Artificial General Intelligence: Blueprint

Mykola Rabchevskiy

July 04, 2013

INTRODUCTION

This is a "blueprint" of the simple AGI system that provides the basic abilities of a General Artificial Intelligence (AGI). It also describes possible further extensions and specializations. The "Blue print" implements the approach described in an earlier paper [1].

1 SYSTEM ARCHITECTURE

1.1 Overview

Knowledge can be divided into **structural knowledge** (*X is Y*) and **temporal knowledge** (*X occurs, and then Y occurs*). Temporal knowledge is the starting point for detection of **cause-and-effect relationships**. Our approach is distinct from many other approaches (like Artificial Neural Networks) that do not provide explicit ways for representing and processing temporal knowledge. Our AGI architecture is based on the tight integration of the structural knowledge represented by a **knowledge graph** (described in [1], section 2) and the explicitly represented temporal knowledge stored in an **event sequence**. Such integration is a foundation for forecasting, learning and decision-making (Fig.1).

1.2 Embodiment: Sensors and Actuators

The term "General" in the acronym "AGI" means that the same system architecture can be used for performing different tasks and that it is compatible with different embodiments with different sets of sensors and actuators

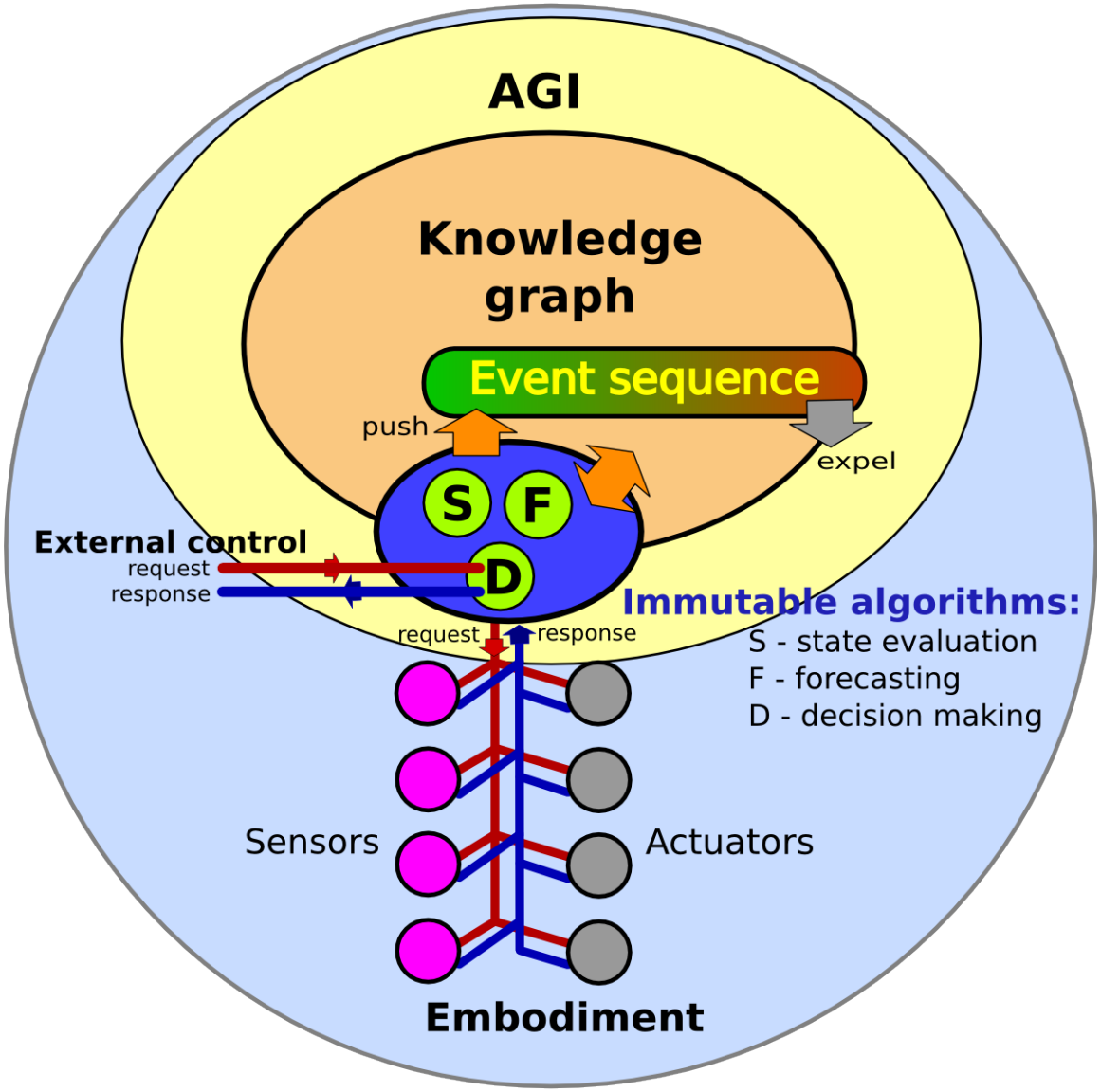

Figure 1: System architecture

(physical or virtual). To achieve this goal sensory output must be represented in a symbolic form as sequence of symbols. A symbol corresponds to an entity, i.e. symbol is a **reference to node of the knowledge graph**. Information must be requested from a sensor by sending it a proper sequence of symbols that describes appropriate details.

Some processing of raw sensor values is required to convert sensor output into symbolic form. The simplest variant of such conversion is a discretization of continual values so each symbol represents some possible level. More complicated preprocessing may include numerical stages (filtering, integration, differentiation etc.) and logical stages (feature detection).

An actuator has the same interface: it gets a request for action as sequence of symbols and responds with a sequence of symbols that provides some information about the performed action.

An actuator can be a physical device as well as some algorithm (routine) executed on demand. Specialized "narrow AI" algorithms (like SLAM, activity planning etc.) can be integrated into AGI system as soft actuators. So interaction between the "brain" and the "body" is implemented as exchange of messages in which each message is a sequence of symbols that refers to the entities stored in the knowledge graph. The difference between sensors and actuators is quite conventional: both should be requested to get some info as a response.

