity Vees Intersecting. Source - Kevin Forsberg and Hal Mooz 2006.

The Dual Vee Model builds on the Vee Model to manage a system of systems. An architecture Vee manages the system and entity Vees branch off the architecture Vee to manage sub-systems.

For example, GPS includes a constellation of satellites, a ground control network, and millions of users worldwide. Each satellite, ground control center, and GPS receiver is a complex system that could be managed by a separate Entity Vee. Development of a satellite can affect the design, manufacturing, or cost of receivers. Similarly, development of a receiver can affect design, manufacturing, or cost of satellites. So everything must be integrated into a system of systems that are developed within a larger Architecture Vee.

## *The Architecture Vee*

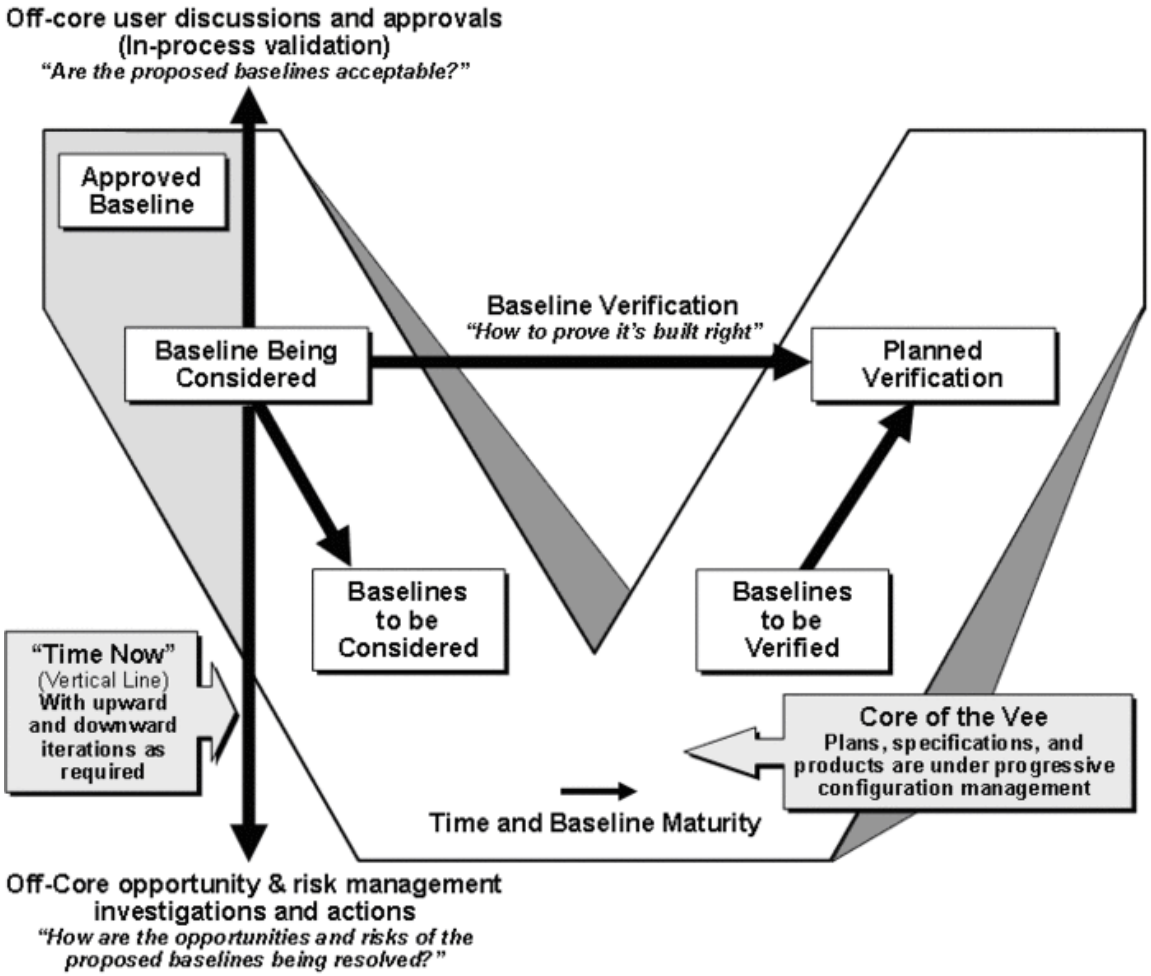
When developing complicated systems, a system engineer must manage a system baseline configuration from start to finish. The baseline can include design documents, user manuals, the product itself, and should answer every What?, Why?, and Who? for a system's architecture. At each development phase, there will be changes to the system, which will change the baseline.




Architecture Development Vee Model (Provides What, Why, and Who). Source - Kevin Forsberg and Hal Mooz 2006.

The core of the Vee is the evolving architecture baseline from initial requirements to a delivered system. The Architecture Vee produces the what, why, and who (which entity level) is responsible for a system's architecture.

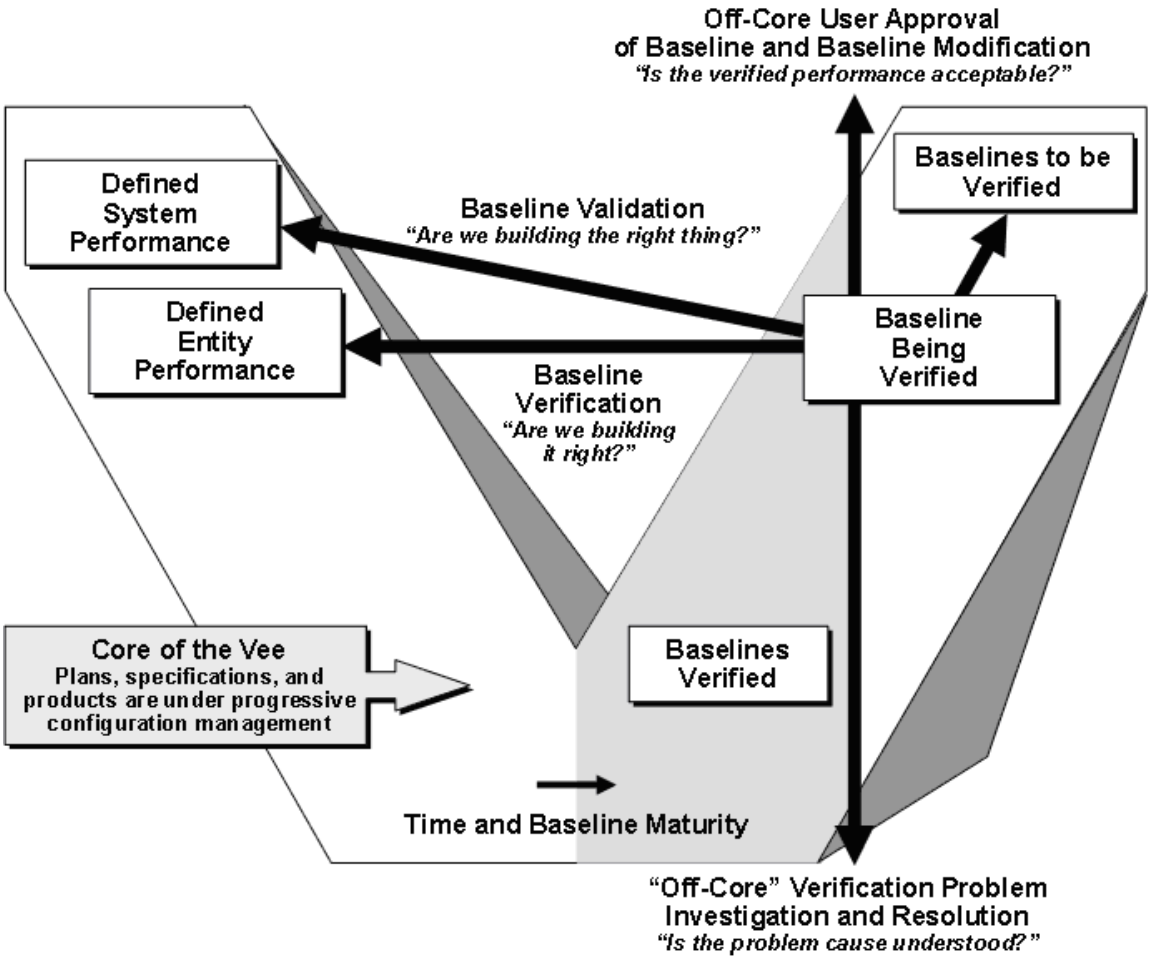
Downward off-Vee core investigations (figure - below) facilitate gaining knowledge to justify architecture baseline decisions made on the Vee core. Upward off core communication with customers and users facilitates in-process validation keeping the stakeholders abreast of and committed to the evolving baseline. Note that in all Vee representations time and maturity move from left to right. Just as we cannot move backward in time, so too one cannot move from right to left in the Vee model representation. Iteration is essential in system development, and all iteration is done vertically off-core, upward to users and customers (which is in-process validation), and downward for opportunity and risk management, as shown in the following figure.




Vee Model - Opportunity and Risk Investigations. Source - Kevin Forsberg and Hal Mooz 2006.

The left leg off-Vee core investigations center around what concept is best and what architecture is best for that concept. For example, commercial products usually face the dilemma as to whether batteries should be standard, unique, replaceable, or not. Downward off-Vee core investigations and analysis can facilitate determining the most desirable approach that would then be baselined on the Vee core if the stakeholders agree. Similar investigations can prove the viability and technical feasibility of candidate concepts.

Right leg off-Vee core downward investigations (figure - below) are directed at investigating integration anomalies to determine their root cause and to correct them. Upward communication with stakeholders determines if they can live with the as-integrated and as-verified performance.



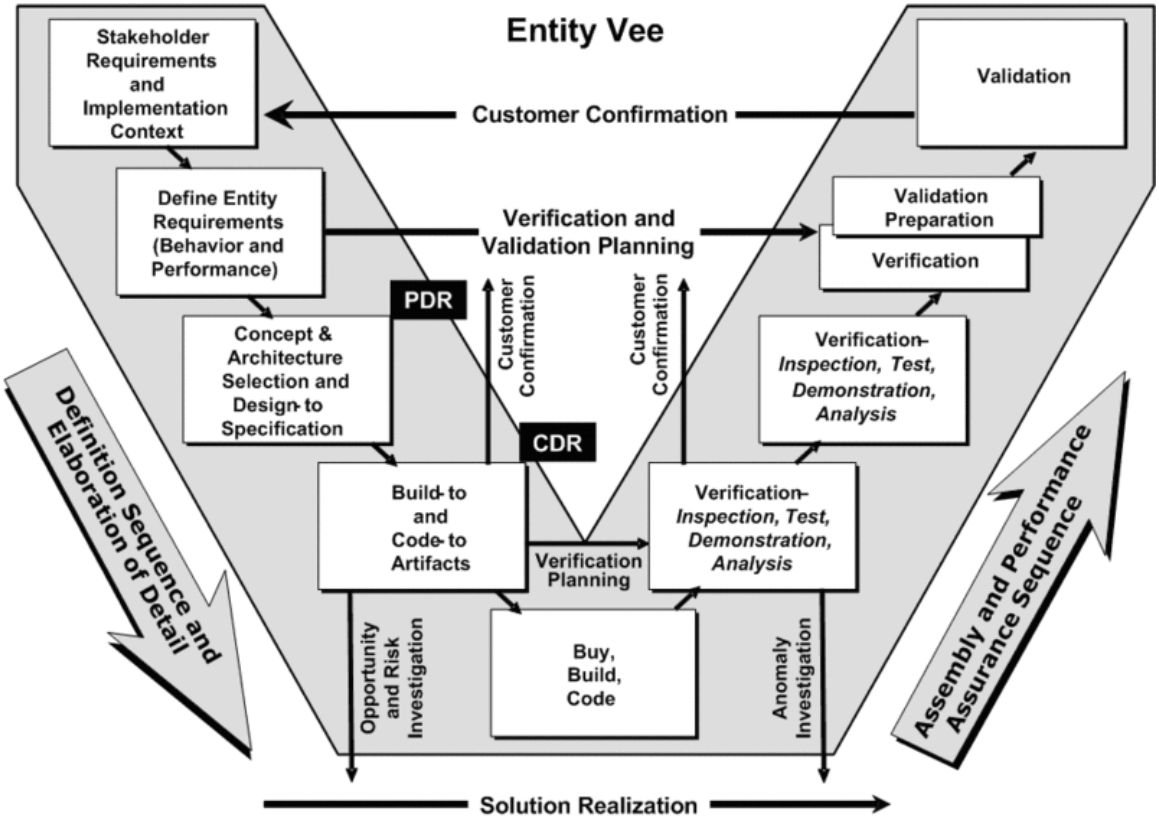

Vee Model - Integration and Verification. Source - Kevin Forsberg and Hal Mooz 2006.

At each decomposition level there is a direct correlation between activities on the left and right sides of the Vee. This is deliberate. For example, the method of integration, verification, and validation to be used on the right must be determined on the left as concepts are defined at each decomposition level. This minimizes the chances that concepts are conceived in a way that cannot be carried out.

## *The Entity Vee Model*

The Entity Vee illustrates the entity development and realization process which describes how each entity will be obtained (development, purchase, reuse, etc.). An Entity Vee (figure - below) exists for every entity of the architecture from the system, down to the lowest configuration items (LCIs), such as computer software units or hardware components. All activities within an Entity Vee reside at the same architecture level (System, Subsystem, LCI). The left Vee leg represents entity definition elaboration from very sketchy user requirements, through concept determination and on to design-to specifications and fully detailed build-to artifacts. The right Vee leg represents the

sequence of entity assembly and performance assurance on through verification and validation of the entity.




Entity Vee Model (Provides How). Source - Kevin Forsberg and Hal Mooz 2006.

At each elaboration, there is a direct correlation between activities on the left and right legs of the Entity Vee. Again this is deliberate. The method of verification to be used on the right Vee leg must be defined as requirements are developed on the left, otherwise requirements might be created that could not be verified. For example “user friendly” is a valid requirement, but it is unverifiable. Instead, a requirement that a computer screen display have “no more than five lines of 14-point text” defines one user's view of “user friendly” in measurable terms. Verification plans should be baselined to ensure verification requirements and methods are known and planned for at the design-to decision gate, commonly called Preliminary Design Review (PDR). Draft verification procedures based on the verification requirements, verification plan, and proposed entity design should be available at the build-to and code-to decision gate, commonly called Critical Design Review (CDR). This reduces the chances that verification as specified cannot in fact be performed.

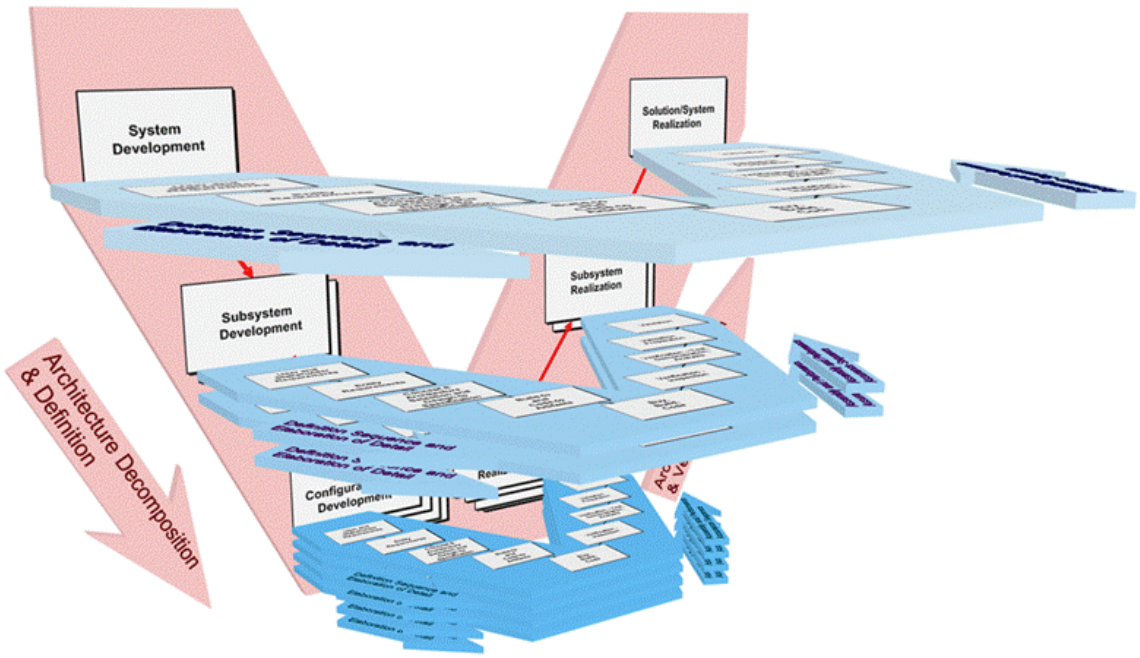
The vertical dimension of the Entity Vee is baseline elaboration at the selected architecture level and the core of the Entity Vee represents entity baseline elaboration progression. Also included (similar to the Architecture Vee) are the activities associated with opportunity and risk management, pursued downward and off-core to the level of

detail necessary for issue evaluation and resolution. For example, laboratory test of a computer chip or of software code may be necessary to confirm technical feasibility. Unlike the commonly held view of the Waterfall Model, there is no prohibition against doing exploratory design and analysis at any point in the project cycle to investigate or prove performance or feasibility. Unlike the Spiral Model, the Vee opportunity and risk investigations may be performed either in series or in parallel with the on-core development work, rather than being conducted sequentially and prior to the design development process. Hardware and software requirements-understanding models or technical feasibility models are encouraged early in the project cycle to pursue opportunities, such as new technologies, and to reduce risk. For instance, to evaluate a concept of a manual override versus full automation, technical feasibility of the two concepts could be modeled with selection based on response time versus cost. Customer confirmation could then provide valuable in-process validation of the preferred approach.

In the right leg, downward off-core investigations are applied to resolve assembly and verification anomalies. This may require descending to design errors, a cold solder joint, or operator error and the like. Upward off-core user interactions obtain user and customer confirmation or rejection of the realized performance. Note that in the entity Vee these interactions address individual entity solutions and not the integration of the architecture which is conducted on the Architecture Vee. At any level of decomposition, the customer of an entity is the manager of the next higher level of decomposition. For example, the power subsystem manager is the customer of the battery and is responsible for battery validation.

## *Dual Vees: Intersecting Architecture and Entity Vees*

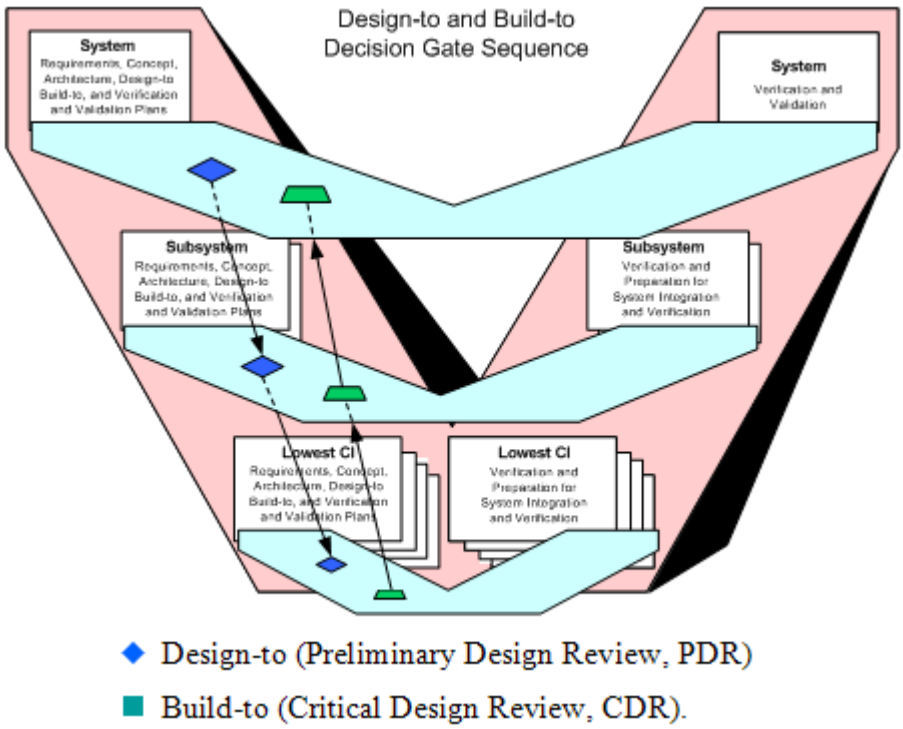
To evolve user needs into a system that satisfies those needs requires a best value solution for every entity of the architecture. This can be visualized by positioning Entity Vees orthogonal to the Architecture Vee as shown in the figure below. For each entity of the Architecture Vee there is a corresponding Entity Vee that addresses the entity development and realization. For example, the Architecture Vee of the figure below contains two subsystems hence there are two Entity Vees to represent the concurrent development of those two subsystems.




Architecture and Entity Vees Intersecting. Source - Kevin Forsberg and Hal Mooz 2006.

## Phasing of decision gates

Architecture entities are developed and integrated into the system architecture in a phased sequence consistent with systems engineering best practices. The figure below provides a three dimensional view of Design-to and Build-to Decision Gate phasing




Design-to and Build-to Decision Gate Sequence. Source - Kevin Forsberg and Hal Mooz 2006.

For simplicity of illustration, only one Entity Vee is shown intersecting the Architecture Vee at each decomposition level. Note that the design-to sequence is top down, starting at the system level and proceeding consistent with decomposition to the lowest configuration item level (LCI). This sequence ensures that there is proper top down requirements flowdown and traceability.

When build-to and code-to artifacts, including draft verification procedures, are ready for baselining, the build-to decision gate sequence is conducted bottom up to prove the feasibility of building or coding the designs. The decision gate also confirms that, if the solution is built according to the build-to artifacts, the required performance will be achieved. This sequence ensures that, if the entity designs satisfy their design-to requirements, the entities will integrate into the next higher level entity, will perform as expected, and will meet user requirements.

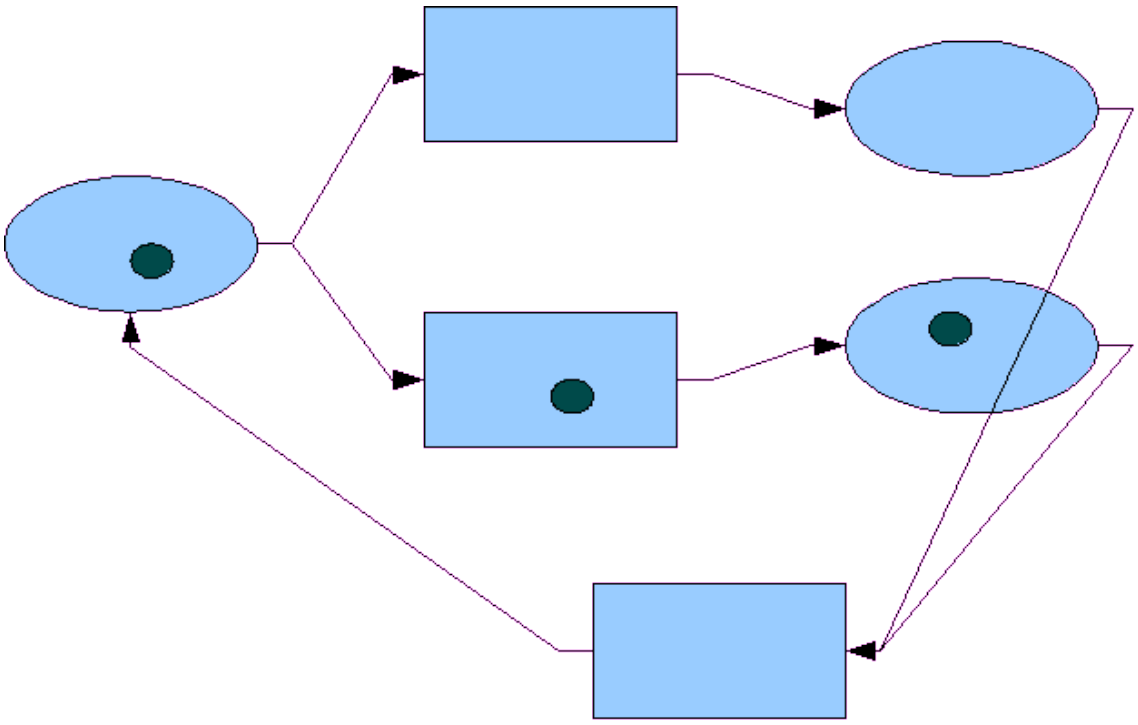
Chapter 5

# Dualistic Petri Nets

**Dualistic Petri nets** (dPNs) are a process-class variant of Petri nets. Like Petri nets in general and many related formalisms and notations, they are used to describe and analyze process architecture.

## *Process Modeling with dPNs*

A simple, yet powerful way to model process architecture is using the dualistic extension of Petri nets called dualistic Petri nets (dPNs). A Petri net (PN) is a graphical, bipartite modeling language that intuitively and mathematically represent theoretical relationships of moving objects in a network of interconnected constructs. Classical Place/Transition PNs can represent theoretical processes, where the movement of objects implies their transformation, but is too absolute to be pragmatic in representing real-world processes. The real world is dualistic in nature and process is a dualistic phenomenon, this can not be easily represented using a digital-type modeling system. Instead, dualistic extensions to Place/Transition PNs have been introduced and used successfully in modeling the architecture of computer-based systems and business processes.



Animation of a Dualistic Petri Net Simulation: Rectangles = Transformations, Ovals = Places

Among the distinctions of dPNs from classical PNs is space and time (due to energy use) in both the place construct and transformation construct. This results in the simulation effect of **marked transformations** that allow for the explicit representation of parallel processing, multiprocessing, and the implicit representation of deterioration of objects – all unique to dualistic Petri nets.

## *Architecture*

Besides a propensity to modeling dualistic real-world behavior, PNs also offer a way to manage complex process systems hierarchically. Using classical PN construction rules, Petri nets of Petri nets can be built and a hierarchical conception of a complex process system can be studied. This structure of hierarchical abstraction is the heart of process architecture!

### Bottom-Up: Starting with the manifested process

Dualistic Petri nets are capable of modeling any process system at its manifested level. When reverse engineering a manifested process, dPNs have a one-to-one correspondence of dPN construct to any manifested process piece, that is, it is isomorphic to the implementation language of the manifested process. For example, several lines of software code could be represented by one dPN transformation construct. Once the manifested process is completely represented by a network of dPNs, small, well-coupled groups of dPN constructs can be lumped together to form higher level dPN constructs –

creating a network of dPNs at a higher level of hierarchical abstraction. Each level of abstraction is consistent with its adjacent levels of abstraction and the rules governing them at each level are the exactly the same because PN abstractions are homomorphic. Now the process design can be considered at various levels of abstraction as deemed appropriate by the process architect, allowing for studies in its dynamic behavior and performance.

A typical use of reverse engineering using dPNs in the business world is in the documentation of processes for quality certification against standards like ISO 9000. In this case, dPNs are used to model pieces of the business process, which are then combined to form an overall enterprise architecture. The process system can be studied to determine each piece's capability and show where risks occur. Requirements are then reverse-engineered and applied at corresponding dPN constructs. Trouble spot processes can be identified and slated for reengineering. The overall dPN map not only gives quality entities the necessary information about a business's current process, but it also gives the process architect a blue print from which to manage and improve said processes. This is a major portion of quality engineering.

## Top-down: From Idea to Implementation

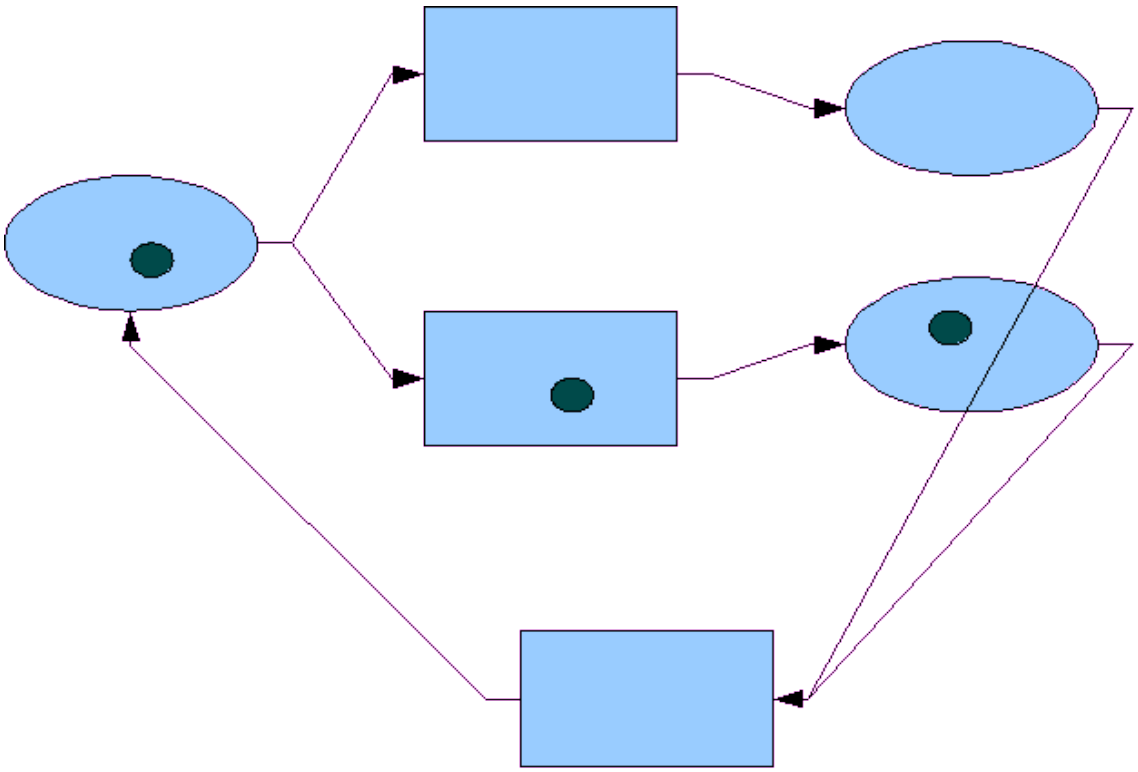
dPN modeling of a new process system starts at a high level of hierarchical abstraction. To design a complex process system, such as a sophisticated hardware component or a major project, the process architect must first define the problem space. Since the problem space is itself a process system, dPNs can be used for its modeling. Abstract dPNs that are yet to be implemented are specified within the context of the problem space. These constructs define the solution space within its context network. It is now up to the process architect to traverse down the hierarchical abstraction dimension, proposing new process designs for the solution space in an iterative manner until specifying the actual implementation in the specific implementation language.