1.3 Knowledge Graph and Event Sequence

The AGI design is based on using the **knowledge graph** for storing **structural knowledge** as described in [1], section 2. This approach incorporates the abilities that are provided by semantic nets, anthologies, rule based systems and predicate based systems.

Past history (experience) is backbone for discovering cause-and-effect relationships and of the ability for learning as a whole. Our design is based on the collection of the experience in an **event sequence** that is permanently extended by pushing information about current system state, performed actions and received sensor data. Events sequence collect all the available information as temporal sequence of a symbols. A reference to a concept that represents a set of entities can also be used as an element of the history sequence (for example to represent a set of simultaneous events).

The longer is the past history stored in an event sequence, the smarter the

system can be, but resources of real system are limited. An obvious way to increase size of the stored experience is to use some kind of data compression. A common way to compress data is to find a repeated subsequence (aka **pattern**) in the sequence, to then store such patterns in a separate place (the knowledge graph) and to replace all matched subsequences by references to the corresponding patterns. On the other hand discovering of the cause-and-effect relationships is based on searching repeated "cause-effect" subsequences, so the detection of repeated patterns lets us kill two birds with one stone: to increase system experience capacity and to provide a way to discover cause-and-effect relationships. The knowledge graph is used to store discovered patterns (among of another knowledge).

Any newly discovered temporal pattern is represented as a combination of previously known entities. A hierarchy of a temporal patterns is represented by a binary tree (kind of *pattern ontology*) where each of two child elements (*head* and *tail* respectively) can be another pattern or a single symbol (aka **tree leaf**). A pattern which consists of more than two elements can be represented by more than one tree, so the knowledge graph keeps all the actual representations of a particular pattern that can be composed using fragments of event sequence stored in the knowledge graph. In the knowledge graph each pattern bonded to a set of **pattern representations** where each element of this set is a root of a binary tree (that in turn is a subgraph of the knowledge graph).