This method for designing a complex process system is reflected in the general software development methodology known as the waterfall model. Actually, this method is not well-suited for the development of complex software without adjusting it to handle the step-wise decomposition of the process architecture. This decomposition occurs entirely within the domain of dPNs from the problem space context model down to the final mapping of the implementation language.

## Process Structure

Whether a dPN hierarchical network map was created from the bottom up or from the top down, it shows the structure of the process system. Complex process systems, such as large software programs, will have several layers of hierarchical abstraction. At the top of the structure is one process represented by a couple of dPN constructs. Each subsequent layer below this process is the decomposition of the dPN constructs made up of more dPNs that are in turn decomposed. The “parent” dPN of a set of decomposed dPNs has associated with it requirements that apply to the decomposed network. These

requirements were determined by studying the parent dPN's **suprastructure** or the hierarchical structure *above* the construct. The decomposed "children" dPNs form the **infrastructure** or the hierarchical structure *below* the parent dPN.



dPNet-modeled Process Architecture

In complex computer design, requirements are generated and infrastructures proposed. Chosen infrastructures are then further decomposed by determining the new constructs' requirements and decomposing them further in this iterative fashion until the dPNs are decomposed into the implementation language of software or hardware specification. The final hierarchical dPN map documents the architectural decisions that were accepted and a specification is in place that can be used to maintain the system's future evolution.

In business processes, process requirements are policies that must be fulfilled by acceptable procedures. Complex procedures can be specified by simpler procedures. Since business processes are processes, dPNs are an ideal modeling language for them – especially when considering complex business processes like logistics.

## *Conclusion*

The entire network of dualistic Petri nets becomes the architecture specification of the process system. If the problem and solution space are entirely in software, it is known as

software architecture. If the problem and solution space are business processes, it is known as enterprise architecture. If the problem and solution space are networked equipment, it is known as network architecture. What is important to each of these applications and to any other process system of varying complexity is that the hierarchical map of the system's structure created by the network of dPNs allows the process architect to study the behavior and performance of the system, keeps architectural design decisions documented, and organizes process requirements along the architectural structure.

Chapter 6

# Change Management (Engineering)

The **change management** process in systems engineering is the process of requesting, determining attainability, planning, implementing, and evaluating of changes to a system. It has two main goals: supporting the processing of changes – which is mainly discussed here – and enabling traceability of changes, which should be possible through proper execution of the process described here.

## *Introduction*

There is considerable overlap and confusion between change management, change control and configuration management. The definition below does not yet integrate these areas.

Change management is an important process, because it can deliver vast benefits (by improving the system and thereby satisfying "customer needs"), but also enormous problems (by ruining the system and/or mixing up the change administration). Furthermore, at least for the Information Technology domain, more funds and work are put into system maintenance (which involves change management) than to the initial creation of a system. Typical investment by organizations during initial implementation of large ERP systems is 15-20% of overall budget.

In the same vein, Hinley describes two of Lehman's laws of software evolution: the law of continuing change (i.e. systems that are used must change or automatically become less useful) and the law of increasing complexity (i.e. through changes the structure of a system becomes ever more complex and more resources are needed to simplify it).

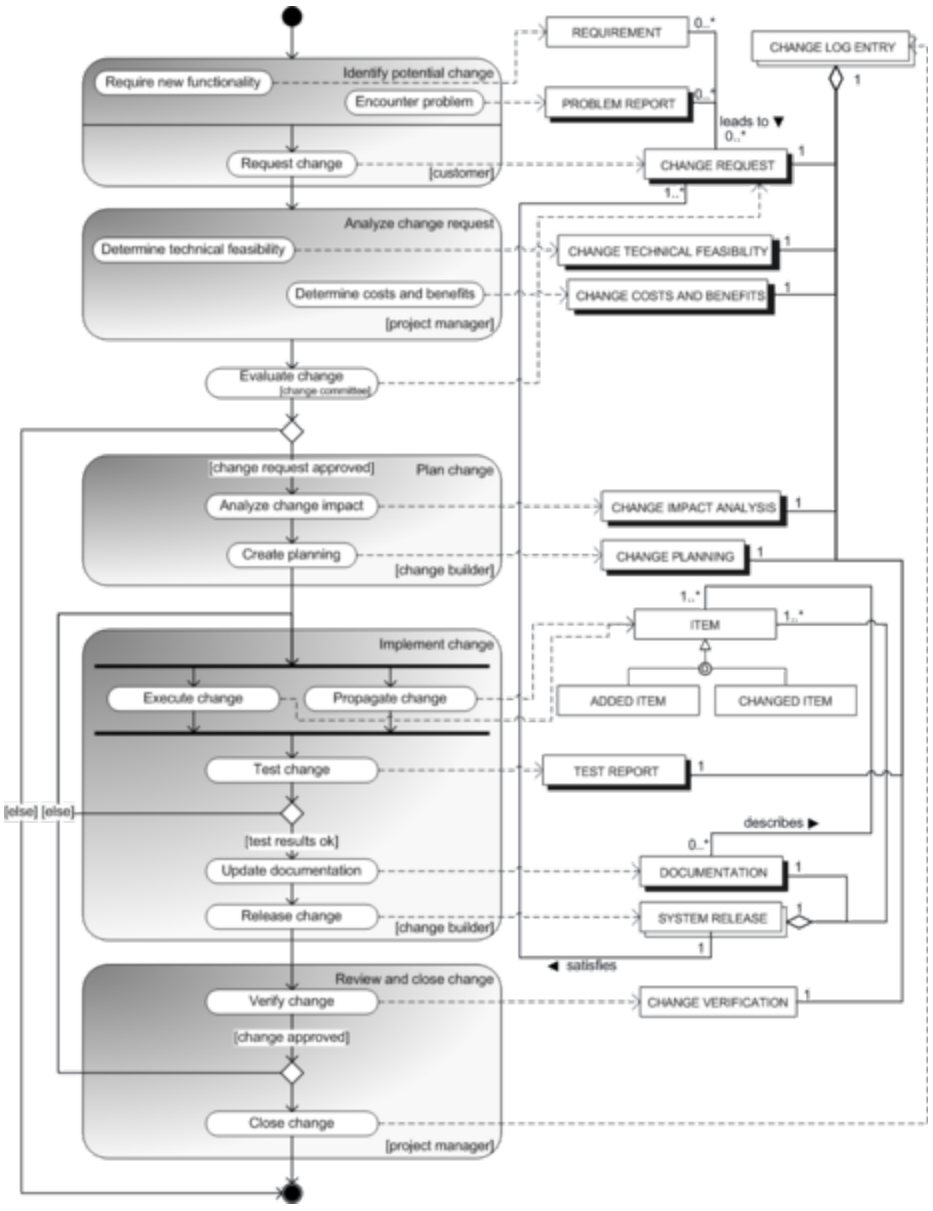
The field of manufacturing is nowadays also confronted with many changes due to increasing and worldwide competition, technological advances and demanding customers. Therefore, (efficient and effective) change management is also of great importance in this area.

It is not unthinkable that the above statements are true for other domains as well, because usually, systems tend to change and evolve as they are used. Below, a generic change management process and its deliverables are discussed, followed by some examples of instances of this process.

Notes: In the process below, it is arguable that the change committee should be responsible not only for accept/reject decisions, but also prioritization, which influences how change requests are batched for processing.

## *The process and its deliverables*

For the description of the change management process, the meta-modeling technique is used. Figure 1 depicts the process-data diagram.



## Activities

There are six main activities, which jointly form the change management process. They are: Identify potential change, Analyze change request, Evaluate change, Plan change, Implement change and Review and close change. These activities are executed by four different roles, which are discussed in Table 1. The activities (or their sub-activities, if applicable) themselves are described in Table 2.

Table 1: Role descriptions for the change management process

| Role | Description |
|---|---|
| **Customer** | The customer is the role that requests a change due to problems encountered or new functionality requirements; this can be a person or an organizational entity and can be in- or external to the company that is asked to implement the change. |
| **Project manager** | The project manager is the owner of the project that the CHANGE REQUEST concerns. In some cases there is a distinct change manager, who in that case takes on this role. |
| **Change committee** | The change committee decides whether a CHANGE REQUEST will be implemented or not. Sometimes this task is performed by the project manager as well. |
| **Change builder** | The change builder is the person who plans and implements the change; it could be argued that the planning component is (partially) taken on by the project manager. |

Table 2: Activity descriptions for the change management process

| Activity | Sub-activity | Description |
|---|---|---|
| **Identify potential change** | Require new functionality | A customer desires new functionality and formulates a REQUIREMENT. |
| | Encounter problem | A customer encounters a problem (e.g. a bug) in the system and this leads to a PROBLEM REPORT. |
| | Request change | A customer proposes a change through creation of a CHANGE REQUEST. |
| **Analyze change request** | Determine technical feasibility | The project manager determines the technical feasibility of the proposed CHANGE REQUEST, leading to a CHANGE TECHNICAL FEASIBILITY. |
| | Determine costs and benefits | The project manager determines the costs and benefits of the proposed CHANGE REQUEST, resulting in CHANGE COSTS AND BENEFITS. This and the above sub-activity can be done in any order and they are independent of each other, hence the modeling as unordered activities. |

| | | |
|---|---|---|
| **Evaluate change** | | Based on the CHANGE REQUEST, its CHANGE TECHNICAL FEASIBILITY and CHANGE COSTS AND BENEFITS, the change committee makes the go/no-go decision. This is modeled as a separate activity because it is an important process step and has another role performing it. It is modeled as a sub-activity (without any activity containing it) as recommended by Remko Helms (personal communication). |
| **Plan change** | Analyze change impact | The extent of the change (i.e. what other items the change effects) is determined in a CHANGE IMPACT ANALYSIS. It could be argued that this activity leads to another go/no-go decision, or that it even forms a part of the Analyze change request activity. It is modeled here as a planning task for the change builder because of its relationship with the activity Propagate change. |
| | Create planning | A CHANGE PLANNING is created for the implementation of the change. Some process descriptions (e.g. Mäkäräinen, 2000) illustrate that is also possible to 'save' changes and process them later in a batch. This activity could be viewed as a good point to do this. |
| **Implement change** | Execute change | The change is 'programmed'; this activity has a strong relationship with Propagate change, because sometimes the change has to be adapted to other parts of the system (or even other systems) as well. |
| | Propagate change | The changes resulting from Execute change have to be propagated to other system parts that are influenced by it. Because this and the above sub-activity are highly dependent on each other, they have been modeled as concurrent activities. |
| | Test change | The change builder tests whether what (s)he has built actually works and satisfies the CHANGE REQUEST. As depicted in the diagram, this can result in an iterative process together with the above two sub-activities. |
| | Update documentation | The DOCUMENTATION is updated to reflect the applied changes. |
| | Release change | A new SYSTEM RELEASE, which reflects the applied change, is made public. |
| **Review and close change** | Verify change | The implementation of the change in the new SYSTEM RELEASE is verified for the last time, now by the project manager. Maybe this has to happen before the release, but due to conflicting |

| | |
|---|---|
| | literature sources and diagram complexity considerations it was chosen to model it this way and include this issue. |
| Close change | This change cycle is completed, i.e. the CHANGE LOG ENTRY is wrapped up. |

## Deliverables

Besides activities, the process-data diagram (Figure 1) also shows the deliverables of each activity, i.e. the data. These deliverables or concepts are described in Table 3; in this context, the most important concepts are: CHANGE REQUEST and CHANGE LOG ENTRY.

A few concepts are defined by the author (i.e. lack a reference), because either no (good) definitions could be found, or they are the obvious result of an activity. These concepts are marked with an asterisk ('*'). Properties of concepts have been left out of the model, because most of them are trivial and the diagram could otherwise quickly become too complex. Furthermore, some concepts (e.g. CHANGE REQUEST, SYSTEM RELEASE) lend themselves for the versioning approach as proposed by Weerd , but this has also been left out due to diagram complexity constraints.

Table 3: Concept descriptions for the change management process

| Concept | Description |
|---|---|
| **REQUIREMENT** | A required functionality of a component (or item; NASA, 2005). |
| **PROBLEM REPORT** | Document describing a problem that cannot be solved by a level 1 help desk employee; contains items like date, contact info of person reporting the problem, what is causing the problem, location and description of the problem, action taken and disposition, but this is not depicted in the diagram (Dennis, et al., 2002). |
| **CHANGE REQUEST** | Document that describes the requested change and why it is important; can originate from PROBLEM REPORTS, system enhancements, other projects, changes in underlying systems and senior management, here summarized as REQUIREMENTS (Dennis, et al., 2002). Important attribute: 'go/no-go decision', i.e. is the change going to be executed or not? |
| **CHANGE LOG ENTRY*** | Distinct entry in the collection of all changes (e.g. for a project); consists of a CHANGE REQUEST, CHANGE TECHNICAL FEASIBILITY, CHANGE COSTS AND BENEFITS, CHANGE IMPACT ANALYSIS, CHANGE PLANNING, TEST REPORT and CHANGE VERIFICATION. Not all these have to be included if the process is terminated earlier (i.e. if the change is not |

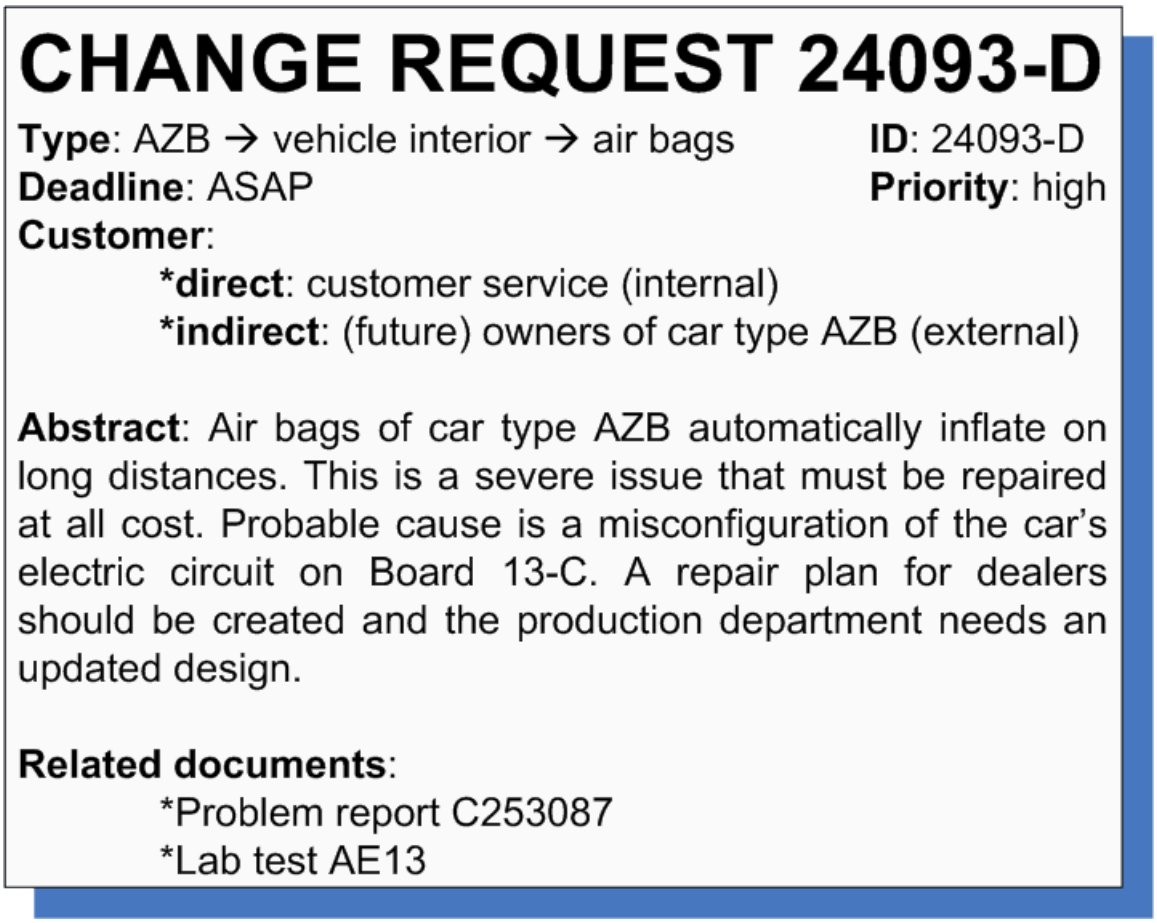
implemented).

| | |
|---|---|
| **CHANGE TECHNICAL FEASIBILITY** | Concept that indicates whether or not "reliable hardware and software, technical resources capable of meeting the needs of a proposed system [i.e. change request] can be acquired or developed by an organization in the required time" (Vogl, 2004). |
| **CHANGE COSTS AND BENEFITS** | The expected effort required to implement and the advantages (e.g. cost savings, increased revenue) gained by implementing the change. Also named economic feasibility (Vogl, 2004). |
| **CHANGE IMPACT ANALYSIS** | An assessment of the extent of the change (Rajlich, 1999). |
| **CHANGE PLANNING** | "A scheme, method or design for the attainment of some objective or to achieve something [i.e. the change]" (Georgetown University, n.d.), in this case the change. |
| **ITEM** | "A non-specific term used to denote any product, including systems, subsystems, assemblies, subassemblies, units, sets, accessories, computer programs, computer software or parts" (Rigby, 2003); has (overlapping) subtypes ADDED ITEM and CHANGED ITEM. |
| **ADDED ITEM*** | Self-explanatory: a newly created ITEM; subtype of ITEM. |
| **CHANGED ITEM*** | Self-explanatory: an ITEM that already existed, but has been altered; subtype of ITEM. |
| **TEST REPORT** | "A document that describes the conduct and results of the testing carried out for a system or component [affected by the change]" (IEEE, 1991). |
| **DOCUMENTATION** | According to the Pennsylvania State University Libraries (2004) definition, DOCUMENTATION is "[p]rinted material which accompanies other materials (usually non-book), and which explains, gives instructions for use, or otherwise functions as a guide to the major materials." In this context, it can also be digital materials or even training, as long as it relates to (pieces of) the system. |
| **SYSTEM RELEASE** | "[M]erchandise issued for sale or public showing" (Princeton University, 2003). Consists of one or more ITEMS and the accompanying DOCUMENTATION. |
| **CHANGE VERIFICATION** | A determination of whether or not the result of the change implementation fulfills the requirements established earlier (Rigby, 2003). |

Besides just ‘changes’, one can also distinguish deviations and waivers. A deviation is an authorization (or a request for it) to depart from a requirement of an item, prior to the creation of it. A waiver is essentially the same, but than during or after creation of the item. These two approaches can be viewed as minimalistic change management (i.e. no real solution to the problem at hand).

## Examples

A good example of the change management process in action can be found in software development. Often users report bugs or desire new functionality from their software programs, which leads to a change request. The product software company then looks into the technical and economical feasibility of implementing this change and consequently it decides whether the change will actually be realized. If that indeed is the case, the change has to be planned, for example through the usage of function points. The actual execution of the change leads to the creation and/or alteration of software code and when this change is propagated it probably causes other code fragments to change as well. After the initial test results seem satisfactory, the documentation can be brought up to date and be released, together with the software. Finally, the project manager verifies the change and closes this entry in the change log.



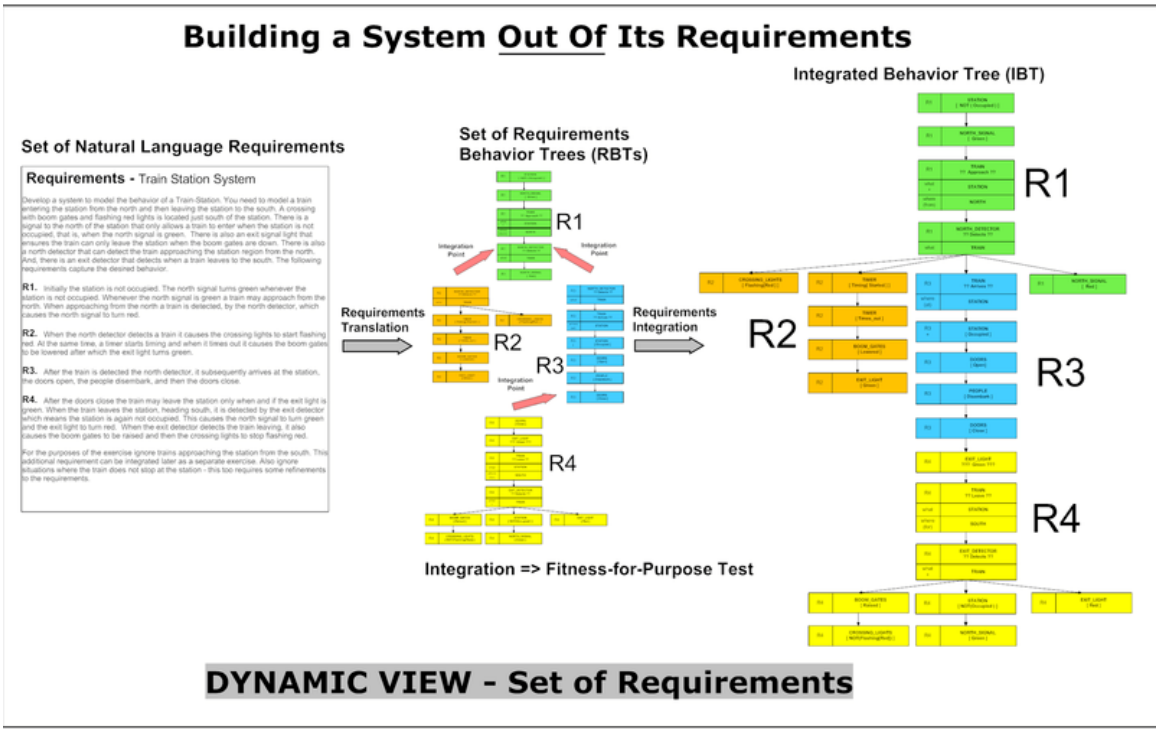
Another typical area for change management in the way it is treated here, is the manufacturing domain. Take for instance the design and production of a car. If for example the vehicle's air bags are found to automatically fill with air after driving long distances, this will without a doubt lead to customer complaints (or hopefully problem reports during the testing phase). In turn, these produce a change request (see Figure 2 on the right), which will probably justify a change. Nevertheless, a – most likely simplistic – cost and benefit analysis has to be done, after which the change request can be approved. Following an analysis of the impact on the car design and production schedules, the planning for the implementation of the change can be created. According to this planning, the change can actually be realized, after which the new version of the car is hopefully thoroughly tested before it is released to the public.

## *In industrial plants*

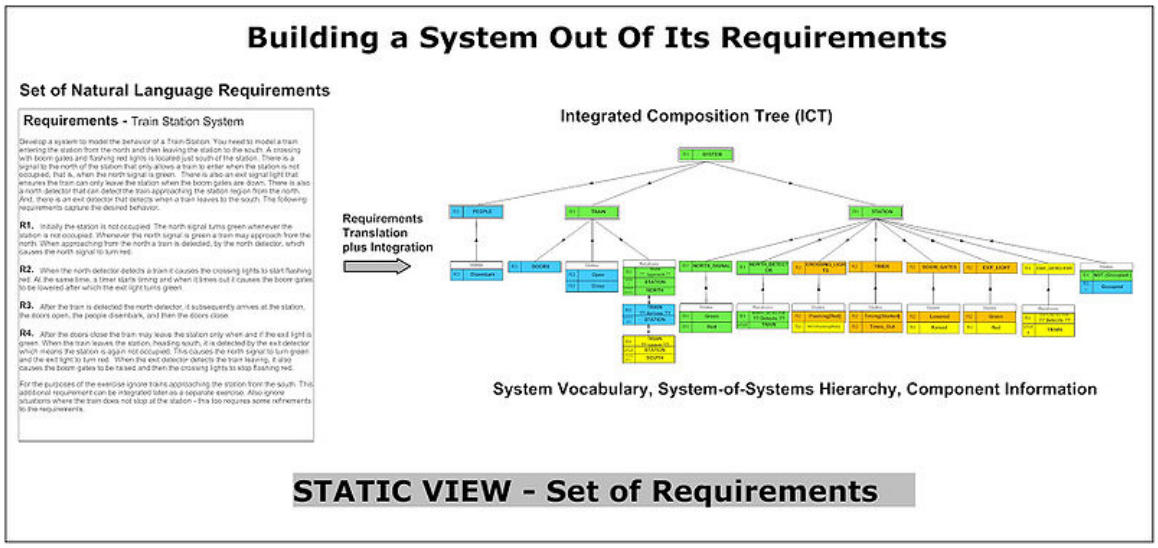
Since complex processes can be very sensitive to even small changes, proper management of change to industrial facilities and processes is recognized as critical to safety. In the US, OSHA has regulations that govern how changes are to be made and documented. The main requirement is that a thorough review of a proposed change be performed by a multi-disciplinary team to ensure that as many possible viewpoints are used to minimize the chances of missing a hazard. In this context, change management is known as Management of Change, or MOC. It is just one of many components of Process Safety Management, section 1910.119(l).1

Chapter 7

# Behavior Trees




Building a System Out of its Requirements - Dynamic View




Building a System Out of its Requirements - Static View

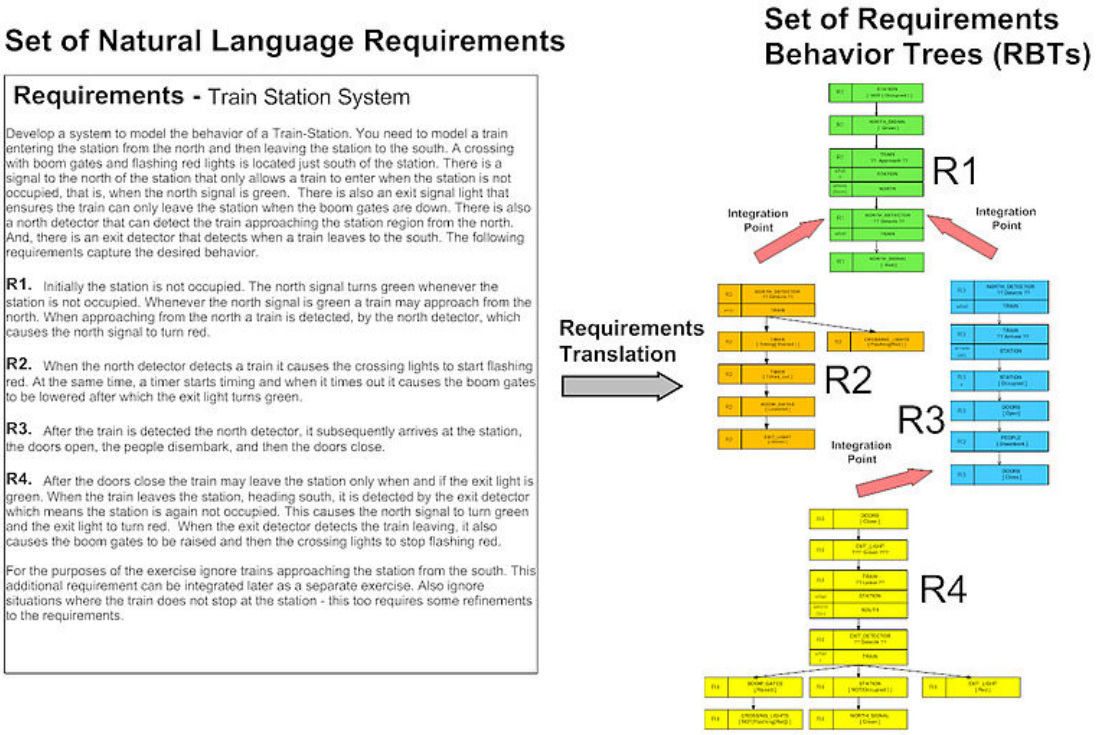
**Behavior Trees** are a formal, graphical modelling language used primarily in systems and software engineering. Behavior trees employ a well defined notation to unambiguously represent the hundreds or even thousands of natural language requirements that are typically used to express the stakeholder needs for a large-scale software-integrated system.

## *Overview*

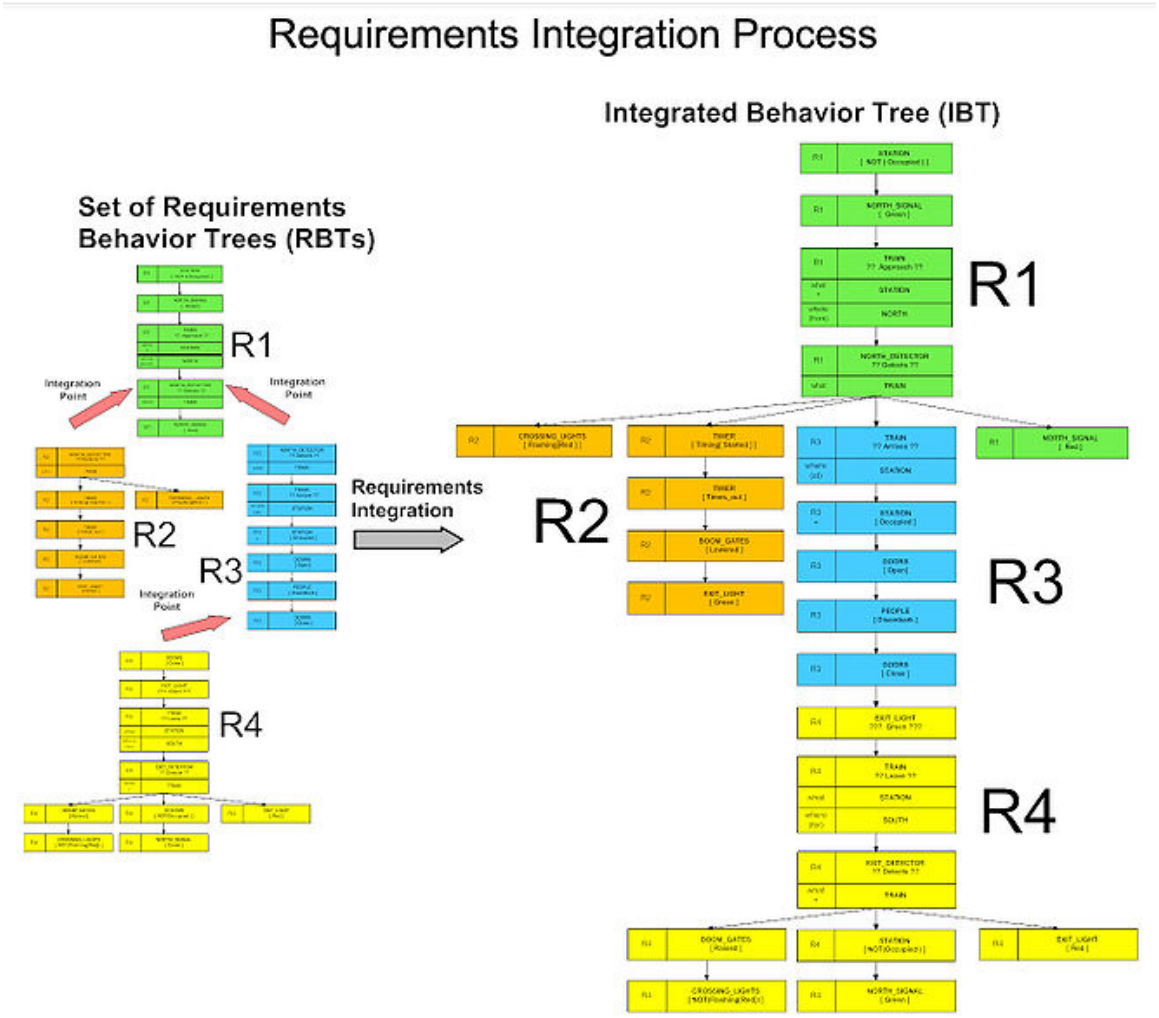
The amount of detail in the large number of natural language requirements for a large-scale system causes short-term memory overload and may create a barrier that prevents anyone from gaining a deep, accurate and holistic understanding of the system needs. Also, because of the use of natural language, there are likely to be many ambiguities, aliases, inconsistencies, redundancies and incompleteness problems associated with the requirements information. This adds further to the uncertainty and complexity. Generally, at best, a few people understand parts of the system or situation well, but no one has other than a superficial understanding of the whole – that is, the detailed integrated behavior of the system.

The Behavior Tree representation, (with the help of the Composition Tree representation that resolves alias and other vocabulary problems with large sets of requirements) allows people to avoid short-term memory overload and produce a deep, accurate, holistic representation of system needs that can be understood by all stakeholders because it strictly uses the vocabulary of the original requirements. Because the Behavior Tree Notation uses a formal semantics, for any given example, it already is, or can be made executable.

## Behavior tree forms




Set of four Requirements Behavior Trees.




Requirements Integration Process

Single and composite or integrated behavior tree forms are both important in the application of behavior trees in systems and software engineering.

- Requirement Behavior Trees : Initially, individual requirement behavior trees (RBTs) are used to capture all the fragments of behavior in each individual natural language requirement by a process of rigorous, intent-preserving and vocabulary-preserving translation. The translation process can uncover a range of defects in original natural language requirements.
- Integrated Behavior Tree : Because a set of requirements imply the integrated behavior of a system, all the individual requirement behavior trees can be composed to construct an integrated behavior tree (IBT) that provides a single holistic view of the emergent integrated behavior of the system. This enables the building of the integrated behavior of a system **out of** its requirements. An analogy to help describe this process is the transition from a randomly arranged set of jigsaw puzzle pieces to putting each of the pieces in its appropriate place.

When we do this, we see each piece of information in its intended context and we see the pieces of information as a whole and the emergent properties of the whole.

Having all the requirements converted to behavior trees (RBTs) is similar to having all the pieces for a jigsaw puzzle randomly spread out on a table - until we put all the pieces together we cannot see the emergent picture and whether any pieces are missing or do not fit. Constructing an Integrated Behavior Tree (IBT) allows us to do this.

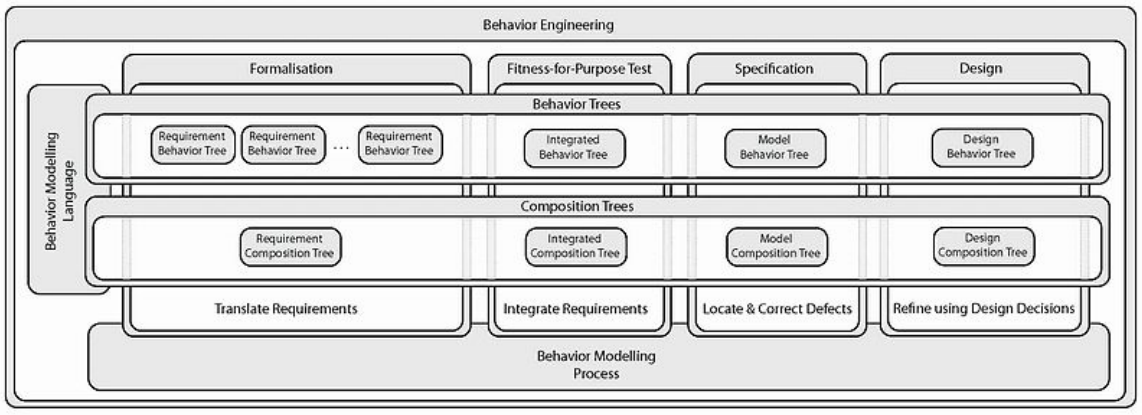
## Behavior engineering process

Representation Used - (critical)

- BEHAVIOR TREES provide a vehicle for growing a shared understanding of a complex system.
- The role of the COMPOSITION TREE in the overall process is to provide a vehicle for overcoming the imperfect knowledge associated with the large set of requirements for a system.

Process Used - (critical)

- BEHAVIOR ENGINEERING uses Behavior Trees to control complexity while growing a shared understanding of a complex system.
- That shared, holistic understanding of a complex system, because it integrates the requirements, shows the emergent behavior of the system implied by requirements.




Phases of the Behavior Modelling Process

## *History*

Behavior Trees and the concepts for their application in systems and software engineering were originally developed by Dromey with first publication of some of the key ideas in 2001. Early publications on this work used the terms “genetic software engineering” and “genetic design” to describe the application of behavior trees. The reason for originally using the word genetic was because sets of genes, sets of jigsaw

puzzle pieces and sets of requirements represented as behavior trees all appeared to share several key properties:

- they contained enough information as a set to allow them to be composed – with behavior trees this allows a system to be built out of its requirements
- the order in which the pieces were put together was not important – with requirements this aids coping with complexity
- when all the members of the set were put together the resulting integrated entity exhibited a set of important emergent properties.

For behavior trees important emergent properties include

- the integrated behavior of the system implied by the requirements
- the coherent behavior of each component referred to in the requirements.

These genetic parallels, in another context, were originally spelled by Woolfson, (A. Woolfson, Living Without Genes, Flamingo, 2000)

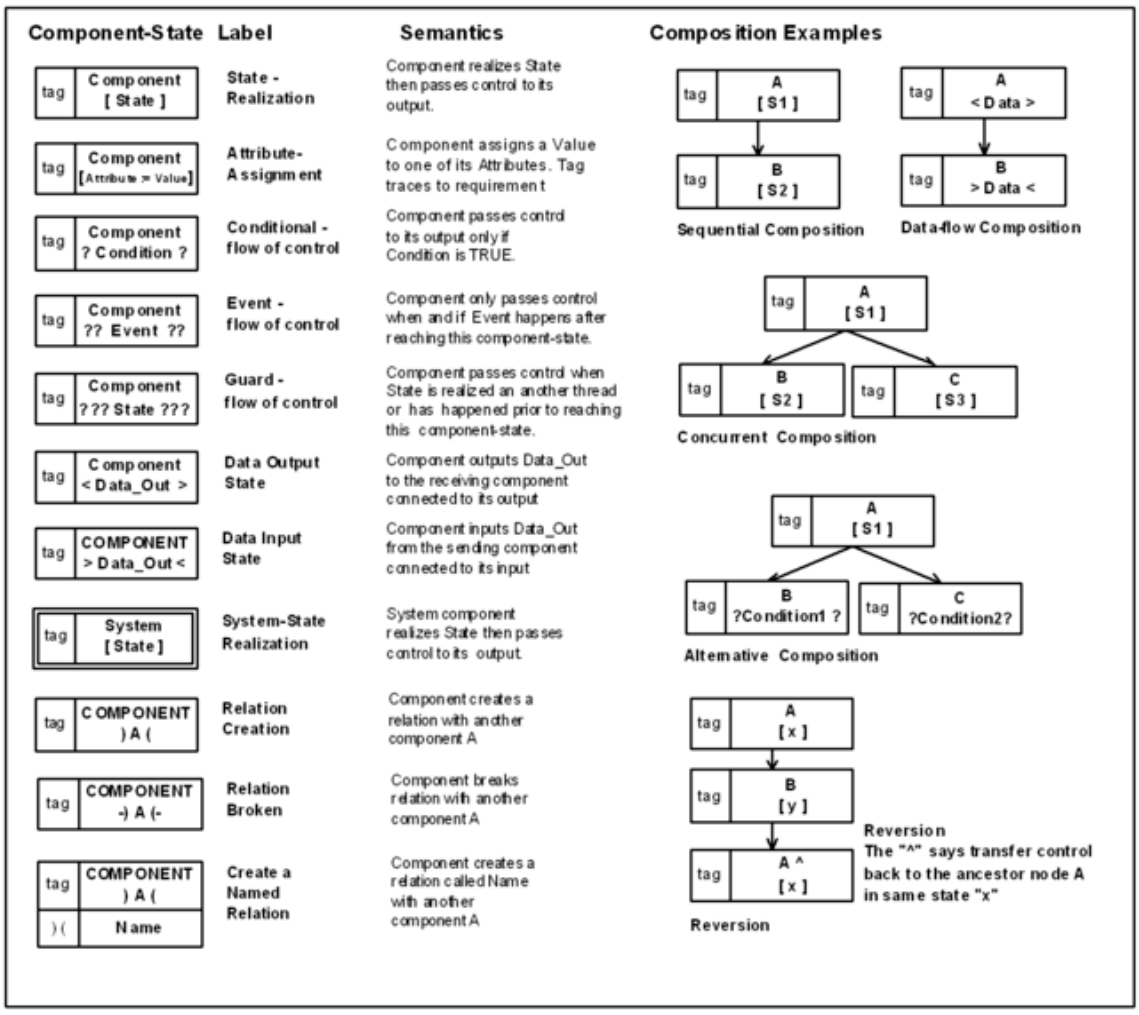
Further weight for use of the term genetic came from eighteenth century thinker Giambattista Vico, who said, “To understand something, and not merely be able to describe it, or analyse it into its component parts, is to understand how it came into being – its genesis, its growth … true understanding is always genetic”. Despite these legitimate genetic parallels it was felt that this emphasis led to confusion with the concept of genetic algorithms. As a result the term Behavior Engineering was introduced to describe the processes that exploit behavior trees to construct systems. The term "behavior engineering" has previously been used in a specialized area of Artificial Intelligence - robotics research. The present use embraces a much broader rigorous formalization and integration of large sets of behavioral and compositional requirements needed to model large-scale systems.

Since the Behavior Tree Notation was originally conceived a number of people from the DCCS (Dependable Complex Computer-based Systems Group – a joint University of Queensland, Griffith University research group) have made important contributions to the evolution and refinement of the notation and to the use of Behavior Trees. Members of this group include: David Carrington, Rob Colvin, Geoff Dromey, Lars Grunske, Ian Hayes, Diana Kirk, Peter Lindsay, Toby Myers, Dan Powell, Cameron Smith, Larry Wen, Nisansala Yatapanage, Kirsten Winter, Saad Zafar, Forest Zheng.

Probabilistic Timed Behavior Trees have recently been developed by Colvin, Grunske and Winter so that reliability, performance and other dependability properties can be expressed.

## *Key concepts*

### Behavior tree notation




Core Elements of the Behavior Tree Notation

A behavior tree is used to formally represent the *fragment of behavior* in each individual requirement. Behavior for a large-scale system in general, where concurrency is admitted, appears abstractly as a set of communicating sequential processes. The Behavior Tree Notation captures these composed component-states in a simple tree-like form.

Behavior is expressed in terms of components realizing states and components creating and breaking relations. Using the logic and graphic forms of conventions found in programming languages, components can support actions, composition, events, control-flow, data-flow, and threads.

Traceability tags (see Section 1.2 of Behavior Tree Notation) in behavior tree nodes link the formal representation to the corresponding natural language requirement. Behavior trees accurately capture behavior expressed in the natural language representation of functional requirements. Requirements Behavior Trees strictly use the vocabulary of the natural language requirements but employ graphical forms for behavior composition in order to eliminate risk of ambiguity. By doing this they provide a direct and clearly traceable relationship between what is expressed in the natural language representation and its formal specification.

A basis of the notation is that behavior is always associated with some component. Component-states which represent nodes of behavior are composed sequentially or concurrently to construct a behavior tree that represents the behavior expressed in the natural language requirements. A behavior tree with leaf nodes may revert back (symbolized by adding the caret operator ^) to an ancestor node to repeat behavior, or start a new thread (symbolized by two carets ^^).

A Behavior Tree specifies state changes in components, how data and control is passed between components and how threads interact. There are constructs for creating and breaking relations. There are also constructs for setting and testing states of components as well as mechanisms for inter-process communication that include message passing (events), shared variable blocking and synchronization.

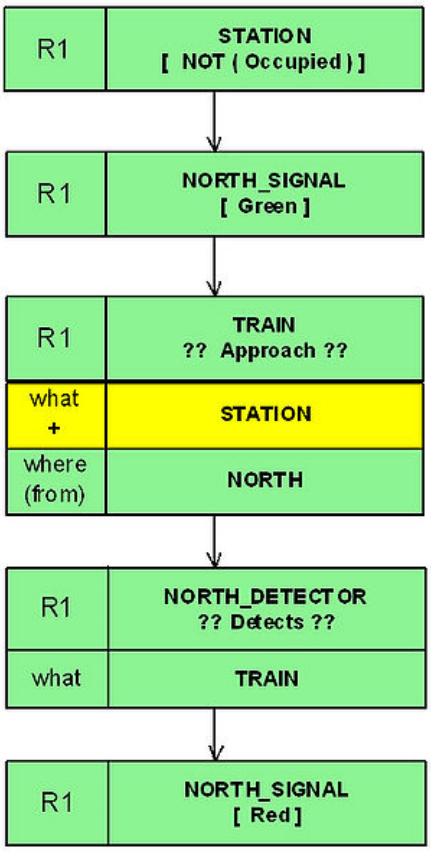
For a complete reference to Behavior Tree notation, version 1.0, see: Behavior Tree Notation v1.0 (2007)

## Semantics

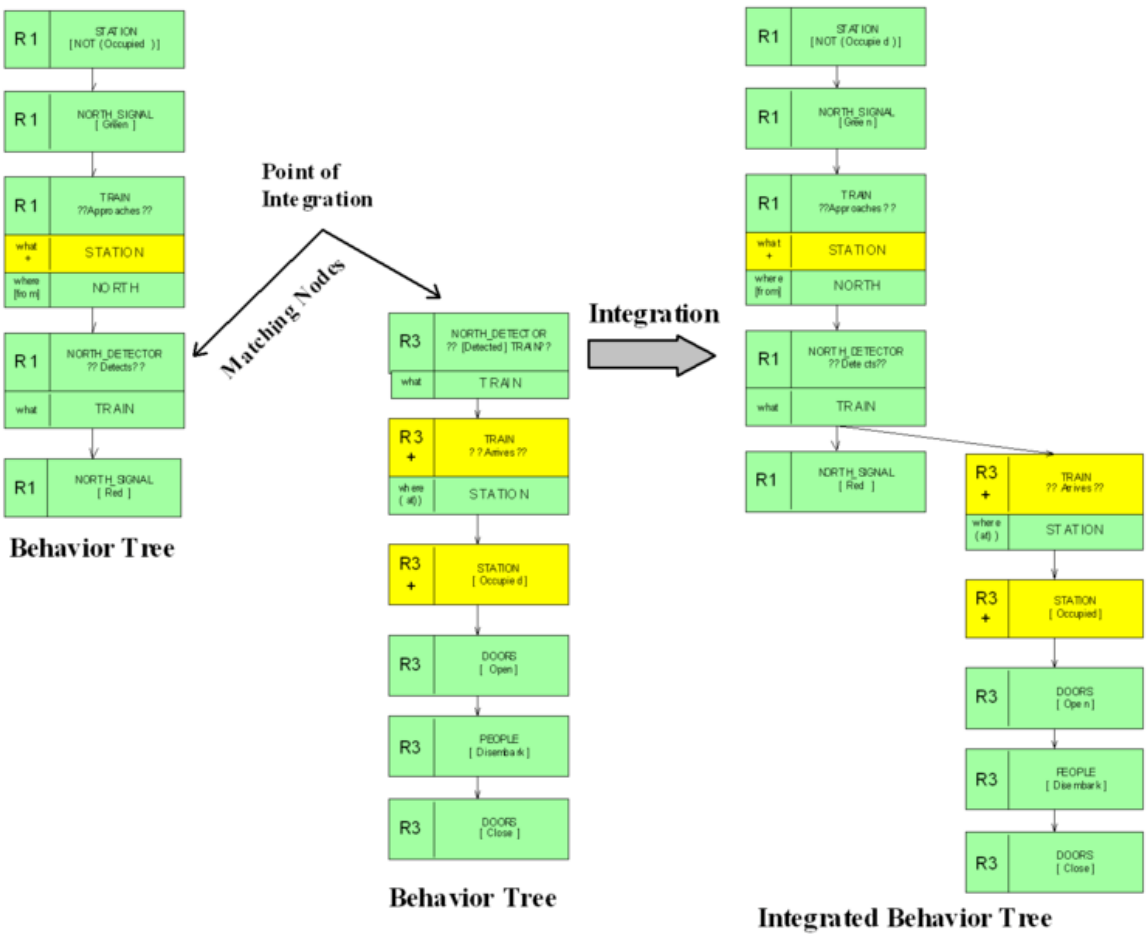
The formal semantics of Behavior Trees is given via a process algebra and its operational semantics. The semantics has been used as the basis for developing simulation, model checking and failure modes and effects analysis.

**Requirements translation**

**Requirement-R1**
*Initially* the **station** is not occupied. The **north signal** turns green *whenever (seq)* the **station** is not occupied. *Whenever* the **north signal** is green *(seq)* a **train** may approach from the north. *When* approaching from the north, *(seq)* a **train** is detected by the **north detector**, which *causes (seq)* the **north_signal** to turn red.




Example Requirement Translation




Requirements Behavior Tree Integration

Requirements translation is the vehicle used to cross the informal-formal barrier. Consider the process of translation for requirement R1 below. The first tasks are to identify the components (**bold**), identify the behaviors (underline) and identify indicators of the order (*italics*) in which behaviors take place. The corresponding behavior tree can then be constructed.

What is clear from the outcome of this process is that apart from pronouns, definite articles etc., essentially all the words in the sentences that contribute to the behavior they describe have been accounted for and used.

## Requirements integration

Once the set of requirements are formalized as individual requirement behavior trees, two joint properties of systems and requirements need to be exploited in order to proceed with composing the integrated behavior tree:

- In general, a fragment of behavior expressed by a requirement always has associated with it a precondition which needs to be satisfied before the behavior

can take place (this precondition may or may not be expressed in the requirement).
- If the requirement is really part of the system then some other requirement in the set must establish the precondition needed in (1).

  *For requirements represented as behavior trees this amounts to finding where the root node of one tree occurs in some other behavior tree and integrating the two trees at that node.*

The example below illustrates requirements integration for two requirements, R1 and R3. In other words, it shows how these two requirements interact.

## Operations on integrated behavior trees

Once an integrated behavior tree has been composed, there are a number of important operations that can be performed upon it.

## Inspection: defect detection and correction

In general, many defects become much more visible when there is an integrated view of the requirements and each requirement has been placed in the behavior context where it needs to execute. For example, it is much easier to tell whether a set of conditions or events emanating from a node is complete and consistent. The traceability tags also make it easy to refer back to the original natural language requirements. There is also the potential to automate a number of defect and consistency checks on an integrated behavior tree.

When all defects have been corrected and the IBT is logically consistent and complete it becomes a Model Behavior Tree (MBT) which serves as a Formal Specification for the System's behavior that has been constructed out the original requirements. This is the clearly defined stopping point for the analysis phase. With other modelling notations and methods (for instance, with UML) it is less clear-cut when modelling can stop. In some cases, parts of a Model Behavior Tree may need to be transformed to make the specification executable. Once an MBT has been made executable it is possible to carry out a number of other dependability checks.

## Simulation

A Model Behavior Tree can be readily simulated in order to explore the dynamic properties of the system. Both a symbolic tool and a graphics tool have been constructed to support these activities.

## Model-checking

A translator has been written to convert a Model Behavior Tree into the "Actions Systems" language. This input can then be fed into the SAL Model-checker in order to allow checks to be made as to whether certain safety and security properties are satisfied.

## Failure Mode and Effects Analysis (FMEA)

Model-checking has often been applied to system models to check that hazardous states cannot be reached during normal operation of the system. It is possible to combine model-checking with behavior trees to provide automated support for failure mode and effects analysis (FMEA). The advantage of using Behavior Trees for this purpose is that they allow the formal method aspects of the approach to be hidden from non-expert users.

## Requirements change

The ideal that is sought when responding to a change in the functional requirements for a system is that it can be quickly determined:

- where to make the change,
- how the change affects the architecture of the existing system,
- which components of the system are affected by the change, and
- what behavioral changes will need to be made to the components (and their interfaces) that are affected by the change of requirements.

Because a system is likely to undergo many sets of changes over its service time, there is also a need to record, manage and optimize the system's evolution driven by the change sequence.

A traceability model, which uses behavior trees as a formal notation to represent functional requirements, reveals change impacts on different types of design constructs (documents) caused by the changes of the requirements. The model introduces the concept of evolutionary design documents that record the change history of the designs. From these documents, any version of a design document as well as the difference between any two versions can be retrieved. An important advantage of this model is that the major part of the procedure to generate these evolutionary design documents can be supported by automated tools.

## Code generation and execution

The Behavior Tree representation of the integrated behavior of the system affords several important advantages as an executable model. It clearly separates the tasks of *component integration* from the task of individual *component implementation*. The integrated behavior of the system that emerges from integrating the requirements can be used as a foundation to create a design by applying design decisions. The result is a Design

Behavior Tree (DBT): an executable multithreaded component integration specification that has been built out of the original requirements.

Behavior Tree models are executed in a virtual machine called the Behavior Run-time Environment (BRE). The BRE links together components using middleware, allowing components to be independent programs written in one of several languages that can be executed in a distributed environment. The BRE also contains an expression parser that automatically performs simple operations to minimize the amount of code required to be manually implemented in the component.

The implementation of components is supported by views that are automatically extractable from the DBT. These views provide the component behavior trees (CBTs) of individual components together with the interfaces of individual components. This information, together with the information in the integrated composition tree (ICT) captured about each individual component, provides the information that is needed to implement each individual component.

Several BRE's can be linked together to form complex systems using a system-of-systems construct and the Behavior Engineering Component Integration Environment (BECIE). BECIE is also used to monitor and control the Behavior Tree models being executed within a BRE, similar to supervisory control and data acquisition (SCADA) systems used in industrial process control.

Executable Behavior Trees have been developed for case studies including automated train protection, mobile robots with dynamic object following, an ambulatory infusion pump and traffic light management systems. A version of the BRE suited for embedded systems (eBRE) is also available that has reduced functionality to tailor it to small-footprint microcontrollers.

## *Applications*

Behavior Tree modelling can and has been applied to a diverse range of applications over a number of years. Some of the main application areas are described below.

### Large-scale systems

Modeling large-scale systems with large sets of natural language requirements has always been the major focus for trialling Behavior Trees and the overall Behavior Engineering process. Conducting these evaluations and trials of the method has involved work with a number of industry partners and government departments in Australia. The systems studied have included a significant number of defense systems, enterprise systems, transportation systems, information systems, health systems and sophisticated control systems with stringent safety requirements. The results of these studies have all been commercial-in-confidence. However the results of the extensive industry trails with Raytheon Australia are presented below in the Industry Section. What all this work has consistently shown is that by translating requirements and creating dynamic and static

integrated views of requirements a very significant number major defects are discovered early, over and above the defects that are found by current industry best-practice.

## Embedded systems

Failure of a design to satisfy a system's requirements can result in schedule and cost overruns. If there are also critical dependability issues, not satisfying system requirements can have life threatening consequences. However in current approaches, ensuring requirements are satisfied is often delayed until late in the development process during a cycle of testing and debugging. This work describes how the system development approach, Behavior Engineering, can be used to develop software for embedded systems. The result is a model-driven development approach that can create embedded system software that satisfies its requirements, as a result of applying the development process.

## Hardware-software systems

Many large-scale systems consist of a mixture of co-dependent software and hardware. The different nature of software and hardware means they are often modelled separately using different approaches. This can subsequently lead to integration problems due to incompatible assumptions about hardware/software interactions. These problems can be overcome by integrating Behavior Trees with the Modelica, mathematical modelling approach. The environment and hardware components are modelled using Modelica and integrated with an executable software model that uses Behavior Trees.

## Role-based access control

To ensure correct implementation of complex access control requirements, it is important that the validated and verified requirements are effectively integrated with the rest of the system. It is also important that the system can be validated and verified early in the development process. An integrated, role-based access control model has been developed. The model is based on the graphical Behavior Tree notation, and can be validated by simulation, as well as verified using a model checker. Using this model, access control requirements can be integrated with the rest of the system from the outset, because: a single notation is used to express both access control and functional requirements; a systematic and incremental approach to constructing a formal Behavior Tree specification can be adopted; and the specification can be simulated and model checked. The effectiveness of the model has been evaluated using a case study with distributed access control requirements.

## Biological systems

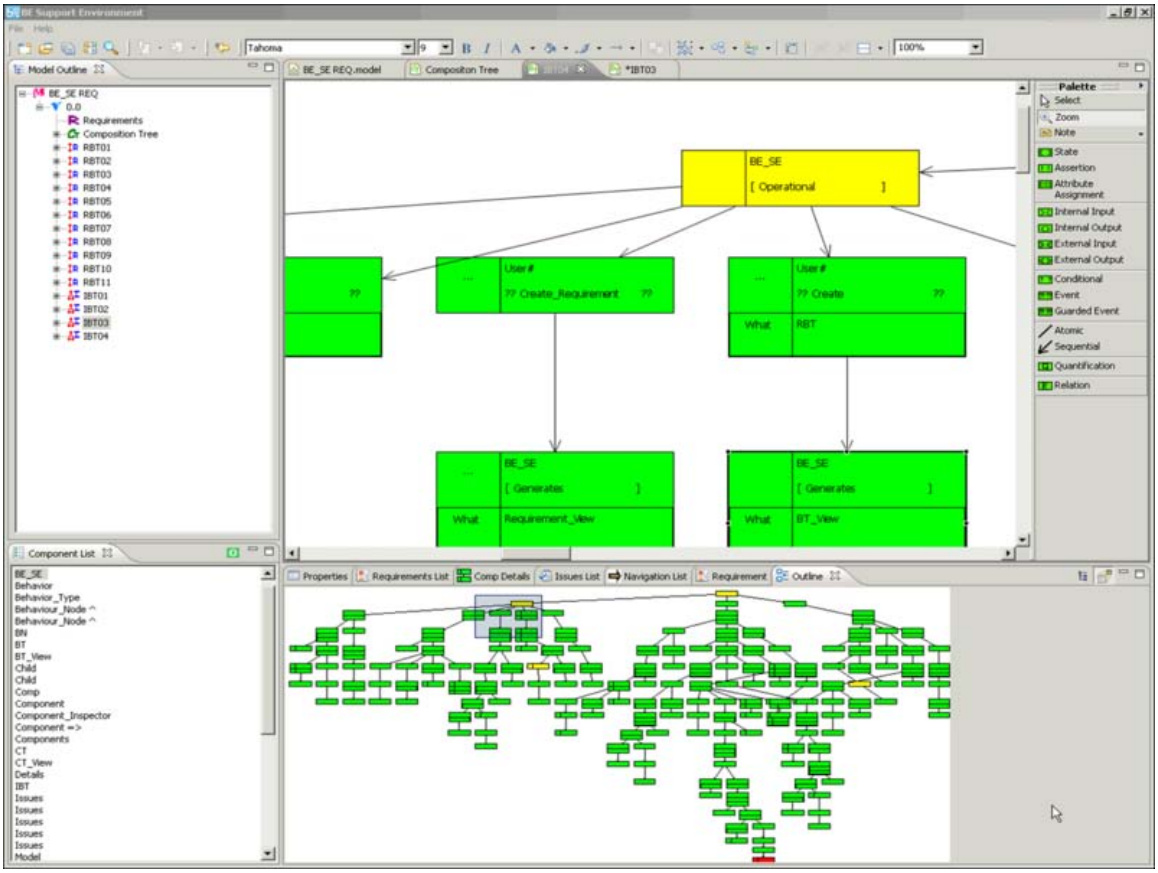
Because Behavior Trees describe complex behavior, they can be used for describing a range of systems not limited to those that are computer-based. In a biological context, BTs can be used to piece together a procedural interpretation of biological functions described in research papers, treating the papers as the requirements documents as

described above. This can help to construct a more concrete description of the process than is possible from reading only, and can also be used as the basis for comparing competing theories in alternative papers. In ongoing research, the Behavior Trees notation is being used to develop models of the brain function in rats under fear conditioning.