The history sequence is updated permanently, which is different from "classic" data compression. When the next symbol is pushed into the sequence, the last two added symbols are checked to see if they are a representation of some known pattern. If so then this pair is replaced by a reference to the known pattern, and new pair of last symbols is checked again (so *cascade updating* is possible). In the case when such **convolution** is not possible the whole event sequence is checked for two consecutive elements that are identical to the pair of last pushed symbols; if such a pair is discovered, a new pattern will be composed and both pairs are replaced by references to the newly created pattern.

Despite the compression the event sequence will grow permanently. To conform to the memory limitation the event sequence can just be truncated on the oldest side so most outdated information become forgotten. Another way to maintain the size of memory used by the remembered history is to create a new pattern that is a generalization of a few existing patterns, then make related substitution in the sequence. This kind of compaction means

losing some details, i.e. such replacing should first be applied to oldest part of sequence. Pattern generalization can be done in a many ways, for example:

- A symbol in a pattern that refers discretized data can be replaced by another symbol that refers the same value but in a coarser scale; this creates new concept of a *coarse* pattern that generalizes a few *fine* patterns
- A set of patterns which represents the different history fragments that finished with the same final effect can be combined into the new generalized concept

1.4 Forecasting and decision-making

After event sequence are updated and compacted, forecasts can be composed. Forecasting is based on simple ideas:

- if something happened in the past it may happen in the future;
- a possible future consists of a few variants with different probabilities;
- the future may depend on performed action or inaction.

The forecast is constructed as a tree of possible future events. Such a tree is constructed using overlapping temporal patterns from the past history; the starting point is the last symbol of the event sequence. When a situation is totally new, forecast tree will be empty (Fig.2). Two kinds of symbol matching are used when patterns are checked for overlapping:

- symbols are identical (refers the same concept)
- one symbol refers to a generalization of the concept referred by other (*orange* matches to *warm color*, *car* matches to *vehicle*).

For each variant at the fork point we can obtain the frequency of such pattern in the history sequence. These frequencies are used for estimation of a relative probability for branches starting at particular fork point. Each node and leaf of the forecast tree becomes labelled by the estimation of its probability.

The forecast three is potentially infinite, so implementation of the forecast composition algorithm should use a limits for the size of the forecast three

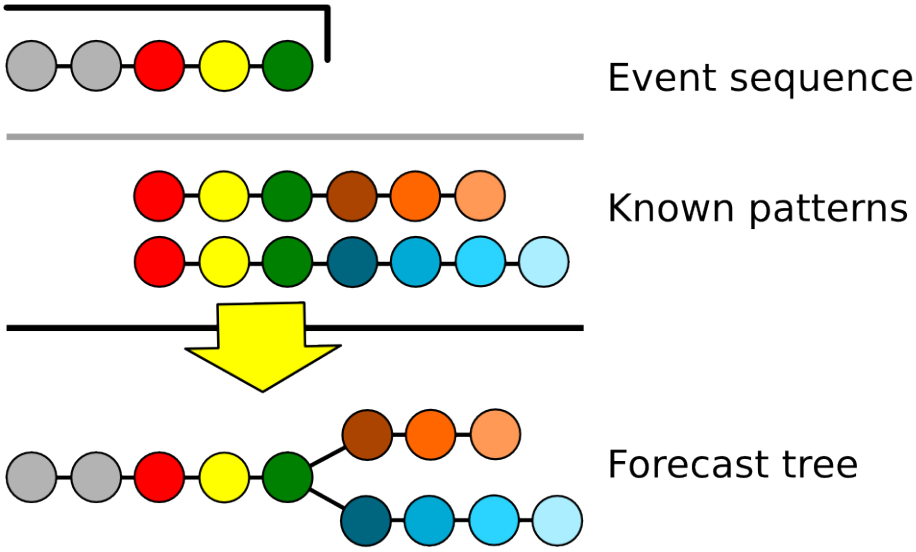

Figure 2: Forecast tree composition

as well as a limit of the time that can be elapsed for composition (there are two design parameters). In the case where the time limitation is reached first, a stressful situation has occurred: a shortened forecast tree will lead to a lower quality decision.