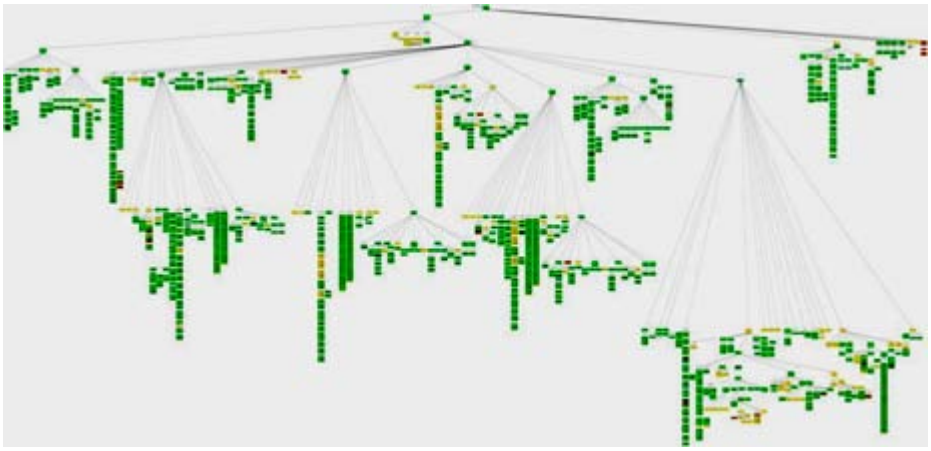
## Game A.I Modeling

While BTs have become popular for modeling the Artificial Intelligence in computer games such as Halo and Spore, these types of trees are very different than the ones described on this page, and are closer to a combination of hierarchical finite state machines or hierarchical task network planners. Soccer-player modeling has also been a successful application of BTs..

## *Scalability and industry applications*




Screen-shot of Behavior Engineering Support Environment Tool



Integrated Behavior Tree - Larger System (more than 1000 requirements)

The first industry trials to test the feasibility of the method and refine its capability were conducted in 2002. Over the last three years a number of systematic industry trials on large-scale defence, transportation and enterprise systems have been conducted. This work has established that the method scales to systems with large numbers of requirements but also that it is important to use tool support in order to efficiently navigate and edit such large integrated views of graphical data. Several main results have come out of this work with industry. On average, over a number of projects, 130 confirmed major defects per 1000 requirements have consistently been found after normal reviews and corrections have been made. With less mature requirements sets much higher defect rates have been observed.

Raytheon Australia supports pioneering systems research

An important part of this work with industry has involved applying the analysis part of the method to six large-scale defence projects for Raytheon Australia. They see the method as “a key risk mitigation strategy, of use in both solution development and as a means of advising the customer on problems with acquisition documentation”. An outcome of these industry trials has been the joint development with Raytheon Australia of an industry-strength tool to support the analysis, editing and display of large integrated sets of requirements. More extensive details of industry findings can be found on the Behavior Engineering website.

Dr Terry Stevenson (Chief Technical Officer, Raytheon Australia) and Mr. Jim Boston (Senior Project Manager Raytheon Australia), Mr. Adrian Pitman from the Australian Defence Materiel Organization, Dr Kelvin Ross (CEO, K.J.Ross & Associates) and Christine Cornish (Bushell & Cornish) have provided the special opportunities needed to support this research and to conduct the industry trials and live project work. This work has been supported by the Australian Research Council – ARC Centre for Complex Systems and funds received from industry.

## *Benefits, advantages*

As a behavior modelling representation, Behavior Trees have a number of significant benefits and advantages:

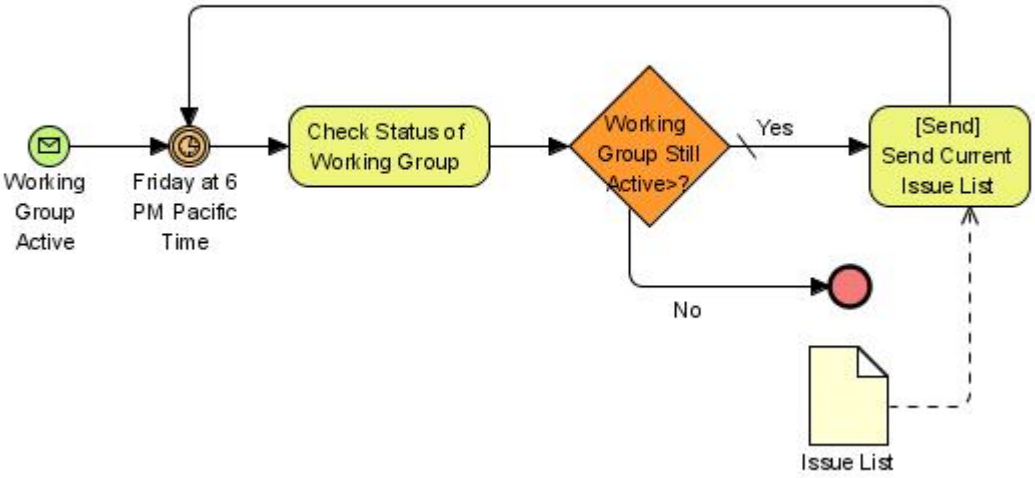
- They employ a well-defined and effective strategy for dealing with requirements complexity, particularly where the initial needs of a system are expressed using hundreds or thousands of requirements written in natural language. This significantly reduces the risk on large-scale projects.
- By rigorously translating then integrating requirements at the earliest possible time they provide a more effective means for uncovering requirements defects than competing methods.
- They employ a single, simple notation for analysis, specification and to represent the behavior design of a system.
- They represent the system behavior as an executable integrated whole.
- They build the behavior of a system out of its functional requirements in a directly traceable way which aids verification and validation.
- They can be understood by stakeholders without the need for formal methods training. By strictly retaining the vocabulary of the original requirements this eases the burden of understanding.
- They have a formal semantics, they support concurrency, they are executable and they can be simulated, model-checked and used to undertake failure mode and effects analysis.
- They can be used equally well to model human processes, to analyse contracts, to represent forensic information, to represent biological systems, and numerous other applications. In each case they deliver the same benefits in terms of managing complexity, and seeing things as a whole. They can also be used for safety critical systems, embedded systems and real-time systems.

## *Criticisms, disadvantages*

- For small textbook level examples, their tree-like nature means that the graphic models produced are sometimes not as compact as Statechart or State Machine behavior specifications.
- The process of requirements translation is very demanding. It is not possible to spend more than three or four hours in a day doing translation even with good tool support.
- Tool support is needed to navigate the very large integrated behavior trees for systems that have hundreds or thousands of requirements.
- For group walkthroughs of very large systems good display facilities are needed.
- There is a need to provide additional sophisticated tool support to fully exploit integrated behavior tree models.

Chapter 8

# Business Process Modeling




Example of business process modeling of a process with a normal flow with the Business Process Modeling Notation.

**Business process modeling** (BPM) in systems engineering and hardware engineering is the activity of representing processes of an enterprise, so that the current process may be analyzed and improved. BPM is typically performed by business analysts and managers who are seeking to improve process efficiency and quality. The process improvements identified by BPM may or may not require Information Technology involvement, although that is a common driver for the need to model a business process, by creating a process master.

Change management programs are typically involved to put the improved business processes into practice. With advances in technology from large platform vendors, the vision of BPM models becoming fully executable (and capable of simulations and round-trip engineering) is coming closer to reality every day.

## *History*

Techniques to model business process such as the flow chart, functional flow block diagram, control flow diagram, Gantt chart, PERT diagram, and IDEF have emerged since the beginning of the 20th century. The Gantt chart were among the first to arrive around 1999, the flow charts in the 1920s, Functional Flow Block Diagram and PERT in the 1987, Data Flow Diagrams and IDEF in the 1970s. Among the modern methods are Unified Modeling Language and Business Process Modeling Notation. Still these represent just a fraction of the methodologies used over the years to document business processes. The term "business process modeling" itself was coined in the 1960s in the field of systems engineering by S. Williams in his 1967 article "Business Process Modeling Improves Administrative Control". His idea was that techniques for obtaining a better understanding of physical control systems could be used in a similar way for business processes. It took until the 1990s before the term became popular.

In the 1990s the term "process" became a new productivity paradigm. Companies were encouraged to think in *processes* instead of *functions* and *procedures*. Process thinking looks at the chain of events in the company from purchase to supply, from order retrieval to sales etc. The traditional modeling tools were developed to picture time and costs, while modern methods focus on cross-function activities. These cross-functional activities have increased severely in number and importance due to the growth of complexity and dependencies. New methodologies such as business process redesign, business process innovation, business process management, integrated business planning among others all "aiming at improving processes across the traditional functions that comprise a company".

In the field of software engineering the term "business process modeling" opposed the common software process modeling, aiming to focus more on the state of the practice during software development. In that time early 1990s all existing and new modeling techniques to picture business processes were considered and called "business process modeling languages." In the Object Oriented approach, it was considered to be an essential step in the specification of Business Application Systems. Business process modeling became the base of new methodologies, that for example also supported data collection, data flow analysis, process flow diagrams and reporting facilities. Around 1995 the first visually oriented tools for business process modeling and implementation were being presented.

## *BPM topics*

### Business model

A business model is a framework for creating economic, social, and/or other forms of value. The term 'business model' is thus used for a broad range of informal and formal descriptions to represent core aspects of a business, including purpose, offerings, strategies, infrastructure, organizational structures, trading practices, and operational processes and policies.

In the most basic sense, a business model is the method of doing business by which a company can sustain itself. That is, generate revenue. The business model spells-out how a company makes money by specifying where it is positioned in the value chain.

## Business process

A business process is a collection of related, structured activities or tasks that produce a specific service or product (serve a particular goal) for a particular customer or customers. There are three main types of business processes:

1. Management processes, the processes that govern the operation of a system. Typical management processes include "Corporate Governance" and "Strategic Management".
2. Operational processes, processes that constitute the core business and create the primary value stream. Typical operational processes are Purchasing, Manufacturing, Marketing, and Sales.
3. Supporting processes, which support the core processes. Examples include Accounting, Recruitment, Technical support.

A business process can be decomposed into several sub-processes, which have their own attributes, but also contribute to achieving the goal of the super-process. The analysis of business processes typically includes the mapping of processes and sub-processes down to activity level. A business process model is a model of one or more business processes, and defines the ways in which operations are carried out to accomplish the intended objectives of an organization. Such a model remains an abstraction and depends on the intended use of the model. It can describe the workflow or the integration between business processes. It can be constructed in multiple levels.

A workflow is a depiction of a sequence of operations, declared as work of a person, work of a simple or complex mechanism, work of a group of persons, work of an organization of staff, or machines. Workflow may be seen as any abstraction of real work, segregated in workshare, work split or whatever types of ordering. For control purposes, workflow may be a view on real work under a chosen aspect.

## Artifact-centric Business process

The Artifact-centric business process model has emerged as a new promising approach for modeling business processes, as it provides a highly flexible solution to capture operational specifications of business processes. It particularly focuses on describing the data of business processes, known as “artifacts”, by characterizing business-relevant data objects, their lifecycles, and related services. The artifact-centric process modelling approach fosters the automation of the business operations and supports the flexibility of the workflow enactment and evolution

## *Business process modeling tools*

Business process modeling tools provide business users with the ability to model their business processes, implement and execute those models, and refine the models based on as-executed data. As a result, business process modeling tools can provide transparency into business processes, as well as the centralization of corporate business process models and execution metrics.

### Modeling and simulation

Modeling and simulation functionality allows for pre-execution "what-if" modeling and simulation. Post-execution optimization is available based on the analysis of actual as-performed metrics.

Business process modeling diagrams are:

- Use case diagrams created by Ivar Jacobson, 1992. Currently integrated in UML
- Activity diagrams, also currently adopted by UML

Some business process modeling techniques are:

- Business Process Modeling Notation (BPMN)
- Cognition enhanced Natural language Information Analysis Method (CogNIAM)
- Extended Business Modeling Language (xBML)
- Event-driven process chain (EPC)
- ICAM DEFinition (IDEF0)
- Unified Modeling Language (UML), extensions for business process such as Eriksson-Penker's
- Riva method using Role Activity Diagrams (RAD)

### Programming languages tools for BPM

BPM suite software provides programming interfaces (web services, application program interfaces (APIs)) which allow enterprise applications to be built to leverage the BPM engine. This component is often referenced as the *engine* of the BPM suite.

Programming languages that are being introduced for BPM include: Some standards:

- BPMN
- Business Process Execution Language (BPEL),
- Web Services Choreography Description Language (WS-CDL).
- XML Process Definition Language (XPDL),

Some vendor-specific languages:

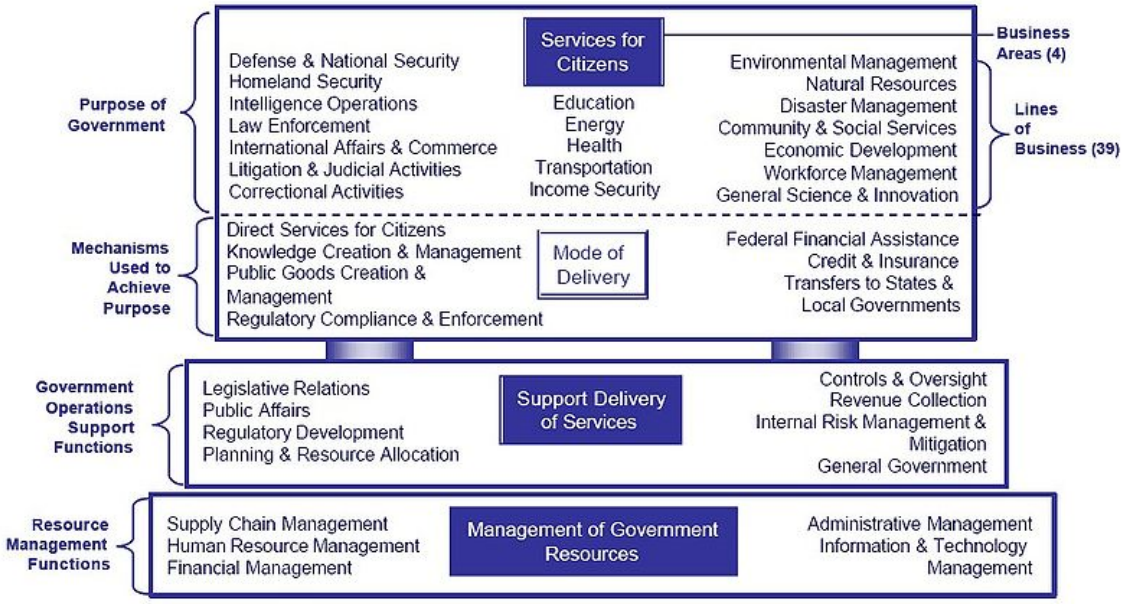
- Architecture of Integrated Information Systems (ARIS) supports EPC,

- Java Process Definition Language (JBPM),

Other technologies related to business process modeling include model-driven architecture and service-oriented architecture.

## *Related topics*

### Business reference model

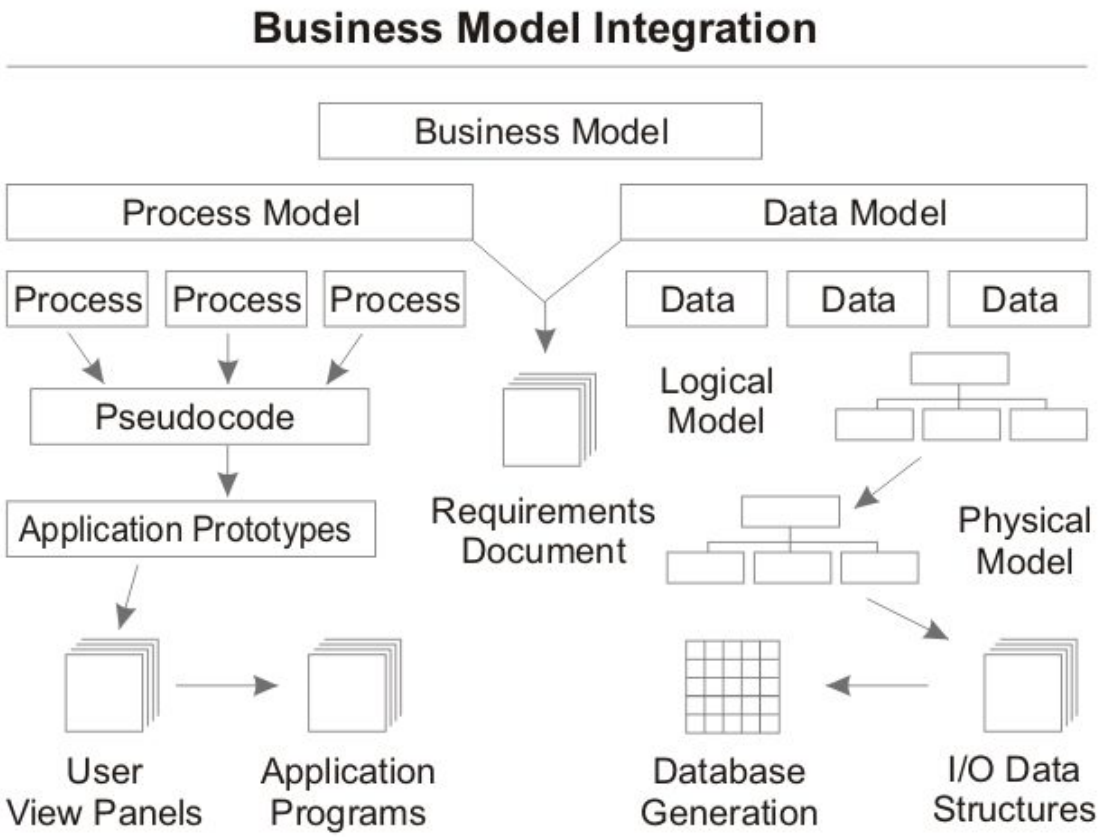



Example of the US Federal Government Business Reference Model.

A business reference model is a reference model, concentrating on the functional and organizational aspects of an enterprise, service organization or government agency. In general a reference model is a model of something that embodies the basic goal or idea of something and can then be looked at as a reference for various purposes. A business reference model is a means to describe the business operations of an organization, independent of the organizational structure that perform them. Other types of business reference model can also depict the relationship between the business processes, business functions, and the business area's business reference model. These reference models can be constructed in layers, and offer a foundation for the analysis of service components, technology, data, and performance.

The most familiar business reference model is the Business Reference Model of the US Federal Government. That model is a function-driven framework for describing the business operations of the Federal Government independent of the agencies that perform them. The Business Reference Model provides an organized, hierarchical construct for describing the day-to-day business operations of the Federal government. While many models exist for describing organizations - organizational charts, location maps, etc. - this model presents the business using a functionally driven approach.

## Business process integration




Example of the interaction between business process and data models.

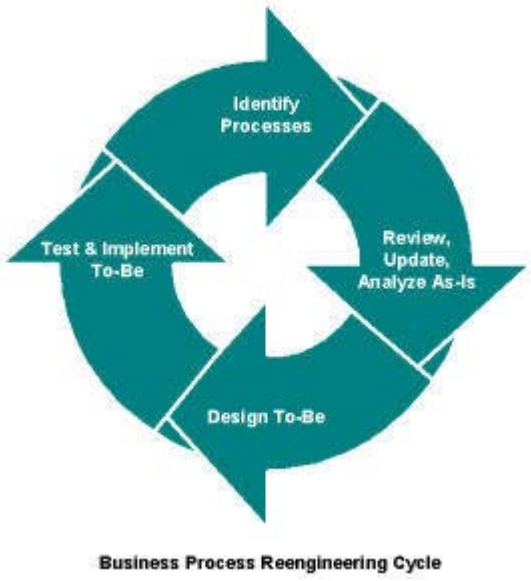
A business model, which may be considered an elaboration of a business process model, typically shows business data and business organizations as well as business processes. By showing business processes and their information flows a business model allows business stakeholders to define, understand, and validate their business enterprise. The data model part of the business model shows how business information is stored, which is useful for developing software code. See the figure on the right for an example of the interaction between business process models and data models.

Usually a business model is created after conducting an interview, which is part of the business analysis process. The interview consists of a facilitator asking a series of questions to extract information about the subject business process. The interviewer is referred to as a facilitator to emphasize that it is the participants, not the facilitator, who provide the business process information. Although the facilitator should have some knowledge of the subject business process, but this is not as important as her mastery of a pragmatic and rigorous method interviewing business experts. The method is important because for most enterprises a team of facilitators is needed to collect information across

the enterprise, and the findings of all the interviewers must be compiled and integrated once completed.

Business models are developed as defining either the current state of the process, in which case the final product is called the "as is" snapshot model, or a concept of what the process should become, resulting in a "to be" model. By comparing and contrasting "as is" and "to be" models the business analysts can determine if the existing business processes and information systems are sound and only need minor modifications, or if reengineering is required to correct problems or improve efficiency. Consequently, business process modeling and subsequent analysis can be used to fundamentally reshape the way an enterprise conducts its operations.

## Business process reengineering




Business Process Reengineering Cycle.

Business process reengineering (BPR) is an approach aiming at improvements by means of elevating efficiency and effectiveness of the processes that exist within and across organizations. The key to business process reengineering is for organizations to look at their business processes from a "clean slate" perspective and determine how they can best construct these processes to improve how they conduct business.

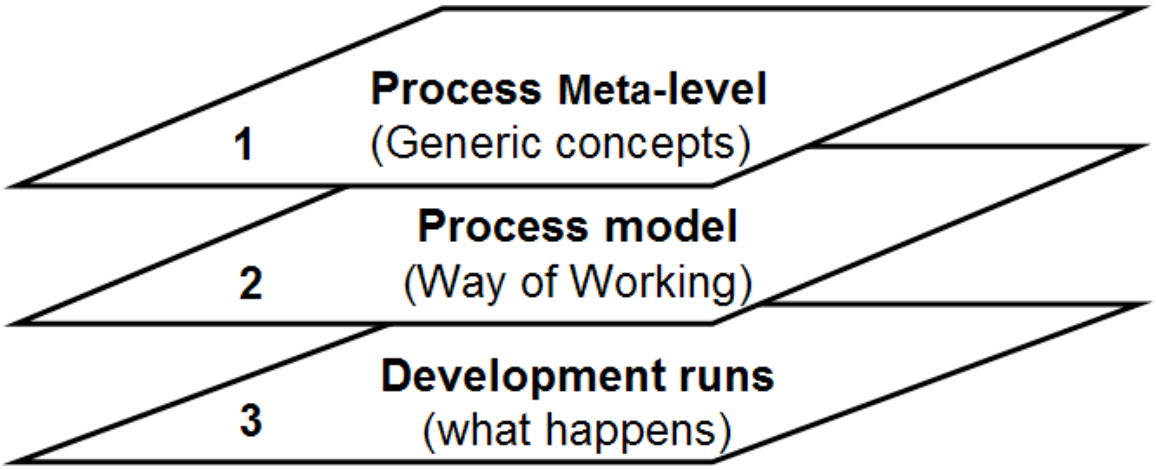
Business process reengineering (BPR) began as a private sector technique to help organizations fundamentally rethink how they do their work in order to dramatically improve customer service, cut operational costs, and become world-class competitors. A key stimulus for reengineering has been the continuing development and deployment of sophisticated information systems and networks. Leading organizations are becoming bolder in using this technology to support innovative business processes, rather than refining current ways of doing work.

## Business process management

Business process management is a field of management focused on aligning organizations with the wants and needs of clients. It is a holistic management approach that promotes business effectiveness and efficiency while striving for innovation, flexibility and integration with technology. As organizations strive for attainment of their objectives, business process management attempts to continuously improve processes - the process to define, measure and improve your processes – a "process optimization" process.

Chapter 9

# Meta-Process Modeling




Abstraction level for processes.

**Meta-process modeling** is a type of metamodeling used in software engineering and systems engineering for the analysis and construction of models applicable and useful to some predefined problems.

Meta-process modeling supports the effort of creating flexible process models. The purpose of process models is to document and communicate processes and to enhance the reuse of processes. Thus, processes can be better taught and executed. Results of using meta-process models are an increased productivity of process engineers and an improved quality of the models they produce.

## *Overview*

Meta-process modeling focuses on and supports the process of constructing process models. Its main concern is to improve process models and to make them evolve, which in turn, will support the development of systems. This is important due to the fact that "processes change with time and so do the Process Models underlying them. Thus, new processes and models may have to be built and existing ones improved". "The focus has been to increase the level of formality of process models in order to make possible their enactment in process-centred software environments". referring to:

A process meta-model is a meta model, "a description at the type level of a process model. A process model is, thus, an instantiation of a process meta-model. [..] A meta-model can be instantiated several times in order to define various process models. A process meta-model is at the meta-type level with respect to a process."
There exist standards for several domains:

- Software Engineering
- Software Process Engineering Metamodel (SPEM) which is defined as a Profile (UML) by the Object Management Group.

## *Topics in metadata modeling*

There are different techniques for constructing process models. "Construction techniques used in the Information Systems area have developed independently of those in Software Engineering. In information systems, construction techniques exploit the notion of a meta-model and the two principal techniques used are those of *instantiation* and *assembly*. In software engineering the main construction technique used today is language-based. However, early techniques in both, information systems and software engineering were based on the experience of process engineers and were, therefore, *ad-hoc* in nature."

### Ad-hoc

"Traditional process models are expressions of the experiences of their developers. Since this experience is not formalised and is, consequently, not available as a fund of knowledge, it can be said that these process models are the result of an ad-hoc construction technique. This has two major consequences: it is not possible to know how these process models were generated, and they become dependent on the domain of experience. If process models are to be domain independent and if they are to be rapidly generable and modifiable, then we need to go away from experience based process model construction. Clearly, generation and modifiability relate to the process management policy adopted (see Usage World). Instantiation and assembly, by promoting modularization, facilitate the capitalisation of good practice and the improvement of given process models."

## Assembly

The assembly technique is based on the idea of a process repository from which process components can be selected. Rolland 1998 lists two selection strategies:

1. Promoting a *global* analysis of the project on hand based on contingency criteria (Example Van Slooten 1996)
2. Using the notion of descriptors as a means to describe process chunks. This eases the retrieval of components meeting the requirements of the user / matching with the situation at hand.

(Example Plihon 1995 in NATURE () and repository of scenario based approaches accessible on Internet in the CREWS project )

For the assembly technique to be successful, it is necessary that process models are modular. If the assembly technique is combined with the instantiation technique then the meta-model must itself be modular.

## Instantiation

For reusing processes a meta-process model identifies "the common, generic features of process models and represents them in a system of concepts. Such a representation has the potential to 'generate' all process models that share these features. This potential is realised when a generation technique is defined whose application results in the desired process model."

Process models are then derived from the process meta-models through *instantiation*. Rolland associates a number of advantages with the instantiation approach:

1. The exploitation of the meta-model helps to define a wide range of process models.
2. It makes the activity of defining process models systematic and versatile.
3. It forces to look for and introduce, in the process meta-model, generic solutions to problems and this makes the derived process models inherit the solution characteristics.

"The instantiation technique has been used, for example, in NATURE, Rolland 1993, Rolland 1994, and Rolland 1996. The process engineer must define the instances of contexts and relationships that comprise the process model of interest."

## Language

Rolland 1998 lists numerous languages for expressing process models used by the software engineering community:

- E3

- Various Prolog dialects for EPOS , Oikos , and PEACE
- PS-Algol for PWI )

as well as further computational paradigms:

- Petri nets in EPOS and SPADE
- Rule based paradigm in MERLIN
- ALF
- Marvel
- EPOS
- Triggers in ADELE and MVP-L ).

Languages are typically related to process programs whereas instantiation techniques have been used to construct process scripts.

## Tool support

The Meta-modeling process is often supported through software tools, called CAME tools (Computer Aided Method Engineering) or Meta-CASE tools (Computer Assisted Software Engineering tools on a Meta-level). Often the instantiation technique "has been utilised to build the repository of Computer Aided Method Engineering environments" (referring to ).

Example tools for meta-process modeling are:

- Maestro II
- MetaEdit+
- Mentor

## *Example: "Multi-model view"*

Colette Rolland (1999) provides an example of a meta-process model which utilizes the instantiation and assembly technique. In the paper the approach is called "Multi-model view" and was applied on the CREWS-L'Ecritoire method. The CREWS-L'Ecritoire method represents a methodical approach for Requirements Engineering, "the part of the IS development that involves investigating problems and requirements of the users community and developing a specification of the future system, the so-called conceptual schema.".

Besides the CREWS-L'Ecritoire approach, the multi-model view has served as a basis for representing :

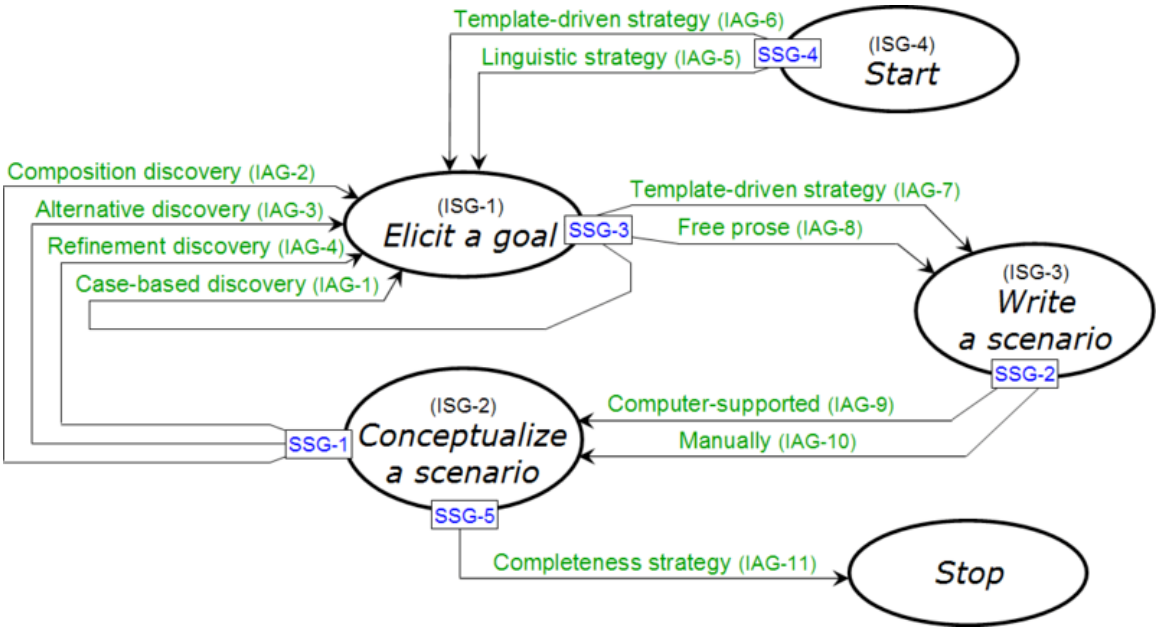
> (a) the three other requirements engineering approaches developed within the CREWS project, Real World Scenes approach , SAVRE approach for scenario exceptions discovery , and the scenario animation approach
> (b) for integrating approaches one with the other and with the OOSE approach

Furthermore, the CREWS-L'Ecritoire utilizes Process Models and Meta-Process Models in order to achieve flexibility for the situation at hand. The approach is based on the notion of a labelled graph of intentions and strategies called a *map* as well as its associated *guidelines* . Together, map (process model) and the guidelines form the method. The main source of this explanation is the elaboration of Colette Rolland in .

## Process model / Map

The map is "a navigational structure which supports the dynamic selection of the intention to be achieved next and the appropriate strategy to achieve it"; it is "a process model in which a nondeterministic ordering of intentions and strategies has been included. It is a labelled directed graph with intentions as nodes and strategies as edges between intentions. The directed nature of the graph shows which intentions can follow which one."

The map of the CREWS-L'Ecritoire method looks as follow:




Process model of the CREWS-L'Ecritoire method

The map consists of goals / *intentions* (marked with ovals) which are connected by *strategies* (symbolized through arrows). An *intention* is a goal, an objective that the application engineer has in mind at a given point of time. A *strategy* is an approach, a manner to achieve an intention. The connection of two goals with a strategy is also called *section*.

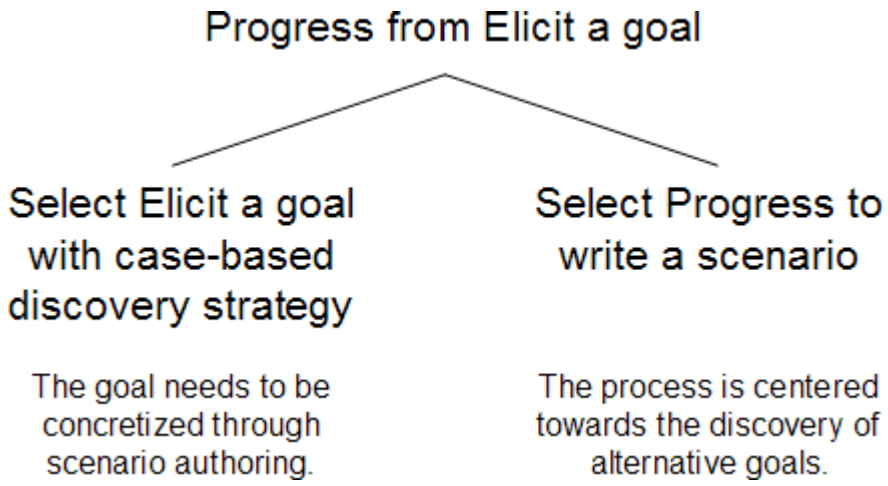
A map "allows the application engineer to determine a path from Start intention to Stop intention. The map contains a finite number of paths, each of them prescribing a way to develop the product, i.e. each of them is a process model. Therefore the map is a multi-

model. It embodies several process models, providing a multi-model view for modeling a class of processes. None of the finite set of models included in the map is recommended 'a priori'. Instead the approach suggests a dynamic construction of the actual path by navigating in the map. In this sense the approach is sensitive to the specific situations as they arise in the process. The next intention and strategy to achieve it are selected dynamically by the application engineer among the several possible ones offered by the map. Furthermore, the approach is meant to allow the dynamic adjunction of a path in the map, i.e. adding a new strategy or a new section in the actual course of the process. In such a case guidelines that make available all choices open to handle a given situation are of great convenience. The map is associated to such guidelines" .

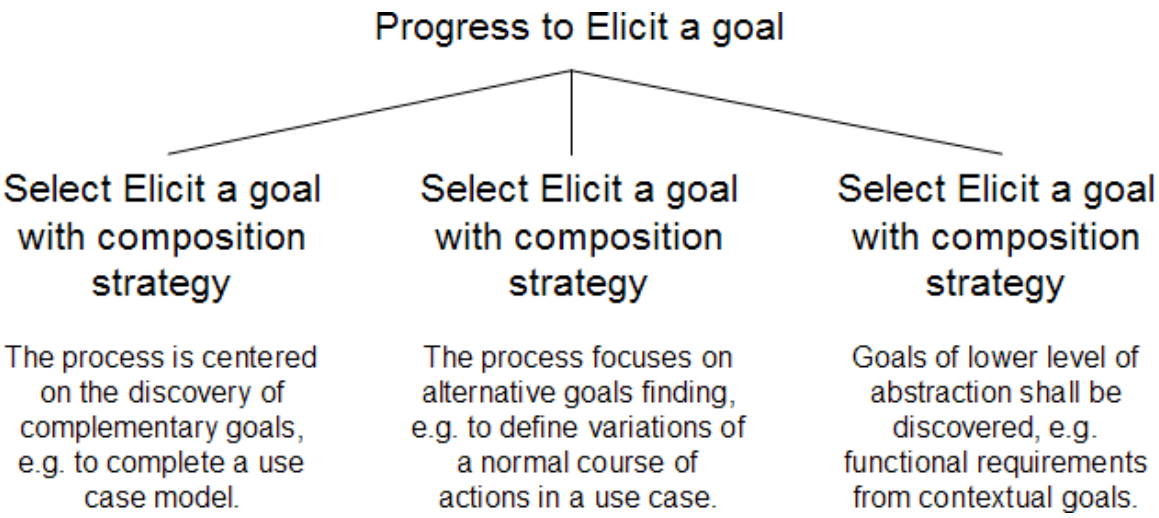
## Guidelines

A guideline "helps in the operationalisation of the selected intention" ; it is "a set of indications on how to proceed to achieve an objective or perform an activity." The description of the guidelines is based on the NATURE project's contextual approach and its corresponding enactment mechanism. Three types of guidelines can be distinguished:

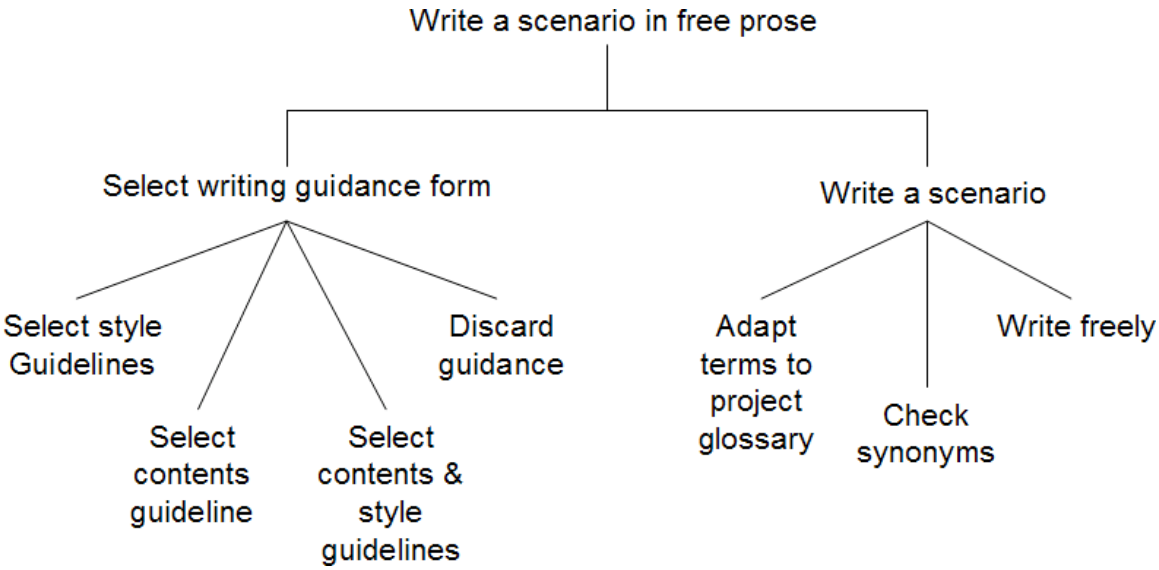
- *Intention Selection Guidelines (ISG)* identify the set of intentions that can be achieved in the next step and selects the corresponding set of either IAGs (only one choice for an intention) or SSGs (several possible intentions).
- *Strategy Selection Guidelines (SSG)* guide the selection of a strategy, thereby leading to the selection of the corresponding IAG.
- *Intention Achievement Guidelines (IAG)* aim at supporting the application engineer in the achievement of an intention according to a strategy, are concerned with the tactics to implement these strategies, might offer several tactics, and thus may contain alternative operational ways to fulfil the intention.




Example of an Intention Selection Guideline 1 (ISG-1)




Example of an Strategy Selection Guideline 1 (SSG-1)




Example of an Intention Achievement Guideline 8 (IAG-8)

In our case, the following guidelines – which correspond with the map displayed above – need to be defined:

Intention Selection Guidelines (ISG)

1. ISG-1 Progress from Elicit a goal
2. ISG-2 Progress from Conceptualize a Scenario
3. ISG-3 Progress from Write a scenario
4. ISG-4 Progress from Start

Strategy Selection Guidelines (SSG)

1. SSG-1 Progress to Elicit a goal
2. SSG-2 Progress to Conceptualize a Scenario
3. SSG-3 Progress to Write a scenario
4. SSG-4 Progress to Elicit a goal
5. SSG-5 Progress to Stop

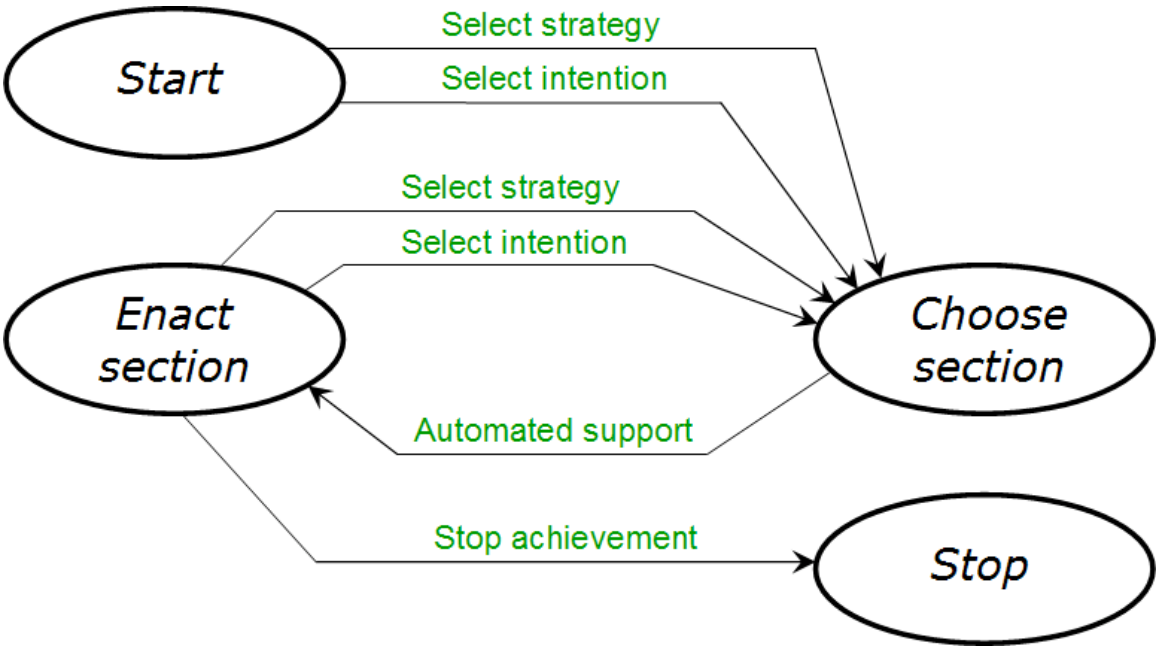
Intention Achievement Guidelines (IAG)

1. IAG-1 Elicit a goal with case-based strategy
2. IAG-2 Elicit a goal with composition strategy
3. IAG-3 Elicit a goal with alternative strategy
4. IAG-4 Elicit a goal with refinement strategy
5. IAG-5 Elicit a goal with linguistic strategy
6. IAG-6 Elicit a goal with template-driven strategy
7. IAG-7 Write a scenario with template-driven strategy
8. IAG-8 Write a scenario in free prose
9. IAG-9 Conceptualize a Scenario with computer support strategy
10. IAG-10 Conceptualize a Scenario manually
11. IAG-11 Stop with completeness strategy

The following graph displays the details for the Intention Achievement Guideline 8 (IAG-8).

## Meta-process map

In the multi-model view as presented in the paper of C. Rolland, the meta-process (the instance of the meta-process model) is "a process for the generation of a path from the map and its instantaneous enactment for the application at hand." While the meta-process model can be represented in many different ways, a map was chosen again as a means to do so. It is not to be mixed up with the map for the process model as presented above.




Meta-process model of the CREWS-L'Ecritoire method

Colette Rolland describes the meta-model as follow : (Meta-intentions are in bold, meta-strategies in italic – in green in the map).

"The **Start** meta-intention starts the construction of a process by selecting a section in the method map which has map intention Start as source. The **Choose Section** meta-intention results in the selection of a method map section. The **Enact Section** meta-intention causes the execution of the method map section resulting from **Choose Section**. Finally, the **Stop** meta-intention stops the construction of the application process. This happens when the **Enact Section** meta-intention leads to the enactment of the method map section having Stop as the target. As already explained in the previous sections, there are two ways in which a section of a method map can be selected, namely by selecting an intention or by selecting a strategy. Therefore, the meta-intention **Choose Section** has two meta-strategies associated with it, *select intention* and *select strategy* respectively. Once a method map section has been selected by **Choose Section**, the IAG to support its enactment must be retrieved; this is represented in [the graph] by associating the meta-strategy *automated support* with the meta-intention, **Enact Section**."

## *Sample process*

The sample process "Eliciting requirements of a Recycling Machine" is about a method for designing the requirements of recycling facilities. The recycling facilities are meant for customers of a supermarket. The adequate method is obtained though instantiation of the meta-process model on the process model.

The following table displays the stepwise trace of the process to elicit requirements for the recycling machine (from ):

| Step | Guideline | Meta-process | Process | Product (Goal = Gxx) |
|---|---|---|---|---|
| 1.1 | SSG-4 | Choose section with select strategy | SSG4 suggests two strategies. The template-driven strategy is chosen because it is the most appropriate way to become familiar with the goal formalisation proposed by the CREWS-L'Ecritoire method | |
| 1.2 | IAG-6 | Enact section with automated support | IAG6 displays a goal statement template and explains the meaning of each parameter. The requirement Engineer (RE) chooses a loose statement having only a verb and a target | G1: Provideverb (Recycling Facilities*) target *RF |
| 2.1 | ISG-1 | Choose section with select intention | ISG1 provides RE with arguments to advise him on choosing one of the two possible intentions from **'Elicit a Goal'**, namely to **'Elicit a Goal or to '**Write a Scenario'**. The former is selected so as to generate alternative design solutions** | |
| 2.2 | IAG-1 | Enact section with automated support | IAG1 uses the goal statement structure and parameter values supplied to generate alternative goals. This leads to 21 alternative goals to G1 which are ORed to G1. After discussion with stakeholders, G4 is selected | G2: Provide bottle RF to our customers with a card-based machine; G3: Provide paper RF to our customers with a card-based machine; G4: Provide bottle and box RR to our customers with a card-based machine; . . . G22: Provide bottle RF to all customers with money return machine |
| 3.1 | SSG-3 | Choose section with select | SSG3 offers two strategies from which the template-driven strategy is chosen. | |

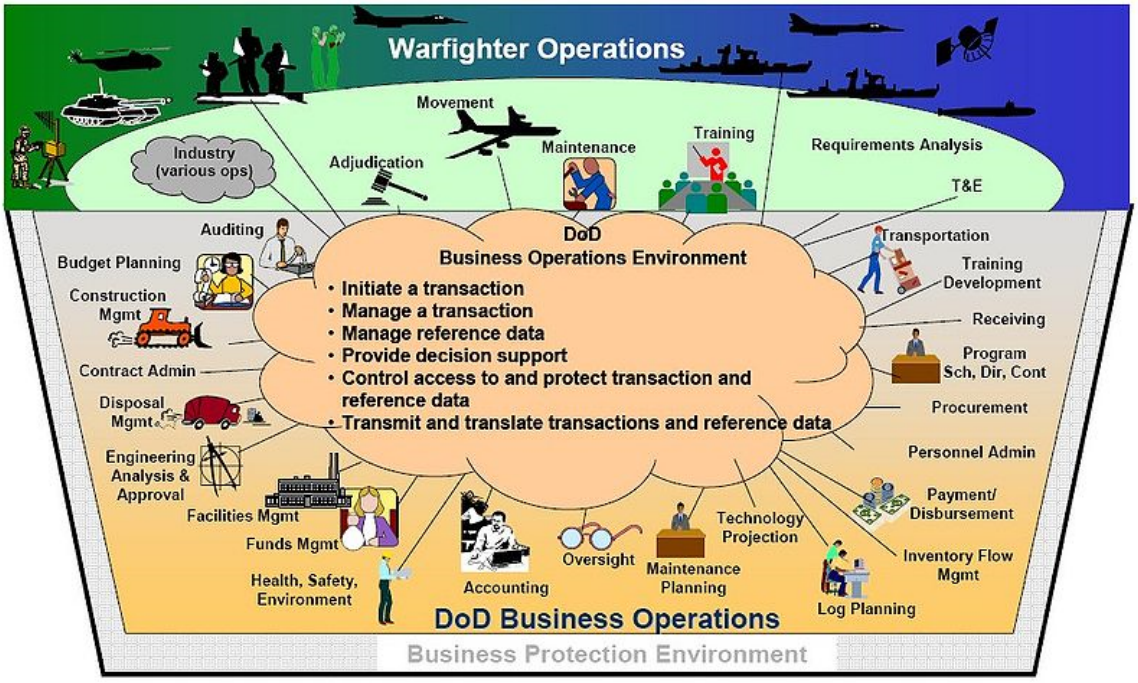
| | | | | |
|---|---|---|---|---|
| | | strategy | This is because there is uncertainty about what a scenario should be. The templates lead to some certainty | |
| 3.2 | IAG-7 | Enact section with automated support | IAG7 proposes a template to be filled in. The template corresponds to a service scenario and contains actions that express services expected from the system | SC4: If the customer gets a card, he recycles objects |
| 4.1 | SSG-2 | Choose section with select strategy | SSG2 offers two strategies to conceptualise a Scenario. Among the two strategies, manual and computer based, the former is chosen since the service scenario (SC4) is very simple and can be handled manually | |
| 4.2 | IAG-10 | Enact section with automated support | IAG10 suggests two things: (1) to avoid anaphoric references such as he, she, etc. (2) to express atomic actions in an explicit ordering (3) to avoid ambiguities The scenario is rewritten accordingly | SC4: 1. The customer gets a card; 2. The customer recycles boxes and bottles |
| 5.1 | SSG-1 | Choose section with select strategy | The RE knows that he wants to analyse the scenario SC4 to discover a new goal. Thus, he knows the target intention ‘Elicit a Goal’ and SSG1 is displayed. SSG1 offers three strategies to discover new goals from scenario analysis. The refinement strategy, is chosen because there is a need to discover the functional requirements of recycling machine | |
| 5.2 | IAG-4 | Enact section with automated | IAG4 guides in transforming actions of the service scenario SC4 into goals which express | G23: Get card from supermarket; G24: Recycle bottles and boxes from RM |

| | | | | |
|---|---|---|---|---|
| | | support | functional requirements. Two goals are generated and related together to G4 with an AND relationship. G24 is selected for further processing | |
| 6.1 | SSG-3 | Choose section with select strategy | The RE knows his target intention, namely 'Write a Scenario'. Thus SSG3 is displayed to help the RE in selecting the right strategy. The free prose strategy is selected because the text is likely to be long and the free prose facilitates this | |
| 6.2 | IAG-8 | Enact section with automated support | IAG8 provides style and contents guidelines adapted to the type of scenario at hand, namely system interaction scenario | SC24-1: The customer inserts his card in the RM. The RM checks if the card is valid and then a prompt is given. The customer inputs the bottles and/or boxes in the RM. If the objects are not blocked, the RM ejects the card and prints a receipt |
| 7.1 | SSG-2 | Choose section with select strategy | SSG2 is displayed. The automated support strategy is selected to take advantage of the powerful linguistic devices and obtain a scenario formulation which will be the basis for automated reasoning | |
| 7.2 | IAG-9 | Enact section with automated support | IAG9 semi-automatically transforms the initial prose into a structured text whose semantics conform to the scenario model. The transformation includes disambiguation, completion and mapping onto the linguistic structures associated to the concepts of the scenario model. | SC24-2: 1. The customer inserts the customer card in the RM, 2. The RM checks if the card is valid, 3. If the card is valid, 4. A prompt is given to the customer, 5. The customer inputs the bottles and the boxes in the RM, 6. The RM |

| | | | | |
|---|---|---|---|---|
| | | | SC24-2 is the result of the transformation of SC24-1. (Underlined statements result of the transformation) | checks if the bottles and the boxes are not blocked, 7. If the bottles and the boxes are not blocked, 8. The RM ejects the card to the customer, 9. The RM prints a receipt to the customer |
| 8.1 | SSG-1 | Choose section with select strategy | Of the three strategies proposed by SSG1, the alternative discovery strategy is chosen. This strategy suits the need to investigate variations and exceptions of the normal course of actions described in SC242 | |
| 8.2 | IAG-3 | Enact section with automated support | IAG3 proposes several tactics to discover alternative goals to G24. The one based on the analysis of conditions in the scenario is selected. This leads to discover G25 and G26 | G25: Recycle box and bottles from RM with invalid card; G26: Recycle box and bottles with a deblocking phase |

Chapter 10

# Operational View




Example of an "Operational View" (OV) on the DoD Electronic Commerce Concept of Operations.

**Operational View** (OV) is one of the basic views, defined in the enterprise architecture (EA) of the Department of Defense Architecture Framework V1.5 (DoDAF). Under DODAF 2, which became operational in 2009, the collections of views are now termed 'viewpoints' and no longer views.

Other enterprise architecture frameworks may or do have operational views. For example the MODAF has an Operational Viewpoint and the NATO Architecture Framework has an Operational View (collection of subviews).

## *Overview*

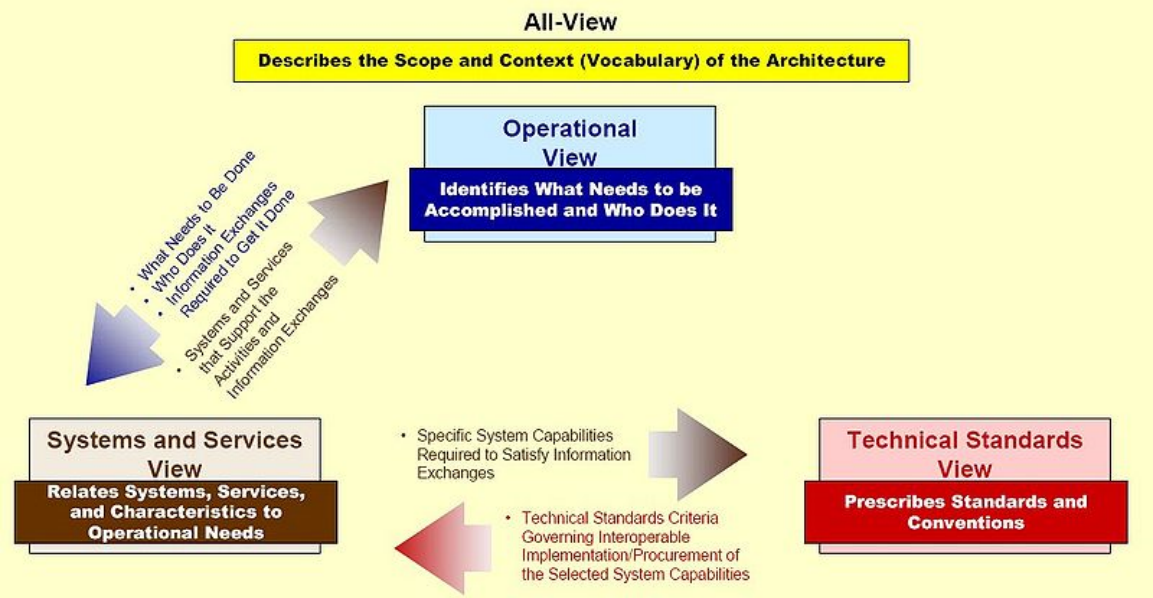
A so called "Operational View" (OV) in the DoDAF Enterprise architecture framework describes the tasks and activities, operational elements, and information exchanges required to conduct operations. A pure Operational View is materiel independent. However, operations and their relationships may be influenced by new technologies such as collaboration technology, where process improvements are in practice before policy can reflect the new procedures.

There may be some cases, as well, in which it is necessary to document the way processes are performed given the restrictions of current systems, in order to examine ways in which new systems could facilitate streamlining the processes. In such cases, an Operational View may have materiel constraints and requirements that must be addressed. For this reason, it may be necessary to include some high-level Systems View (SV) architecture data as overlays or augmenting information onto the Operational View products.

## *Operational View topics*

### DoDAF

The Department of Defense Architecture Framework (DoDAF) defines a standard way to organize a systems architecture into complementary and consistent views. It is especially suited to large systems with complex integration and interoperability challenges, and is apparently unique in its use of "operational views" detailing the external customer's operating domain in which the developing system will operate.




DoDAF view model shows the linkages among views.

The DoDAF defines a set of products that act as mechanisms for visualizing, understanding, and assimilating the broad scope and complexities of an architecture description through graphic, tabular, or textual means. These products are organized under four views:

- Overarching All View (AV),
- Operational View (OV),
- Systems View (SV), and the
- Technical Standards View (TV).

Each view depicts certain perspectives of an architecture as described below. Only a subset of the full DoDAF viewset is usually created for each system development. The figure represents the information that links the operational view, systems and services view, and technical standards view. The three views and their interrelationships driven – by common architecture data elements – provide the basis for deriving measures such as interoperability or performance, and for measuring the impact of the values of these metrics on operational mission and task effectiveness.