Decision-making, using the forecast tree, is based on an assumption that actions represented in such a tree are followed by caused system states. So the first step of decision making is to prepare a table that contains the probability estimation for each combination of action (or inaction) followed by most distant in time effect (system state). If some path of the forecast tree doesn't start with action then inaction is assumed. Some paths of a forecasting tree can contain *a priory* unacceptable situation; actions that lead to such states can be excluded from consideration as *a priory* wrong.

When such a table is composed then a choice of the most appropriate action (or inaction) becomes a well-defined multi-criteria optimization problem. Obviously a set of optimization criteria should be defined dependent on the embodiment and system mission. A common approach assumes single main criterion tightly connected to the system mission and a few restriction criteria which reflect physical restrictions, security requirements and so on. The set of optimization criteria should be immutable to be sure that system behaviour follow the system requirements; they can be hard-coded as part of decision-making unit or stored in the knowledge graph as immutable entities.

In the case of a complicated system, the values of criteria parameters will depend on the current state (this includes body state, environment state, quality of forecasting for different past periods and so on), so the relative weights of criteria will be different in different cases.

To provide world **exploration behaviour** the system uses **random action selection** side by side with optimal action selection. Obvious cases where random selection is useful are cases where forecast tree is empty (a totally new situation) and cases where an optimal solution is unavailable (i.e. no acceptable solution based on current experience). But random action selection can also be used in "ordinary" cases to provide exploratory behaviour as way to increase the intensity of the self-learning and to increase the system adaptivity (by inevitable reduction of the predictability). Related parameters and/or rules can be hard-coded in the decision-making unit or stored in the knowledge graph as immutable information.

1.5 Hierarchical design

The architecture of a complex AGI system can use simple AGI systems as nodes of the hierarchical structure. Each subsystem contains its own event sequence, forecasting and decision-making units, but nodes above the lowest level are connected to child AGI subsystems instead of sensors and actuators, so higher level nodes have no direct access to the sensors or actuators (Fig.3).

Child nodes of a particular parent node work autonomously most of the time but a parent node can override the decision made by child node by sending request for some action (so child node play the role of a *soft actuator* for the parent node). The action forced by the parent node is remembered (with related consequences) in a child's event sequence. Any extra functionality related to forecasting and decision-making apart from ability to override decision made on the child level by such one made by a parent module is not required. The experience that is forced by a parent node is the used as part of the whole child node experience, so there is a kind of *teaching* where the parent node (or the operator in case of top node) forces the child node to obtain a new experience. Repeating such exercises, with produces a positive effect (by changing related statistics used at decision-making), is a way of creating stable sequences of actions (kind of *conditioned reflex*) that can be triggered by single initial command from the parent node: the followed action sequence is produced "automatically" by the above described decision-making algorithm.

2 HOW IT WORKS

2.1 Main cycle

Each system node infinitely repeat "main loop" that includes the followed actions:

- Querying the sensors or child nodes
- Evaluating the system state and pushing the result into the event sequence
- Composing the forecast
- Making a decision
- If some action is selected then push it into the event sequence, execute it and push feedback (response) into the event sequence