## OV products

The Department of Defense Architecture Framework (DoDAF) has defined a series of seven different types of Operational View products. There are seven OV products described in this section:

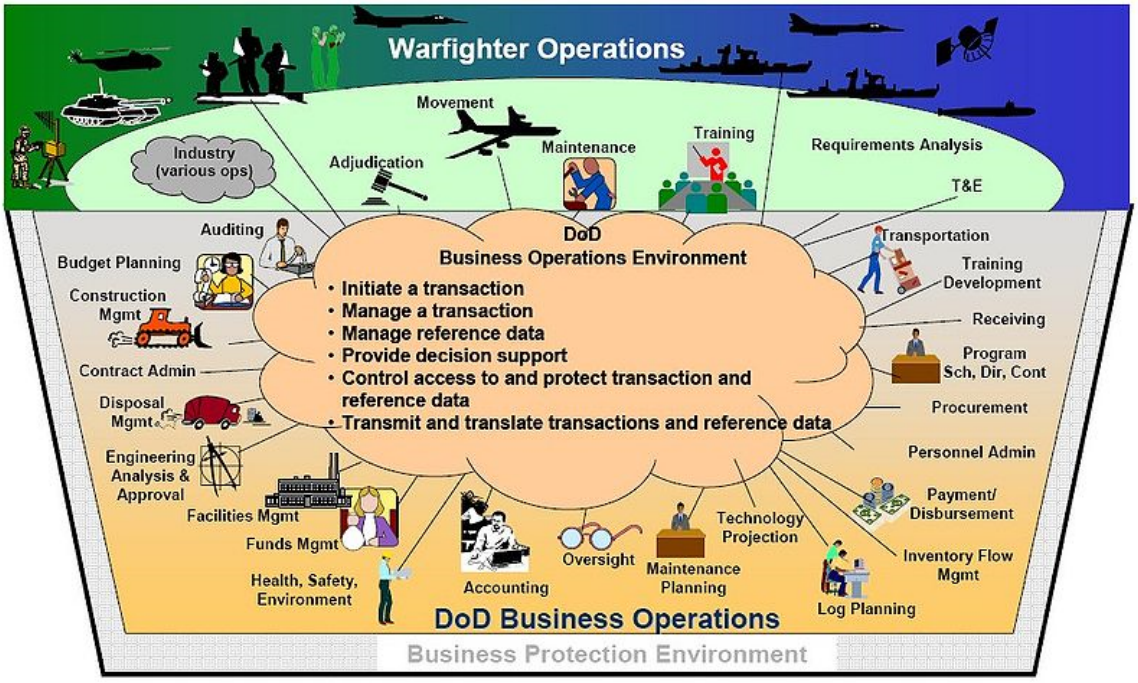
- High-Level Operational Concept Graphic (OV-1)
- Operational Node Connectivity Description (OV-2)
- Operational Information Exchange Matrix (OV-3)

- Organizational Relationships Chart (OV-4)
- Operational Activity Model (OV-5)
- Operational Rules Model, State Transition Description, and Event-Trace

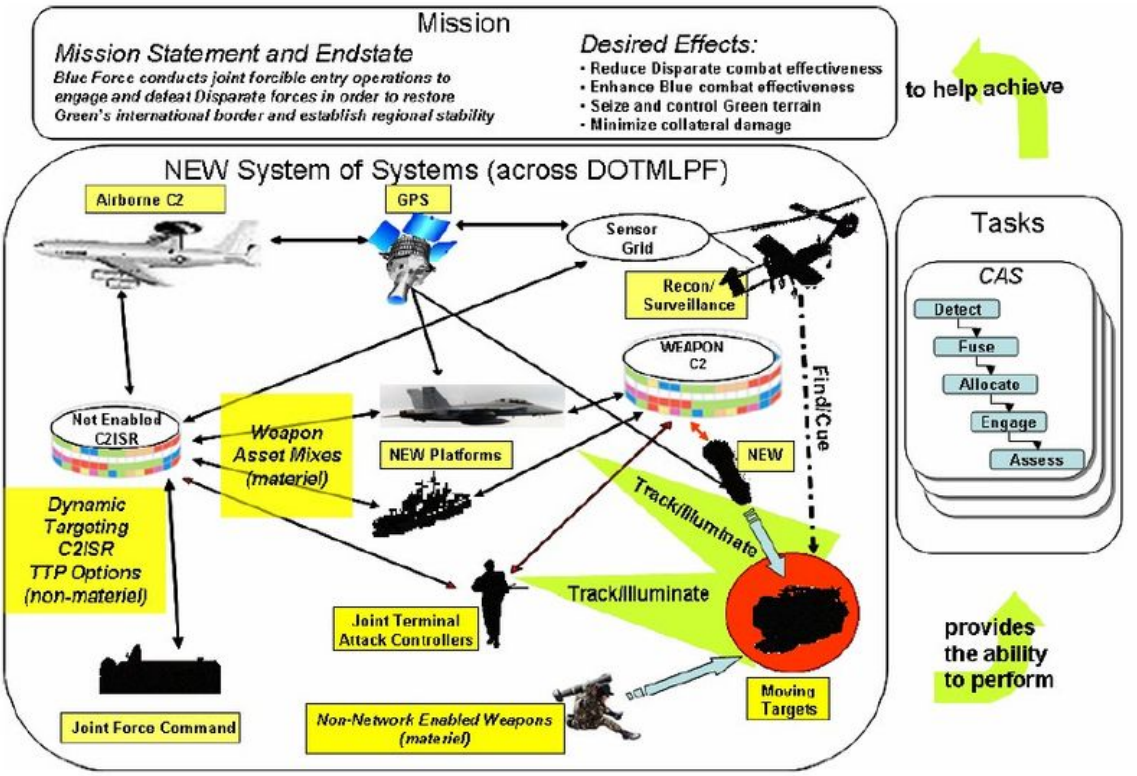
Description (OV-6a, 6b, and 6c)

- Logical Data Model (OV-7)

## High-Level Operational Concept Graphic

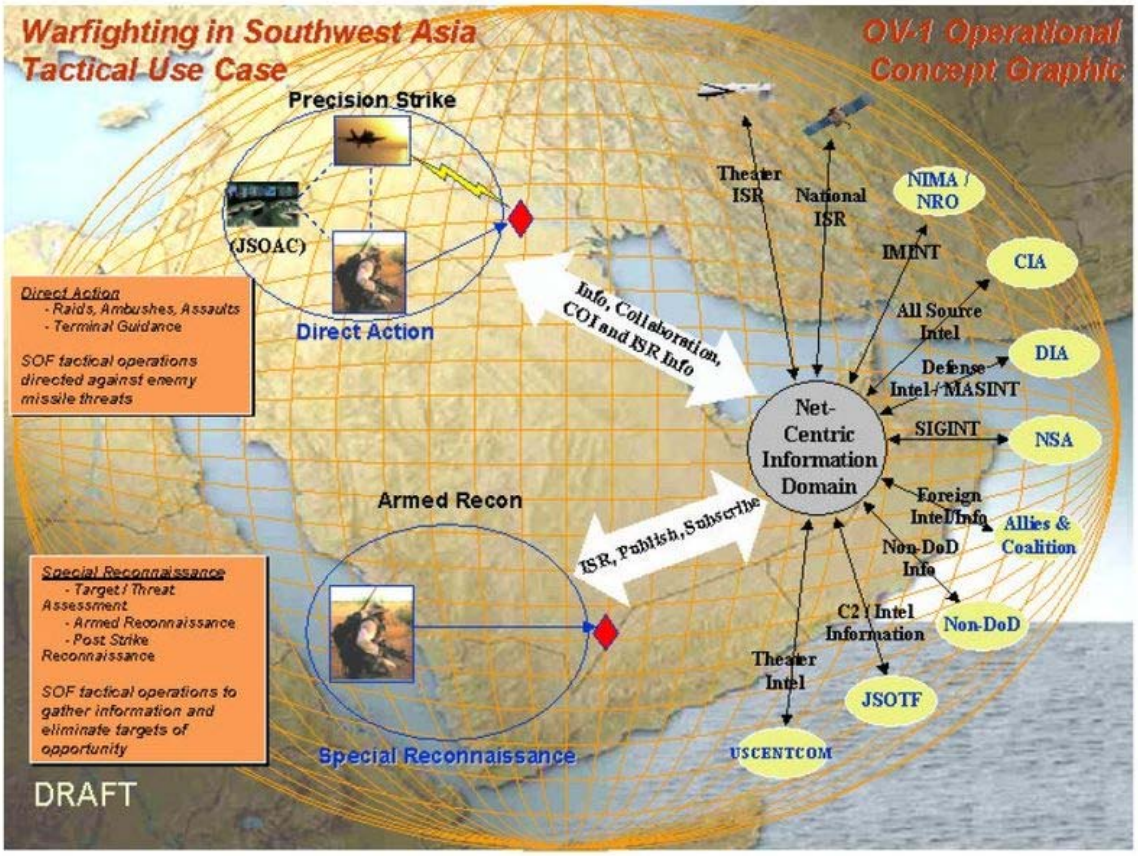



DoD Electronic Commerce Concept of Operations (OV-1).




Joint Network Enabled Weapon (NEW) Capability Operational Concept Graphic (OV-1).

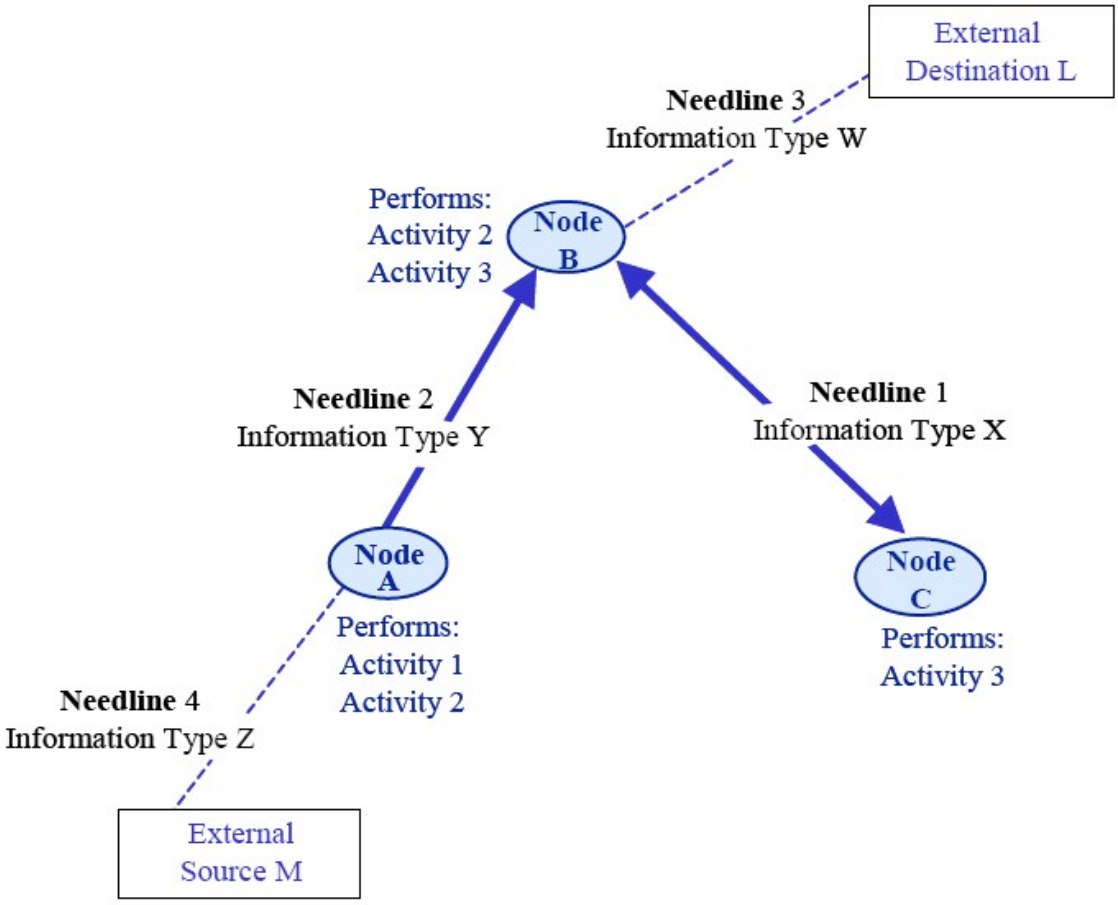

Joint Task Force Concept of Operations (OV-1).



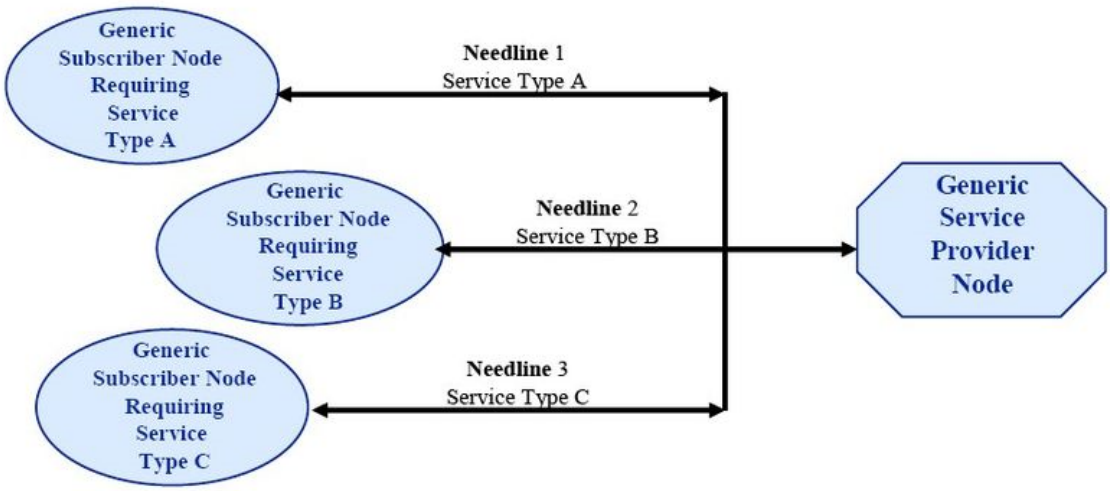

Example OV-1 for NCOW.

- High-Level Operational Concept Graphic (OV-1) : High level graphical and textual description of operational concept (high level organizations, missions, geographic configuration, connectivity, etc).

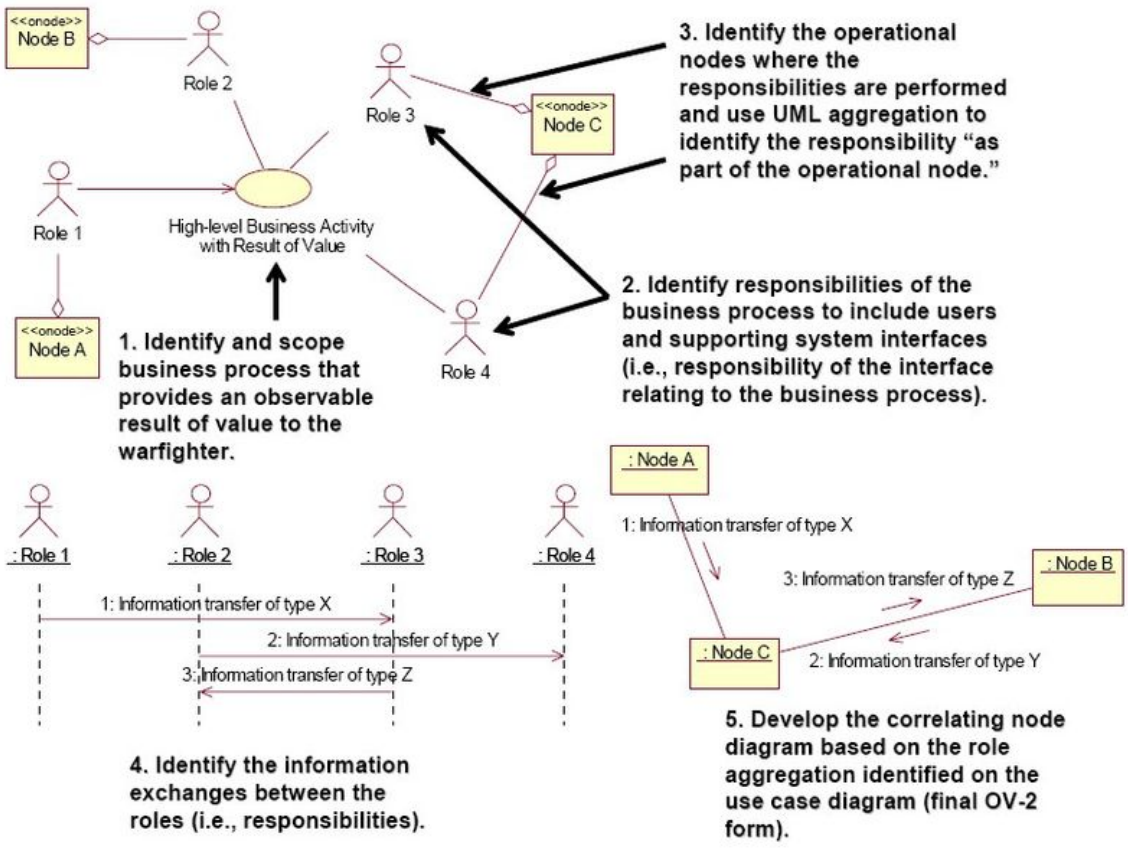
## Operational Node Connectivity Description




OV-2 Template.




Notional Example of an OV-2 Depicting Service Providers.




UML OV-2 Template.

- Operational Node Connectivity Description (OV-2) : Operational nodes, activities performed at each node, and connectivites and information flow between nodes.

## Operational Information Exchange Matrix

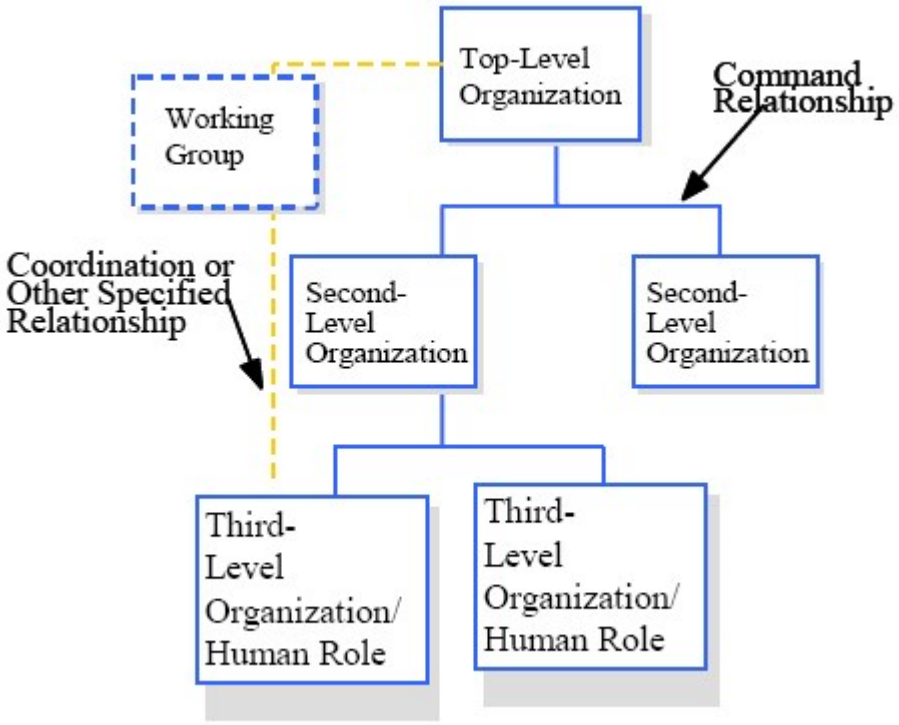
| Needline Identifier | Information Exchange Identifier | Information Element Description | | | | | Producer | | Consumer | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Information Element Name and Identifier | Content | Scope | Accuracy | Language | Sending Op Node Name and Identifier | Sending Op Activity Name and Identifier | Receiving Op Node Name and Identifier | Receiving Op Activity Name and Identifier |

| Needline Identifier | Information Exchange Identifier | Nature of Transaction | | | | | Performance Attributes | | Information Assurance | | | | | Security | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Mission/Scenario UJTL or METL | Transaction Type | Triggering Event | Interoperability Level Required | Criticality | Periodicity | Timeliness | Access Control | Availability | Confidentiality | Dissemination Control | Integrity | Accountability | Protection (Type Name, Duration, Date) | Classification | Classification Caveat |

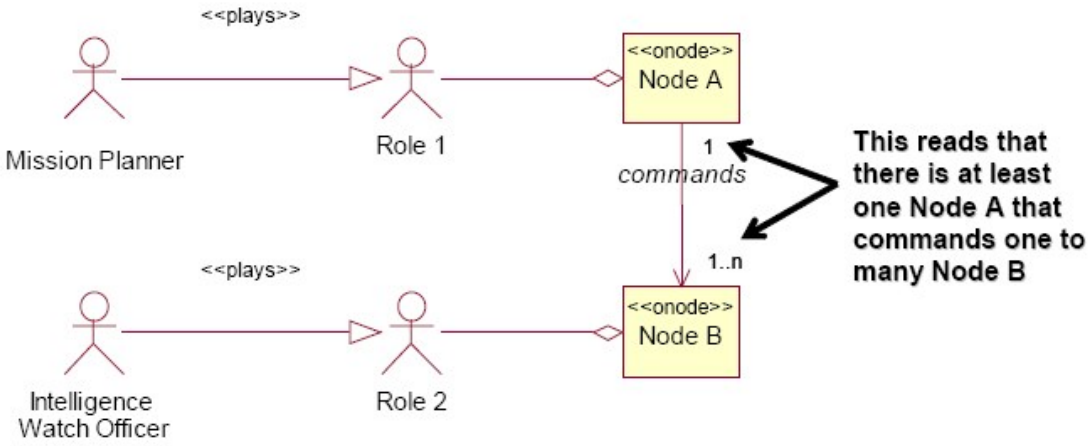
OV-3 – Template.

- Operational Information Exchange Matrix (OV-3) : Information exchanged between nodes and the relevant attributes of that exchange such as media, quality, quantity, and the level of interoperability required.

## Organizational Relationships Chart



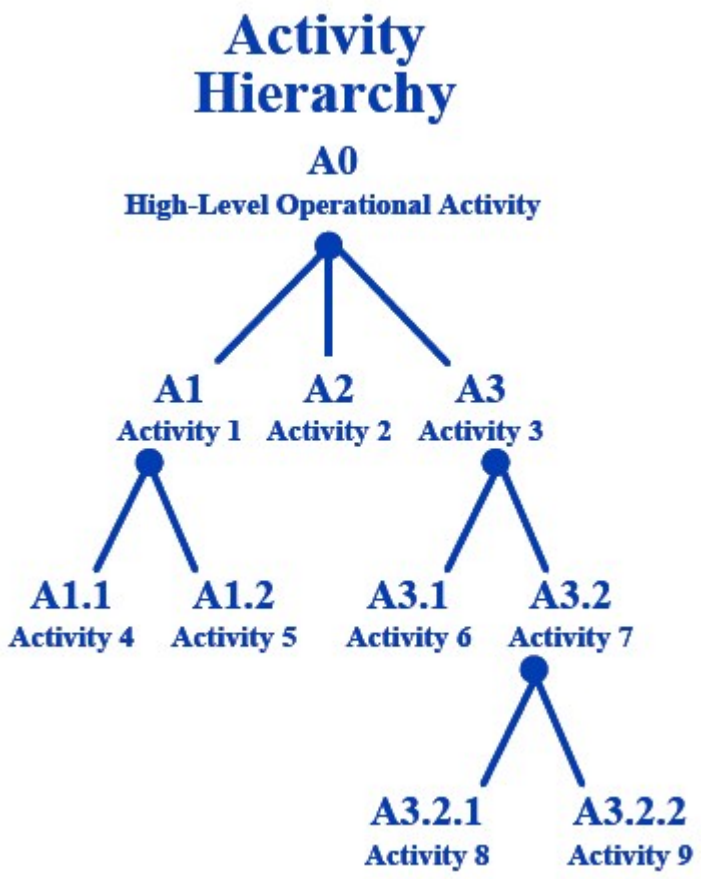

OV-4 – Template.



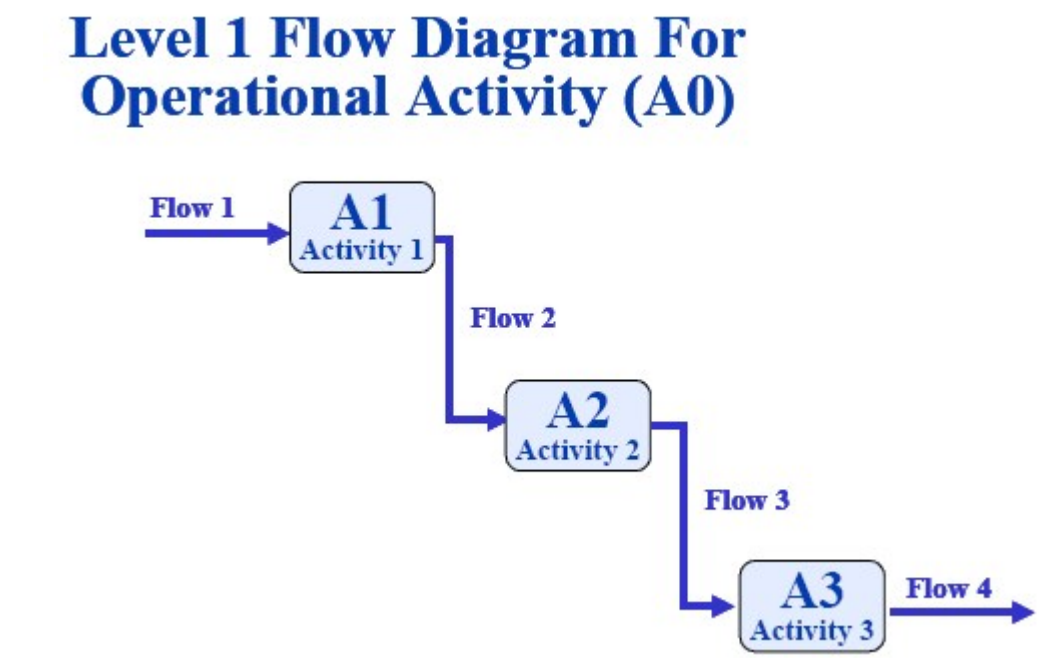

UML OV-4 Sample.

- Organizational Relationships Chart (OV-4) : Command, control, coordination, and other relationships among organizations.

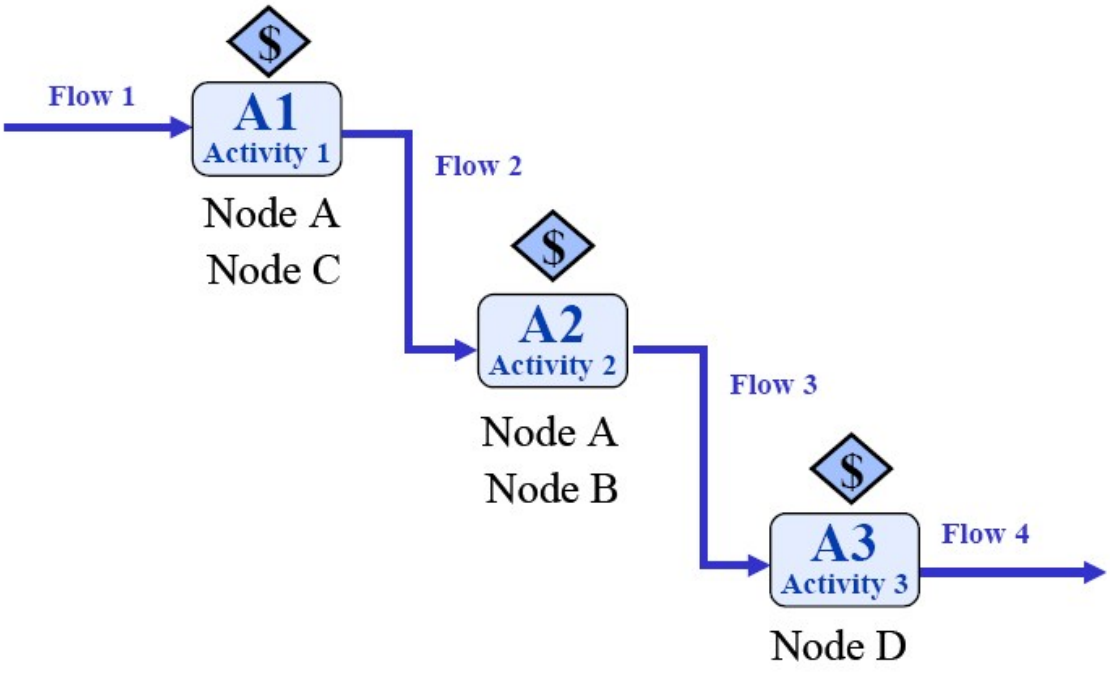
## Operational Activity Model



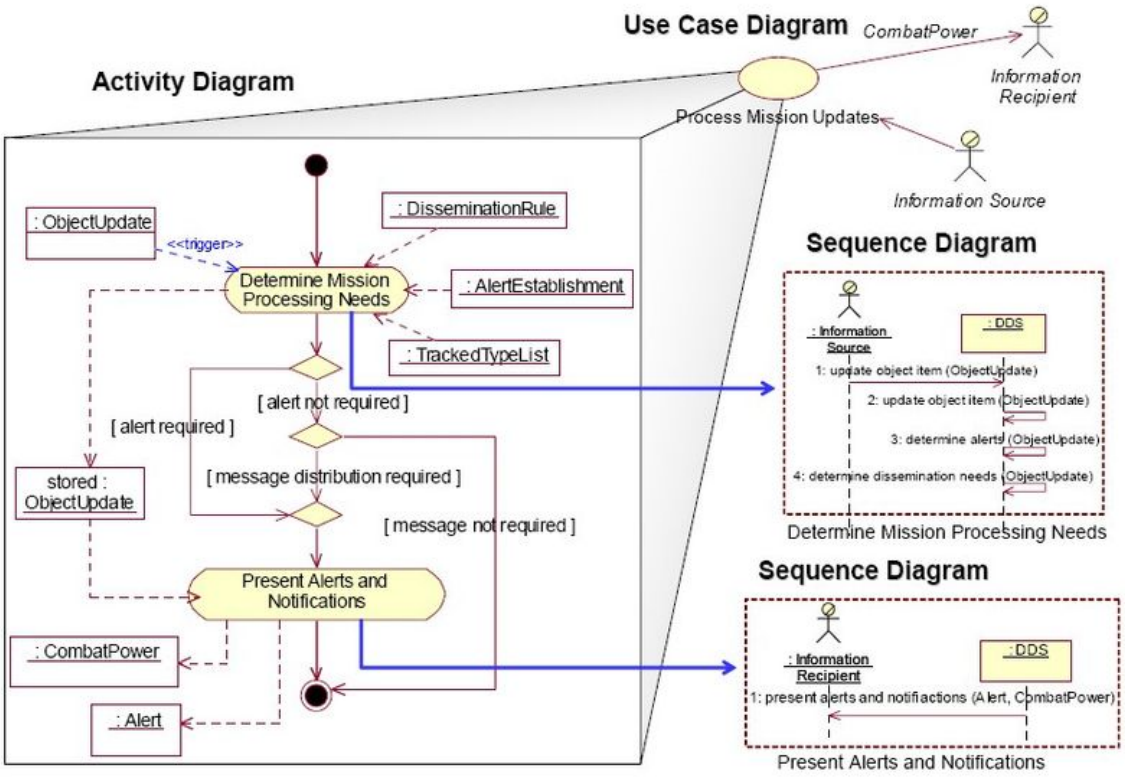

Operational Activity Hierarchy Chart (OV-5) – Template.




Operational Activity Diagram (OV-5) – Template.




OV-5 – Template with Notional Annotations.



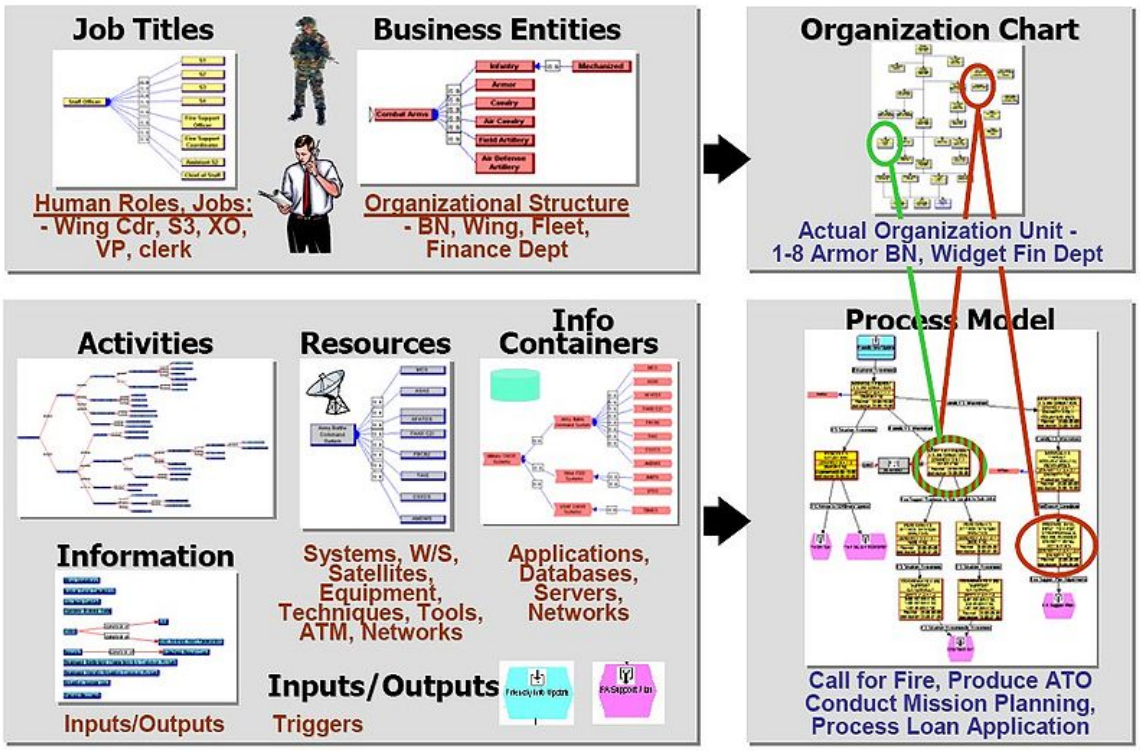

UML Example OV-5.

- Operational Activity Model (OV-5) : Activities, relationships among activities, inputs and outputs. In addition, overlays can show cost, performing nodes, or other pertinent information.

## Other OV products

- Operational Rules Model, State Transition Description, and Event-Trace Description (OV-6a, 6b, and 6c)
  - Operational Rules Model (OV-6a) : One of the three products used to describe operational activity sequence and timing that identifies the business rules that constrain the operation.
  - Operational State Transition Description (OV-6b) : One of the three products used to describe operational activity sequence and timing that identifies responses of a business process to events.
  - Operational Event-Trace Description (OV-6c) : One of the three products used to describe operational activity sequence and timing that traces the actions in a scenario or critical sequence of events.
- Logical Data Model (OV-7) : Documentation of the data requirements and structural business process rules of the Operational View.

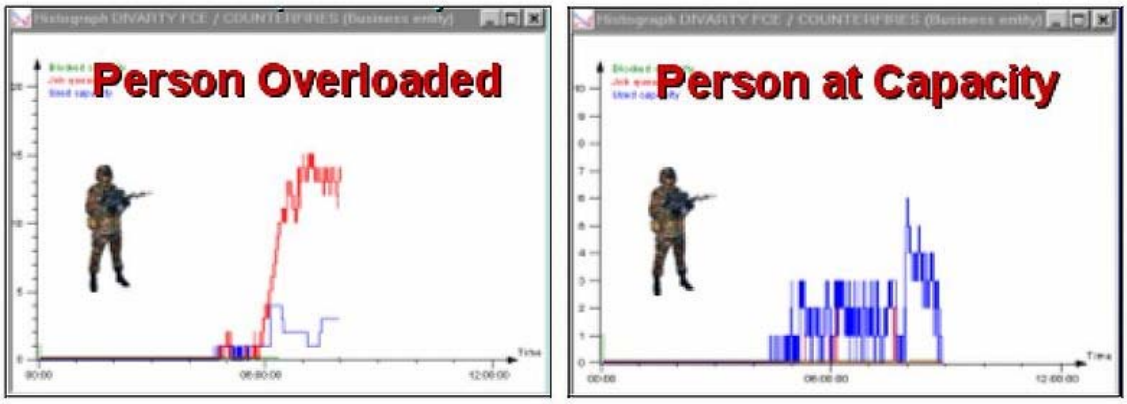
## Executable Operational Architecture




Anatomy of an Executable Operational Architecture.

In addition to examining behavior over time, one can also assess an overall dynamic mission cost over time in terms of human and system/network resource dollar costs and their processes dollar costs. Analysis of dollar costs in executable architectures is a first step in an architecture based investment strategy, where we eventually need to align architectures to funding decisions to ensure that investment decisions are directly linked to mission objectives and their outcomes. The figure on the right illustrates the anatomy of one such dynamic model.

State transitions in executable operational architectural models provide for descriptions of conditions that control the behavior of process events in responding to inputs and in producing outputs. A state specifies the response of a process to events. The response may vary depending on the current state and the rule set or conditions. Distribution settings determine process time executions. Examples of distribution strategies include: constant values, event list, constant interval spacing, normal distribution, exponential distribution, and so forth. Priority determines the processing strategy if two inputs reach a process at the same time. Higher priority inputs are usually processed before lower priority inputs.




Sample Histograms Showing Results of a Simulation Run.

Processes receiving multiple inputs need to define how to respond. Examples of responses include: process each input in the order of arrival independent of each other, process only when all inputs are available, or process as soon as any input is detected. Processes producing multiple outputs can include probabilities (totaling 100 percent), under which each output would be produced.Histograms are examples of generated timing descriptions. They are graphic representations of processes, human and system resources, and their used capacity over time during a simulation run. These histograms are used to perform dynamic impact analysis of the behavior of the executable architecture. Figure 4-23 is an example showing the results of a simulation run of human resource capacity.

Chapter 11

# Modeling Perspective & Modular Design

## Modeling Perspective

A **modeling perspective** in information systems is a particular way to represent pre-selected aspects of a system. Any perspective has a different focus, conceptualization, dedication and visualization of what the model is representing.

The traditional way to distinction between modeling perspectives is structural, functional and behavioral/processual perspectives. This together with rule, object, communication and actor and role perspectives is one way of classifying modeling approaches.

### *Types of perspectives*

#### Structural modeling perspective

This approach concentrates on describing the static structure. The main concept in this modeling perspective is the entity, this could be an object, phenomena, concept, thing etc.

The data modeling languages have traditionally handled this perspective, examples of such being:

- The ER-language (Entity-Relationship)
- Generic Semantic Modeling language (GSM)
- Other approaches including:
    - The NIAM language (Binary relationship language)
    - Conceptual graphs (Sowa)

Looking at the ER-language we have the basic components:

- Entities: Distinctively identifiable phenomenon.
- Relationships: An association among the entities.
- Attributes: Used to give value to a property of an entity/relationship.

Looking at the generic semantic modeling language we have the basic components:

- Constructed types built by abstraction: Aggregation, generalization, and association.
- Attributes.
- Primitive types: Data types in GSM are classified into printable and abstract types.

    - Printable: Used to specify visible values.
    - Abstract: Representing entities.

## Functional modeling perspective

The functional modeling approach concentrates on describing the dynamic process. The main concept in this modeling perspective is the process, this could be a function, transformation, activity, action, task etc. A well-known example of a modeling language employing this perspective is data flow diagrams.

The perspective uses four symbols to describe a process, these being:

- Process: Illustrates transformation from input to output.
- Store: Data-collection or some sort of material.
- Flow: Movement of data or material in the process.
- External Entity: External to the modeled system, but interacts with it.

Now, with these symbols, a process can be represented as a network of these symbols. This decomposed process is a DFD, data flow diagram.

## Behavioral perspective

Behavioral perspective gives a description of system dynamics. The main concepts in behavioral perspective are states and transitions between states. State transitions are triggered by events. State Transition Diagrams (STD/STM), State charts and Petri-nets are some examples of well-known behaviorally oriented modeling languages. Different types of State Transition Diagrams are used particularly within real-time systems and telecommunications systems.

## Rule perspective

Rule perspective gives a description of goals/means connections. The main concepts in rule perspective are rule, goal and constraint. A rule is something that influences the actions of a set of actors. The standard form of rule is "IF condition THEN action/expression". Rule hierarchies (goal-oriented modeling), Tempora and Expert systems are some examples of rule oriented modeling.

## Object perspective

The object-oriented perspective describes the world as autonomous, communicating objects. An object is an "entity" which has a unique and unchangeable identifier and a local state consisting of a collection of attributes with assignable values. The state can only be manipulated with a set of methods defined on the object. The value of the state can only be accessed by sending a message to the object to call on one of its methods. An event is when an operation is being triggered by receiving a message, and the trace of the events during the existence of the object is called the object's life cycle or the process of an object. Several objects that share the same definitions of attributes and operations can be parts of an object class. The perspective is originally based on design and programming of oriented systems. Unified Modelling Language (UML) is a well known language for modeling with an object perspective.

## Communication perspective

This perspective is based on language/action theory from philosophical linguistics. The basic assumption in this perspective is that person/objects cooperate on a process/action thorough communication within them.

An illocutionary act consists of five elements: Speaker, hearer, time, location and circumstances. It is a reason and goal for the communication, where the participations in a communication act is oriented towards mutual agreement. In a communication act, the speaker generally can raise three claims: truth (referring an object), justice (referring a social world of the participations) and claim to sincerity (referring the subjective world of the speaker).

## Actor and role perspective

Actor and role perspective is a description of organisational and system structure. An actor can be defined as a phenomenon that influences the history of another actor, whereas a role can be defined as the behaviour which is expected by an actor, amongst other actors, when filling the role. Modeling within these perspectives is based both on work with object-oriented programming languages and work with intelligent agents in artificial intelligence. I* is an example of an actor oriented language.

# Modular Design

In systems engineering, **modular design** — or "modularity in design" — is an approach that subdivides a system into smaller parts (modules) that can be independently created and then used in different systems to drive multiple functionalities. A modular system can be characterized by the following:

> "(1) Functional partitioning into discrete scalable, reusable modules consisting of isolated, self-contained functional elements; (2) Rigorous use of well-defined modular interfaces, including object-oriented descriptions of module functionality; (3) Ease of change to achieve technology transparency and, to the extent possible, make use of industry standards for key interfaces."

Besides reduction in cost (due to lesser customization, and less learning time), and flexibility in design, modularity offers other benefits such as augmentation (adding new solution by merely plugging in a new module), and exclusion. Examples of modular systems are cars, computers and high rise buildings. Earlier examples include looms, railroad signaling systems, telephone exchanges, pipe organs and electric power distribution systems. Computers use modularity to overcome changing customer demands and to make the manufacturing process more adaptive to change. Modular design is an attempt to combine the advantages of standardization (high volume normally equals low manufacturing costs) with those of customization. A downside to modularity (and this depends on the extent of modularity) is that modular systems are not optimized for performance. This is usually due to the cost of putting up interfaces between modules.

## *Proper inter-modular design*

Recognizing that excessive inter-module dependencies are an indicator of poor software design, a system should be intended to be loosely coupled to avoid unnecessary dependencies. Thus, inter-modular design should be easy to work with because modules can be easily understood in isolation, and changes or extensions to functionality would be easily localized.

## *Modular design in cars*

Aspects of modular design can be seen in cars or other vehicles to the extent of there being certain parts to the car that can be added or removed without altering the rest of the car.

A simple example of modular design in cars is the fact that, while many cars come as a basic model, paying extra will allow for "snap in" upgrades such as a more powerful engine or seasonal tires; these do not require any change to other units of the car such as the chassis, steering or exhaust systems.

## *Modular design in buildings*

Modular design can be seen in certain buildings, especially modular buildings. Modular buildings (and also modular homes) generally consist of universal parts (or modules) that are manufactured in a factory and then shipped to a build site where they are assembled into a variety of arrangements.

Modular buildings can be added to or reduced in size by adding or removing certain components. This can be done without altering larger portions of the building. Modular buildings can also undergo changes in functionality using the same process of adding or removing modular components.

For example, an office building can be built using modular parts such as walls, frames, doors, and windows. The office interior can then be partitioned (or divided) with more walls and furnished with desks, computers, and whatever else is needed for a functioning workspace. If the office needs to be expanded or redivided to accommodate employees, modular components such as wall panels can be added or relocated to make the necessary changes without altering the whole building. Later on, this same office can be broken down and rearranged to form a retail space, conference hall or any other possible type of building using the same modular components that originally formed the office building. The new building can then be refurnished with whatever items are needed to carry out its desired functions.

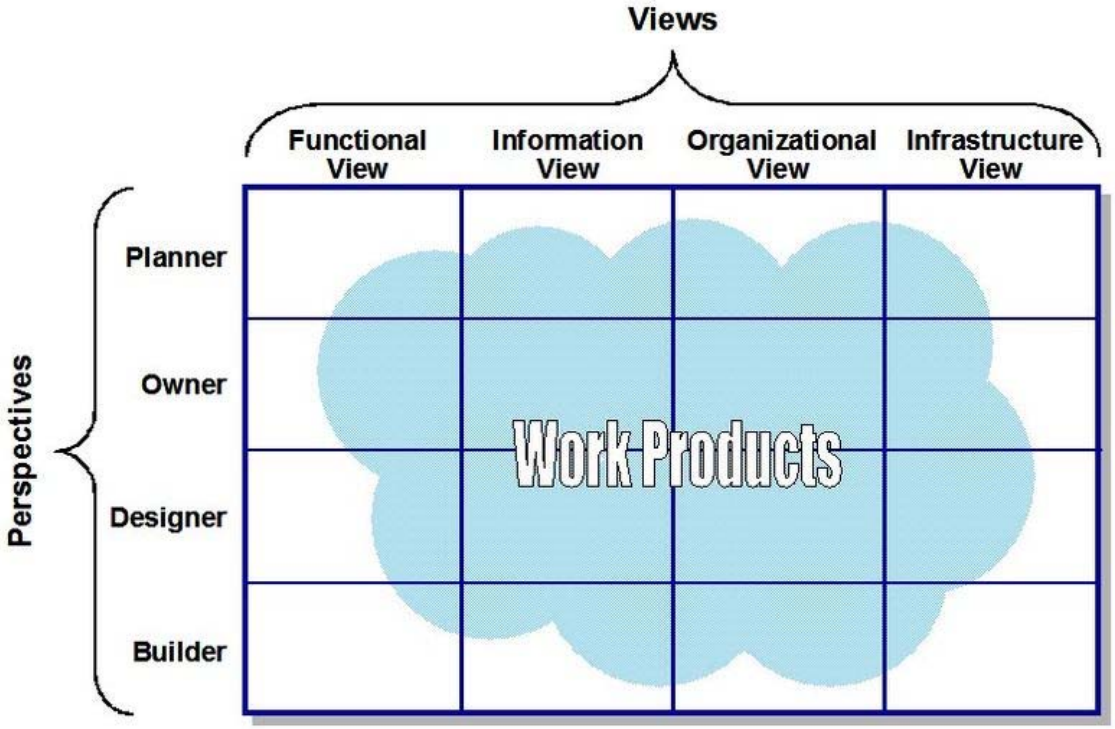
## *Modular design in computers*

Same as modular design in other things (eg cars, fridges, even furniture). The idea is to build computers with easily replaceable parts that use standardized interfaces. This allows you to upgrade certain aspects of the computer with ease without having to buy another computer altogether.

A computer is actually one of the best examples of modular design - typical modules are Power supply unit (computer)s, processors, mainboards, graphics cards, hard drives, optical drives, etc. All of these parts should be easily interchangeable, as long as you use parts that support the same standard interface as the part you replaced.

Chapter 12

# View Model




The TEAF Matrix of Views and Perspectives.

A **view model** or *viewpoints framework* in systems engineering, software engineering, and enterprise engineering is a framework which defines a coherent set of *views* to be used in the construction of a system architecture, software architecture, or enterprise architecture. A *view* is a representation of a whole system from the perspective of a related set of concerns. View model provides guidance and rules for constructing, classifying, and organizing architectures.

Since the early 1990s there have been a number of efforts to define standard approaches for describing and analyzing system architectures. Each of the recent Enterprise Architecture frameworks define a set of views, but these sets are not always called "view models".

Usually one *view* is a very concrete product that presents a specific set of architecture data for a given system. However, the same term is sometimes used to refer to a view *definition*, including the particular viewpoint and the corresponding guidance that defines each concrete view. The term *view model* is related to view definitions.

## Overview

The purpose of viewpoints and views is to enable human engineers to comprehend very complex systems, and to organize the elements of the problem and the solution around domains of expertise. In the engineering of physically-intensive systems, viewpoints often correspond to capabilities and responsibilities within the engineering organization.

Most complex system specifications are so extensive that no single individual can fully comprehend all aspects of the specifications. Furthermore, we all have different interests in a given system and different reasons for examining the system's specifications. A business executive will ask different questions of a system make-up than would a system implementer. The concept of viewpoints framework, therefore, is to provide separate viewpoints into the specification of a given complex system in order to facilitate communication with the stakeholders. Each viewpoint satisfies an audience with interest in a particular set of aspects of the system. Each viewpoint may use a specific *viewpoint language* that optimizes the vocabulary and presentation for the audience of that viewpoint. Viewpoint modeling has become an effective approach for dealing with the inherent complexity of large distributed systems.

Current software architectural practices, as described in IEEE Std. 1471, divide the design activity into several areas of concerns, each one focusing on a specific aspect of the system. Examples include the "4+1" view model, the Zachman Framework, TOGAF, DoDAF and, RM-ODP.

## History

Since the early 1990s there have been a number of efforts to define standard approaches for describing and analyzing system architectures. Many of these have been funded by the United States Department of Defense, but some have sprung from international or national efforts in ISO or the IEEE. The most relevant of these, the IEEE Recommended Practice for Architectural Description of Software-Intensive Systems (IEEE-Std-1471-2000) provides some very useful definitions and guidelines for what a system architecture is and for the use of viewpoint specifications to address the stakeholder concerns.

The IEEE 1471 specification describes the process for developing architecture descriptions under a number of scenarios, including precedented and unprecedented

design, evolutionary design, and capture of design of existing systems. In all of these scenarios the overall process is the same: identify stakeholders, elicit concerns, identify a set of viewpoints to be used, and then apply these viewpoint specifications to develop a set of relevant views of the system. Although IEEE-1471 does mention the possibility of defining system, functional, and technical views (among others), it does not go so far as to define any specific set of views nor what these might be. Eventually in 1996 the ISO Reference Model for Open Distributed Processing (RM-ODP) was published to provide a useful framework for describing the architecture and design of large scale distributed systems.

## *View model topics*

### View

A view of a system is a representation of the system from the perspective of a viewpoint. This viewpoint on a system involves a perspective focusing on specific concerns regarding the system, which suppresses details to provide a simplified model having only those elements related to the concerns of the viewpoint. For example, a security viewpoint focuses on security concerns and a security viewpoint model contains those elements that are related to security from a more general model of a system.

A view allows a user to examine a portion of a particular interest area. For example, an Information View may present all functions, organizations, technology, etc. that use a particular piece of information, while the Organizational View may present all functions, technology, and information of concern to a particular organization. In the Zachman Framework views comprise a group of work products whose development requires a particular analytical and technical expertise because they focus on either the “what,” “how,” “who,” “where,” “when,” or “why” of the enterprise. For example, Functional View work products answer the question “how is the mission carried out?” They are most easily developed by experts in functional decomposition using process and activity modeling. They show the enterprise from the point of view of functions. They also may show organizational and information components, but only as they relate to functions.

### Viewpoints

Viewpoint is a systems engineering concept that describes a partitioning of concerns in system restricted to a particular set of concerns. Adoption of a viewpoint is usable so that issues in those aspects can be addressed separately. A good selection of viewpoints also partitions the design of the system into specific areas of expertise.

Viewpoints provide the conventions, rules, and languages for constructing, presenting and analysing views. In ISO/IEC 42010:2007 (IEEE-Std-1471-2000) a viewpoint is a specification for an individual view. A view is a representation of a whole system from the perspective of a point. A view may consist of one or more architectural models. Each such architectural model is developed using the methods established by its associated architectural system, as well as for the system as a whole.

## Modeling perspectives

Modeling perspectives is a set of different ways to represent pre-selected aspects of a system. Each perspective has a different focus, conceptualization, dedication and visualization of what the model is representing.

In information systems, the traditional way to divide modeling perspectives is to distinguish the structural, functional and behavioral/processual perspectives. This together with rule, object, communication and actor and role perspectives is one way of classifying modeling approaches

## Viewpoint model

In any given viewpoint, it is possible to make a model of the system that contains only the objects that are visible from that viewpoint, but also captures all of the objects, relationships and constraints that are present in the system and relevant to that viewpoint. Such a model is said to be a viewpoint model, or a view of the system from that viewpoint.

A given view is a specification for the system at a particular level of abstraction from a given viewpoint. Different levels of abstraction contain different levels of detail. Higher-level views allow the engineer to fashion and comprehend the whole design and identify and resolve problems in the large. Lower-level views allow the engineer to concentrate on a part of the design and develop the detailed specifications.

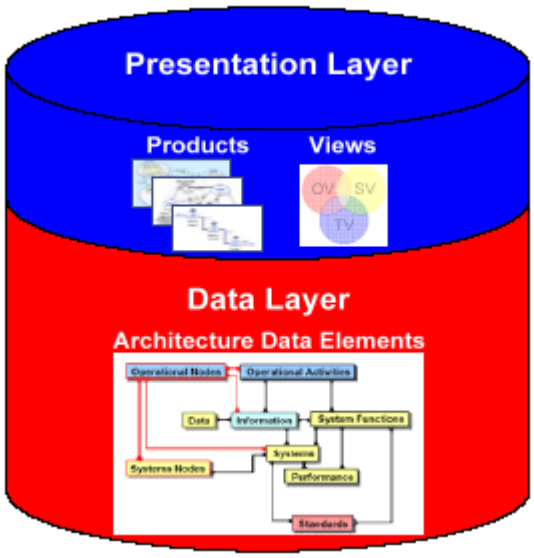



Illustration of the views, products and data in DoDAF Architecture Framework.

In the system itself, however, all of the specifications appearing in the various viewpoint models must be addressed in the realized components of the system. And the specifications for any given component may be drawn from many different viewpoints. On the other hand, the specifications induced by the distribution of functions over specific components and component interactions will typically reflect a different partitioning of concerns than that reflected in the original viewpoints. Thus additional viewpoints, addressing the concerns of the individual components and the bottom-up synthesis of the system, may also be useful.

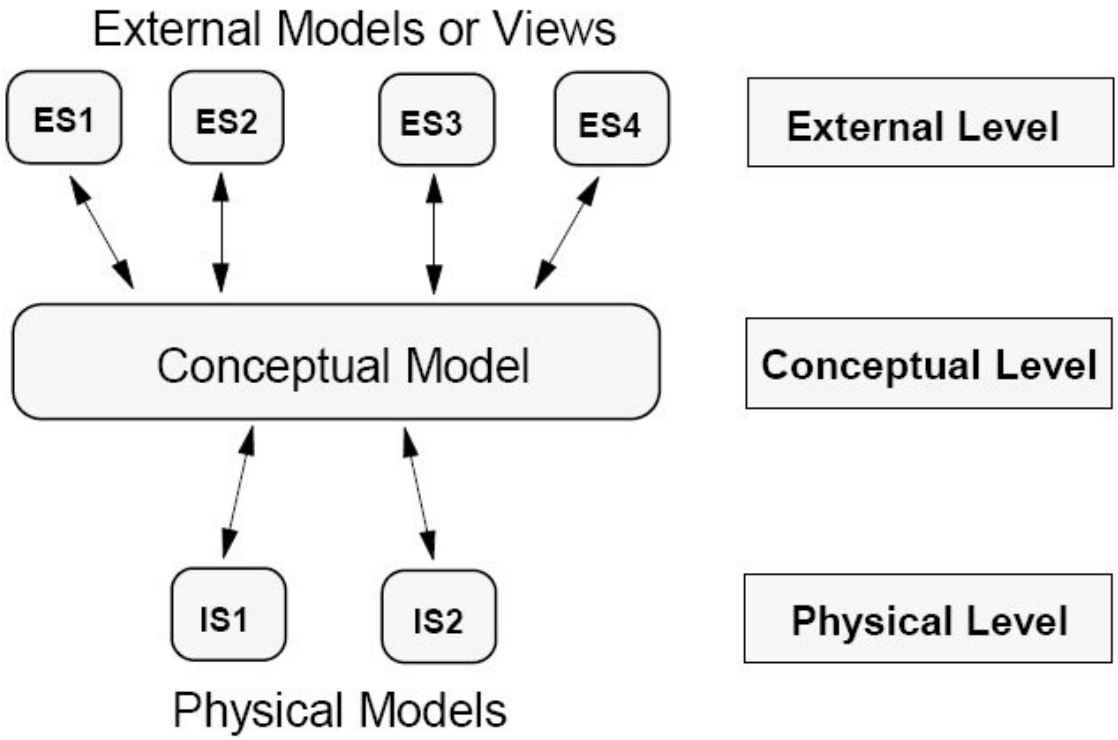
## Architecture description

An architecture description is a representation of a system architecture, at any time, in terms of its component parts, how those parts function, the rules and constraints under which those parts function, and how those parts relate to each other and to the environment. In an architecture description the *architecture data* is shared across several views and products.

At the data layer are the architecture data elements and their defining attributes and relationships. At the presentation layer are the products and views that support a visual means to communicate and understand the purpose of the architecture, what it describes, and the various architectural analyses performed. Products provide a way for visualizing architecture data as graphical, tabular, or textual representations. Views provide the ability to visualize architecture data that stem across products, logically organizing the data for a specific or holistic perspective of the architecture.

## *Types of System View Models*

### Three schema approach




The notion of a three-schema model was first introduced in 1977 by the ANSI/X3/SPARC three level architecture, which determined three levels to model data.

The Three schema approach for data modeling, introduced in 1977, can be considered one of the first view models. It is an approach to building information systems and systems information management, that promotes the conceptual model as the key to achieving data integration. The Three schema approach defines three schema's and views:

- External schema for user views
- Conceptual schema integrates external schemata
- Internal schema that defines physical storage structures

At the center, the conceptual schema defines the ontology of the concepts as the users think of them and talk about them. The physical schema describes the internal formats of the data stored in the database, and the external schema defines the view of the data presented to the application programs. The framework attempted to permit multiple data models to be used for external schemata.

Over the years, the skill and interest in building information systems has grown tremendously. However, for the most part, the traditional approach to building systems has only focused on defining data from two distinct views, the "user view" and the

"computer view". From the user view, which will be referred to as the “external schema,” the definition of data is in the context of reports and screens designed to aid individuals in doing their specific jobs. The required structure of data from a usage view changes with the business environment and the individual preferences of the user. From the computer view, which will be referred to as the “internal schema,” data is defined in terms of file structures for storage and retrieval. The required structure of data for computer storage depends upon the specific computer technology employed and the need for efficient processing of data.

## 4+1 View Model

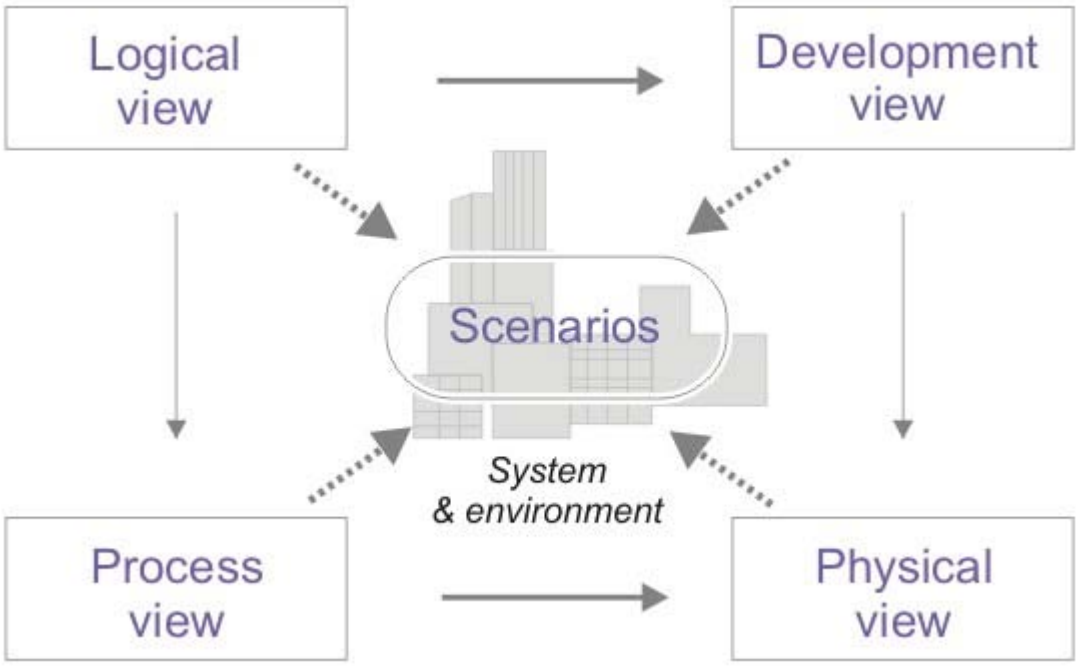



Illustration of the 4+1 Architectural View Model.

4+1 is a view model designed by Philippe Kruchten in 1995 for describing the architecture of software-intensive systems, based on the use of multiple, concurrent views. The views are used to describe the system in the viewpoint of different stakeholders, such as end-users, developers and project managers. The four views of the model are logical, development, process and physical view:

The four views of the model are concerned with :

- *Logical view* : is concerned with the functionality that the system provides to end-users.
- *Development view* : illustrates a system from a programmers perspective and is concerned with software management.

- *Process view* : deals with the dynamic aspect of the system, explains the system processes and how they communicate, and focuses on the runtime behavior of the system.
- *Physical view* : depicts the system from a system engineer's point-of-view. It is concerned with the topology of software components on the physical layer, as well as communication between these components.

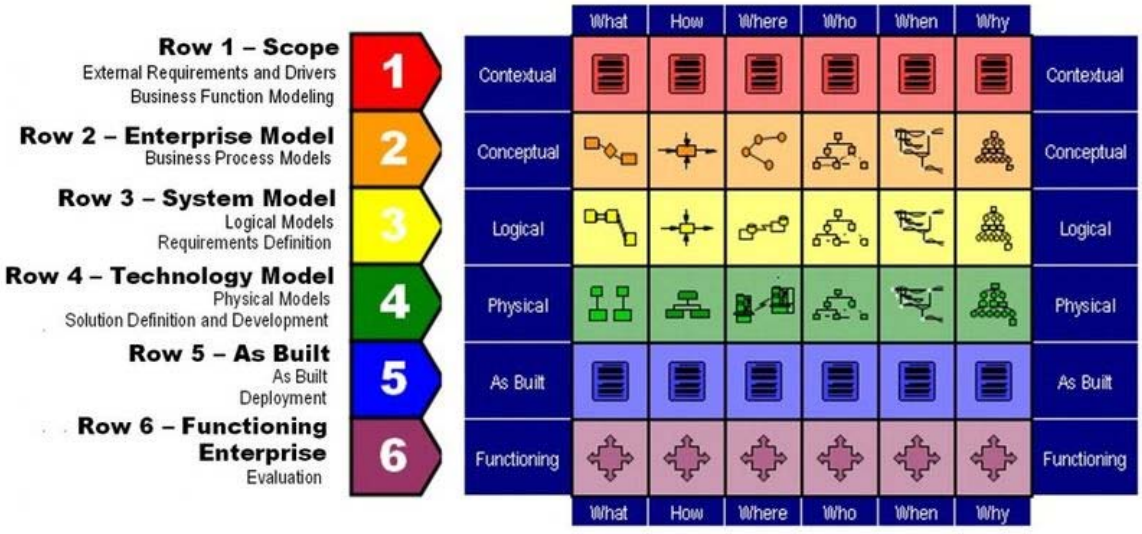
In addition selected use cases or scenarios are utilized to illustrate the architecture. Hence the model contains 4+1 views.

## *Types of Enterprise Architecture View Models*

Enterprise Architecture framework defines how to organize the structure and views associated with an Enterprise Architecture. Because the discipline of Enterprise Architecture and Engineering is so broad, and because enterprises can be large and complex, the models associated with the discipline also tend to be large and complex. To manage this scale and complexity, an Architecture Framework provides tools and methods that can bring the task into focus and allow valuable artifacts to be produced when they are most needed.

Architecture Frameworks are commonly used in Information technology and Information system governance. An organization may wish to mandate that certain models be produced before a system design can be approved. Similarly, they may wish to specify certain views be used in the documentation of procured systems - the U.S. Department of Defense stipulates that specific DoDAF views be provided by equipment suppliers for capital project above a certain value.

### Zachman Framework



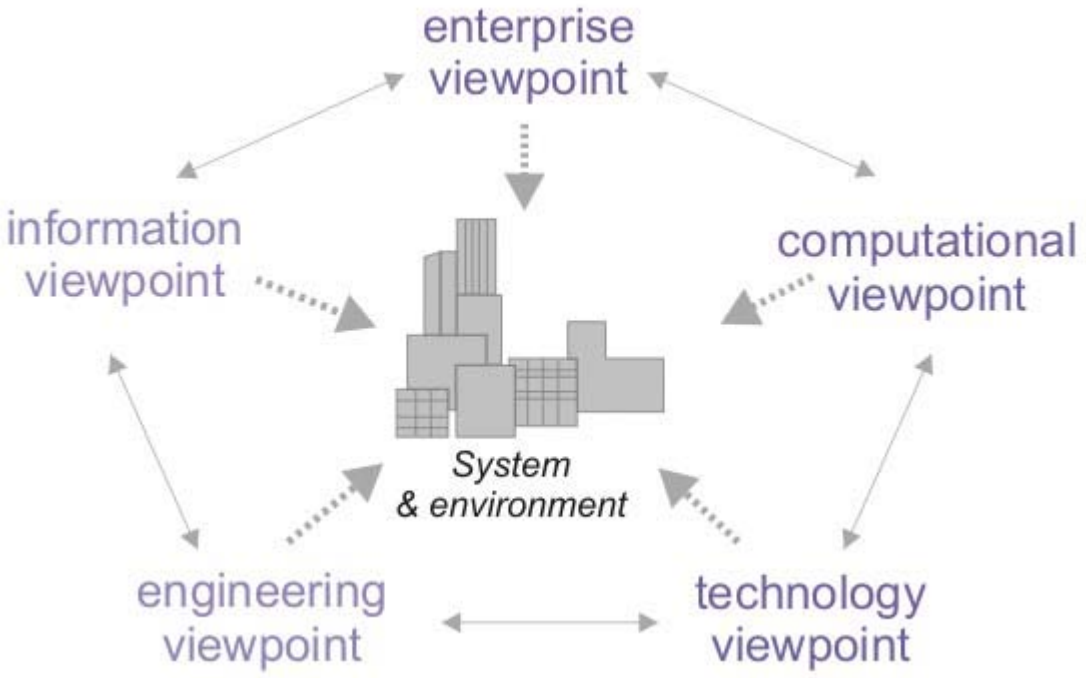

Simplified illustration of the Zachman Framework with an explanation of the rows. The original framework is more advanced.

The Zachman Framework, originally conceived by John Zachman at IBM in the 1987, is a framework for enterprise architecture, which provides a formal and highly structured way of viewing and defining an enterprise.

The Framework is used for organizing architectural "artifacts" in a way that takes into account both who the artifact targets (for example, business owner and builder) and what particular issue (for example, data and functionality) is being addressed. These artifacts may include design documents, specifications, and models.

The Zachman Framework is often referenced as a standard approach for expressing the basic elements of enterprise architecture. The Zachman Framework has been recognized by the U.S. Federal Government as having "... received worldwide acceptance as an integrated framework for managing change in enterprises and the systems that support them."

## RM-ODP views




The RM-ODP view model, which provides five generic and complementary viewpoints on the system and its environment.

The International Organization for Standardization (ISO) Reference Model for Open Distributed Processing (RM-ODP) specifies a set of viewpoints for partitioning the design of a distributed software/hardware system. Since most integration problems arise in the design of such systems or in very analogous situations, these viewpoints may prove useful in separating integration concerns. The RMODP viewpoints are:

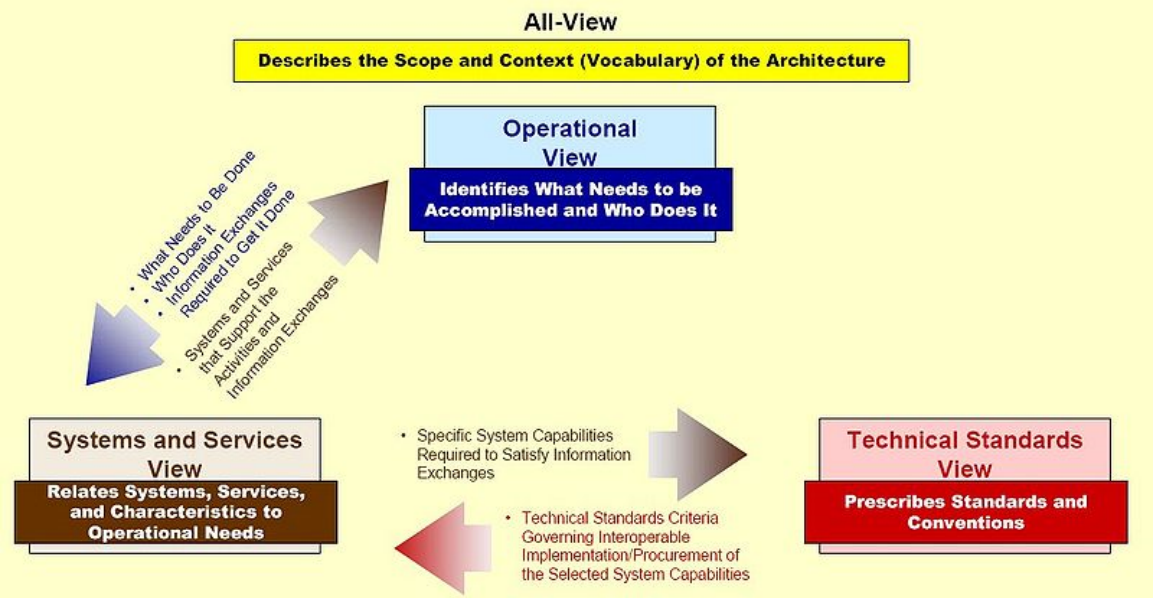
- the *enterprise viewpoint*, which is concerned with the purpose and behaviors of the system as it relates to the business objective and the business processes of the organization
- the *information viewpoint*, which is concerned with the nature of the information handled by the system and constraints on the use and interpretation of that information
- the *computational viewpoint*, which is concerned with the functional decomposition of the system into a set of components that exhibit specific behaviors and interact at interfaces
- the *engineering viewpoint*, which is concerned with the mechanisms and functions required to support the interactions of the computational components
- the *technology viewpoint*, which is concerned with the explicit choice of technologies for the implementation of the system, and particularly for the communications among the components

RMODP further defines a requirement for a design to contain specifications of consistency between viewpoints, including:

- the use of enterprise objects and processes in defining information units
- the use of enterprise objects and behaviors in specifying the behaviors of computational components, and use of the information units in defining computational interfaces
- the association of engineering choices with computational interfaces and behavior requirements
- the satisfaction of information, computational and engineering requirements in the chosen technologies

## DoDAF views

The Department of Defense Architecture Framework (DoDAF) defines a standard way to organize an enterprise architecture (EA) or systems architecture into complementary and consistent views. It is especially suited to large systems with complex integration and interoperability challenges, and is apparently unique in its use of "operational views" detailing the external customer's operating domain in which the developing system will operate.




DoDAF linkages among views.

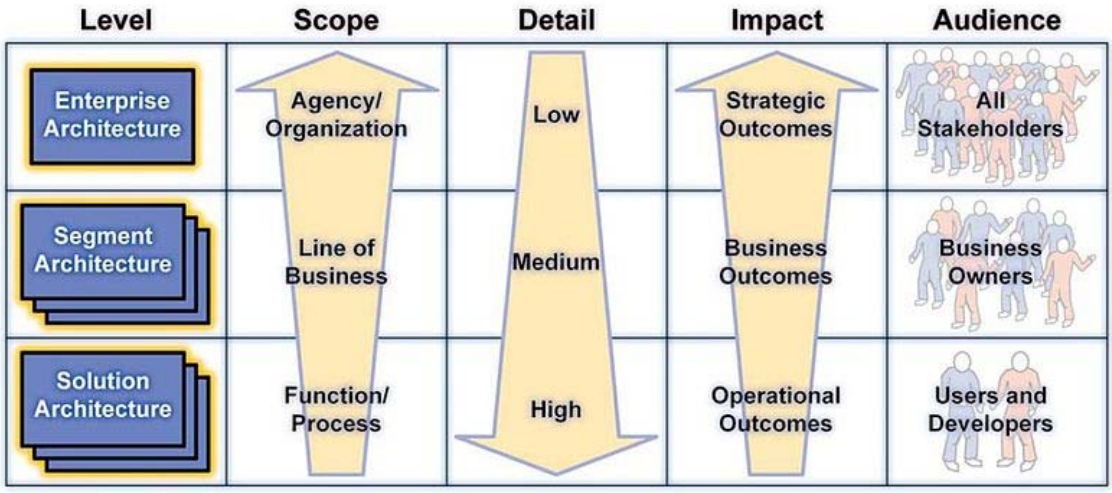
The DoDAF defines a set of products that act as mechanisms for visualizing, understanding, and assimilating the broad scope and complexities of an architecture description through graphic, tabular, or textual means. These products are organized under four views:

- Overarching All View (AV),
- Operational View (OV),
- Systems View (SV), and the
- Technical Standards View (TV).

Each view depicts certain perspectives of an architecture as described below. Only a subset of the full DoDAF viewset is usually created for each system development. The figure represents the information that links the operational view, systems and services view, and technical standards view. The three views and their interrelationships driven – by common architecture data elements – provide the basis for deriving measures such as interoperability or performance, and for measuring the impact of the values of these metrics on operational mission and task effectiveness.

## Federal Enterprise Architecture views

In the US Federal Enterprise Architecture enterprise, segment, and solution architecture provide different business perspectives by varying the level of detail and addressing related but distinct concerns. Just as enterprises are themselves hierarchically organized, so are the different views provided by each type of architecture. The Federal Enterprise Architecture Practice Guidance (2006) has defined three types of architecture:




Federal Enterprise Architecture levels and attributes

- Enterprise architecture,
- Segment architecture, and
- Solution architecture.

By definition, Enterprise Architecture (EA) is fundamentally concerned with identifying common or shared assets – whether they are strategies, business processes, investments, data, systems, or technologies. EA is driven by strategy; it helps an agency identify whether its resources are properly aligned to the agency mission and strategic goals and objectives. From an investment perspective, EA is used to drive decisions about the IT investment portfolio as a whole. Consequently, the primary stakeholders of the EA are the senior managers and executives tasked with ensuring the agency fulfills its mission as effectively and efficiently as possible.

By contrast, segment architecture defines a simple roadmap for a core mission area, business service, or enterprise service. Segment architecture is driven by business management and delivers products that improve the delivery of services to citizens and agency staff. From an investment perspective, segment architecture drives decisions for a business case or group of business cases supporting a core mission area or common or shared service. The primary stakeholders for segment architecture are business owners and managers. Segment architecture is related to EA through three principles: structure, reuse, and alignment. First, segment architecture inherits the framework used by the EA, although it may be extended and specialized to meet the specific needs of a core mission area or common or shared service. Second, segment architecture reuses important assets defined at the enterprise level including: data; common business processes and investments; and applications and technologies. Third, segment architecture aligns with elements defined at the enterprise level, such as business strategies, mandates, standards, and performance measures.

## Nominal set of views

In search of "Framework for Modeling Space Systems Architectures" Peter Shames and Joseph Skipper (2006) defined a "nominal set of views", Derived from CCSDS RASDS, RM-ODP, ISO 10746 and compliant with IEEE 1471.

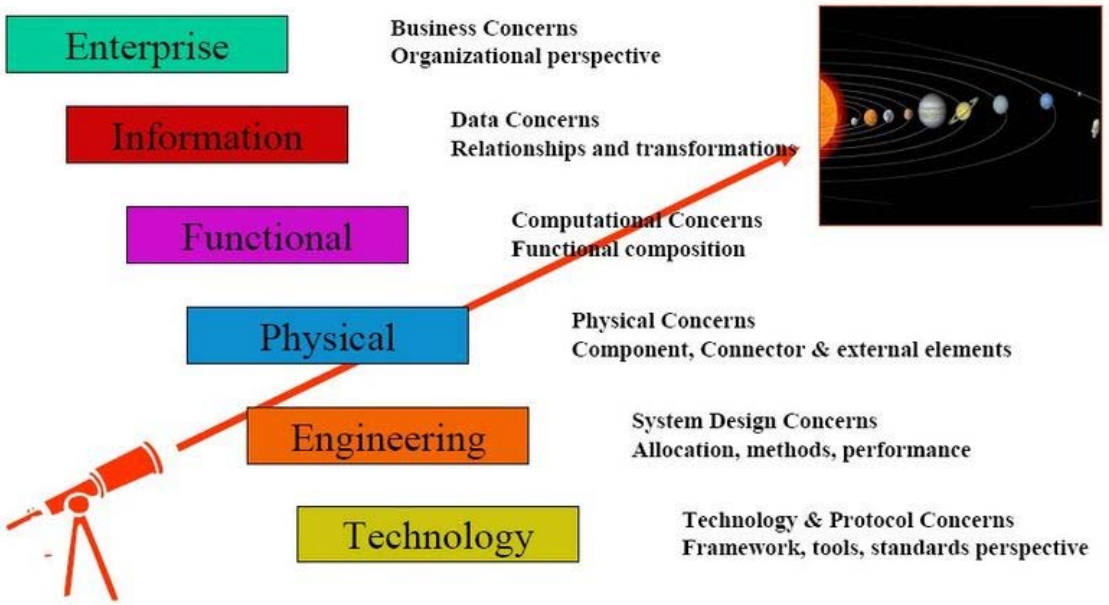



Illustration of the "Nominal set of views".

This "set of views", as described below, is a listing of possible modeling viewpoints. Not all of these views may be used for any one project and other views may be defined as necessary. Note that for some analyses elements from multiple viewpoints may be combined into a new view, possibly using a layered representation.

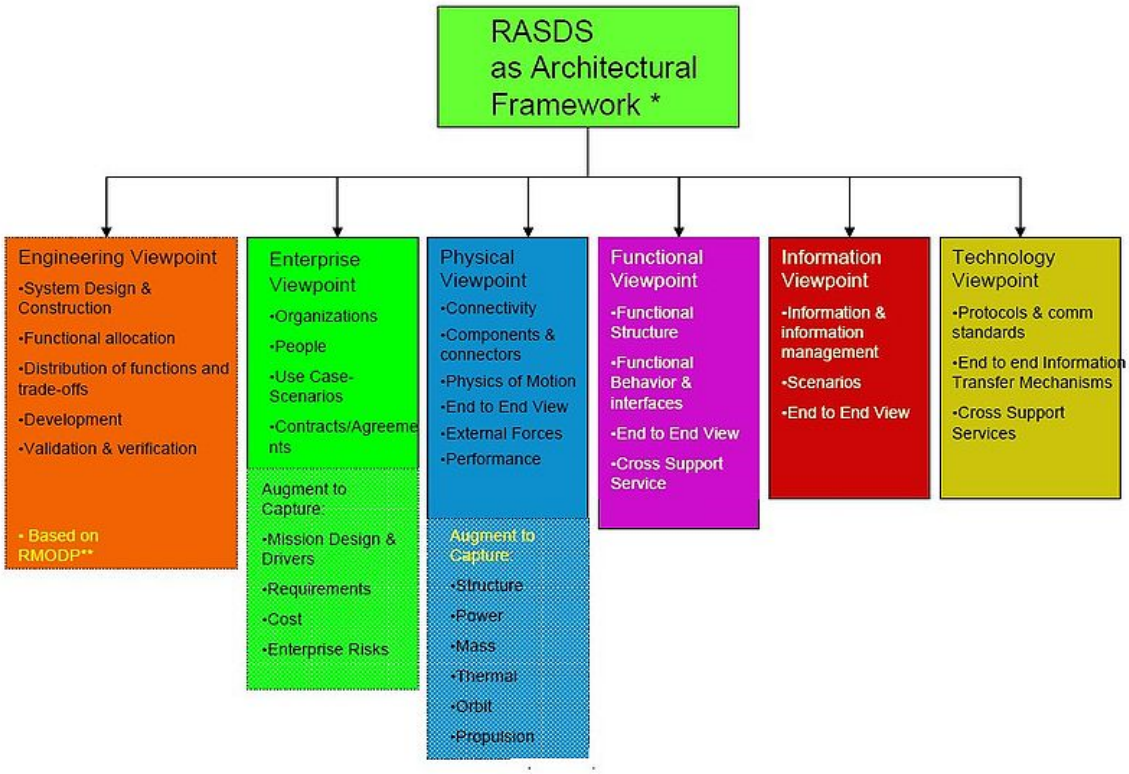
In a latter presentation this nominal set of views was presented as an Extended RASDS Semantic Information Model Derivation.

Enterprise Viewpoint

- Organization view – Includes organizational elements and their structures and relationships. May include agreements, contracts, policies and organizational interactions.
- Requirements view – Describes the requirements, goals, and objectives that drive the system. Says what the system must be able to do.
- Scenario view – Describes the way that the system is intended to be used, see scenario planning. Includes user views and descriptions of how the system is expected to behave.

Information viewpoint

- Metamodel view – An abstract view that defines information model elements and their structures and relationships. Defines the classes of data that are created and managed by the system and the data architecture.
- Information view – Describes the actual data and information as it is realized and manipulated within the system. Data elements are defined by the metamodel view and they are referred to by functional objects in other views.



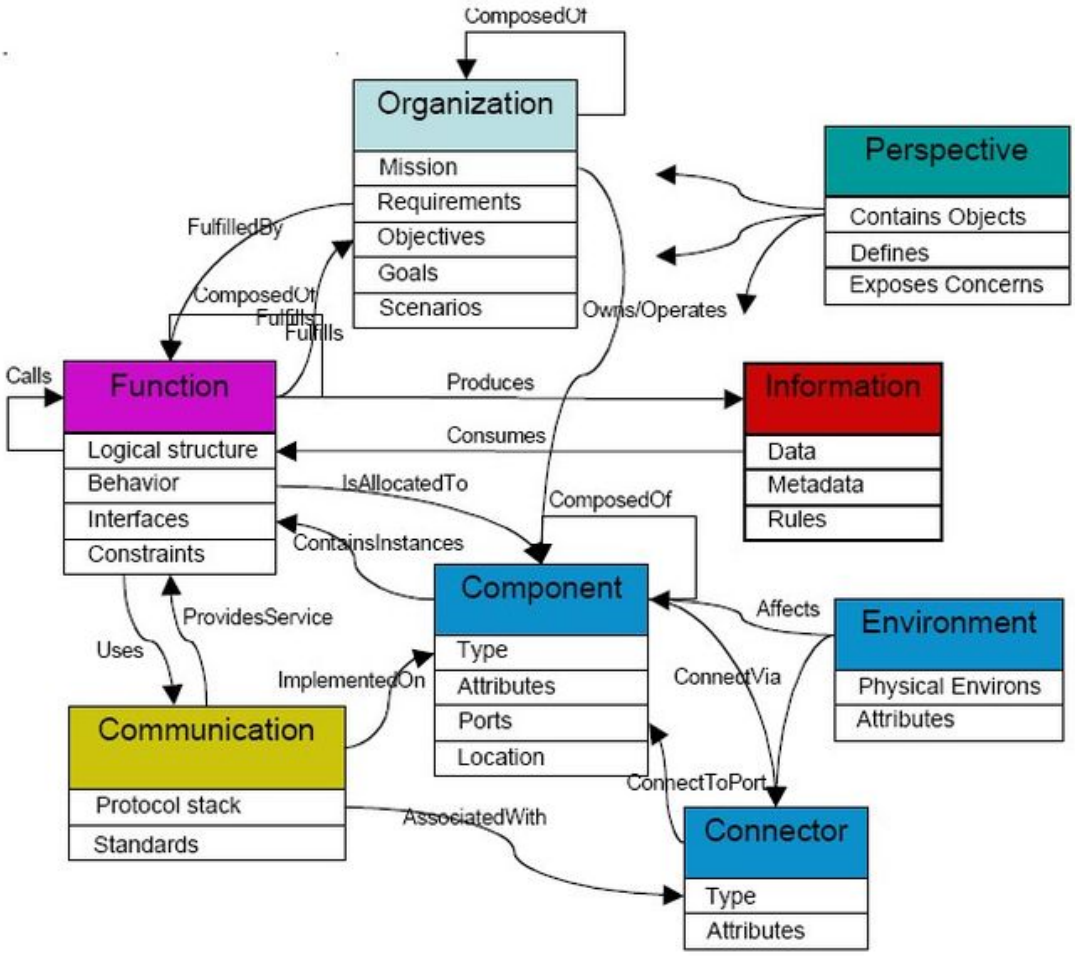

Reference Architecture for Space Data Systems.

Functional viewpoint

- Functional Dataflow view – An abstract view that describes the functional elements in the system, their interactions, behavior, provided services, constraints and data flows among them. Defines which functions the system is capable of performing, regardless of how these functions are actually implemented.
- Functional Control view – Describes the control flows and interactions among functional elements within the system. Includes overall system control interactions, interactions between control elements and sensor / effector elements and management interactions.

Physical viewpoint

- Data System view – Describes instruments, computers, and data storage components, their data system attributes and the communications connectors (busses, networks, point to point links) that are used in the system.
- Telecomm view – Describes the telecomm components (antenna, transceiver), their attributes and their connectors (RF or optical links).
- Navigation view – Describes the motion of the major elements within the system (trajectory, path, orbit), including their interaction with external elements and forces that are outside of the control of the system, but that must be modeled with it to understand system behavior (planets, asteroids, solar pressure, gravity)
- Structural view – Describes the structural components in the system (s/c bus, struts, panels, articulation), their physical attributes and connectors, along with the relevant structural aspects of other components (mass, stiffness, attachment)
- Thermal view – Describes the active and passive thermal components in the system (radiators, coolers, vents) and their connectors (physical and free space radiation) and attributes, along with the thermal properties of other components (i.e. antenna as sun shade)
- Power view – Describes the active and passive power components in the system (solar panels, batteries, RTGs) within the system and their connectors, along with the power properties of other components (data system and propulsion elements as power sinks and structural panels as grounding plane)
- Propulsion view – Describes the active and passive propulsion components in the system (thrusters, gyros, motors, wheels) within the system and their connectors, along with the propulsive properties of other components




MBED Top Level Ontology based on the Nominal set of views.

Engineering viewpoint

- Allocation view – Describes the allocation of functional objects to engineered physical and computational components within the system, permits analysis of performance and used to verify satisfaction of requirements
- Software view - Describes the software engineering aspects of the system, software design and implementation of functionality within software components, select languages and libraries to be used, define APIs, do the engineering of abstract functional objects into tangible software elements. Some functional elements, described using a software language, may actually be implemented as hardware (FPGA, ASIC)
- Hardware views – Describes the hardware engineering aspects of the system, hardware design, selection and implementation of all of the physical components to be assembled into the system. There may be many of these views, each specific to a different engineering discipline.

- Communications Protocol view – Describes the end to end design of the communications protocols and related data transport and data management services, shows the protocol stacks as they are implemented on each of the physical components of the system.
- Risk view – Describes the risks associated with the system design, processes, and technologies, assigns additional risk assessment attributes to other elements described in the architecture
- Control Engineering view - Analyzes system from the perspective of its controllability, allocation of elements into system under control and control system
- Integration and Test view – Looks at the system from the perspective of what must be done to assemble, integrate and test system and sub-systems, and assemblies. Includes verification of proper functionality, driven by scenarios, in satisfaction of requirements.
- IV&V view – independent validation and verification of functionality and proper operation of the system in satisfaction of requirements. Does system as designed and developed meet goals and objectives.

Technology viewpoint

- Standards view – Defines the standards to be adopted during design of the system (e.g. communication protocols, radiation tolerance, soldering). These are essentially constraints on the design and implementation processes.
- Infrastructure view – Defines the infrastructure elements that are to support the engineering, design, and fabrication process. May include data system elements (design repositories, frameworks, tools, networks) and hardware elements (chip fabrication, thermal vacuum facility, machine shop, RF testing lab)
- Technology Development & Assessment view – Includes description of technology development programs designed to produce algorithms or components that may be included in a system development project. Includes evaluation of properties of selected hardware and software components to determine if they are at a sufficient state of maturity to be adopted for the mission being designed.

In contrast to the previous listed view models, this "nominal set of views" lists a whole range of views, possible to develop powerful and extensible approaches for describing a general class of software intensive system architectures.