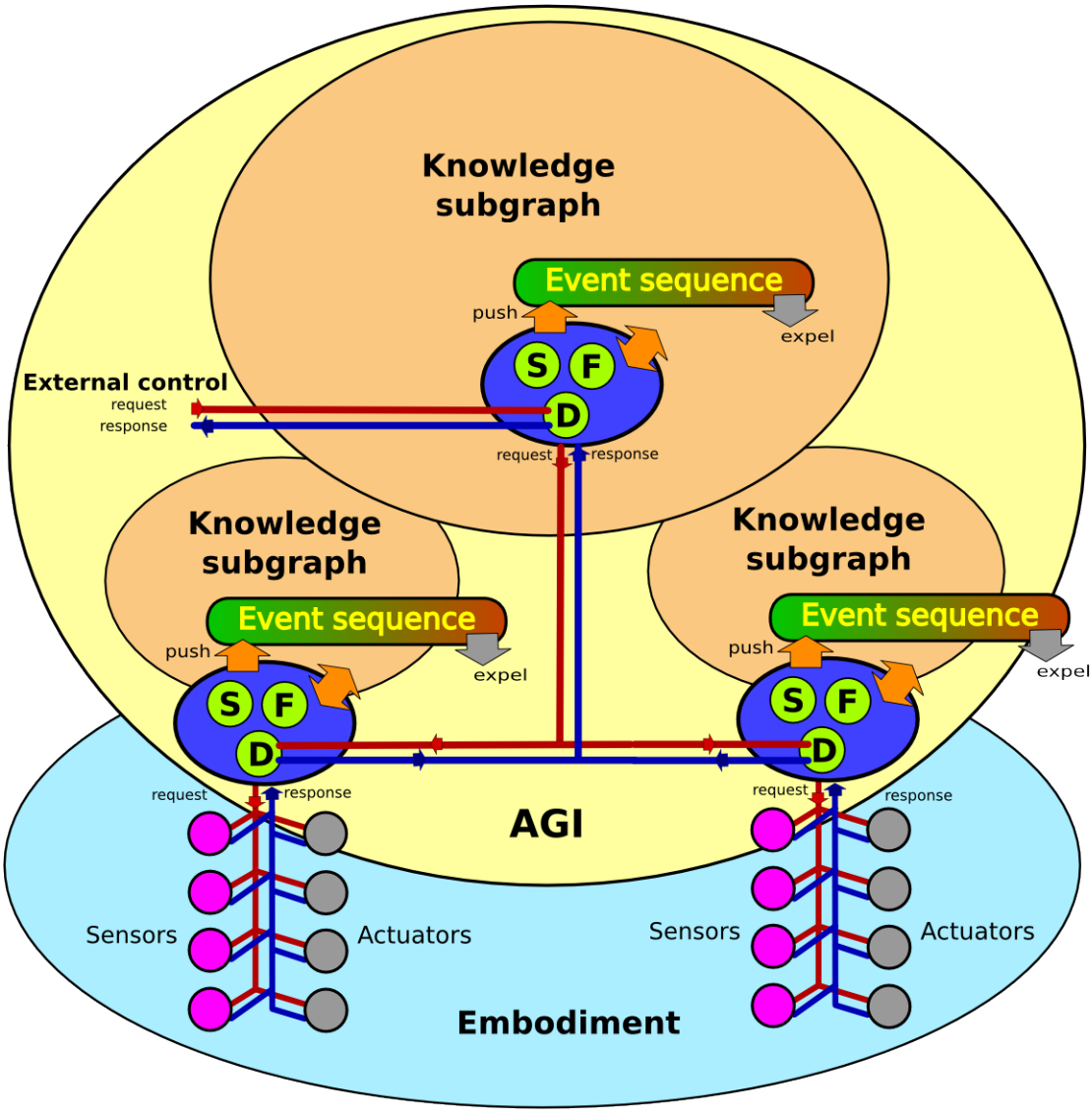

Figure 3: Hierarchical design

Each symbol pushed into the event sequence initiates compression/generalization as described above; sequence truncation is performed in case where event sequence reaches the predefined limit.

The action performed by *soft actuators* can modify the mutable part of the knowledge graph as result of action along with response stored into event sequence.

If the size of the knowledge graph has reached a predefined limit then entities that are less usable and/or are not presented in the event sequences should be discarded. The system design should define the criteria for the evaluation of an entity's usability; such criteria can significantly affect the behaviour of the system.

2.2 System life cycle and learning modes

The initial period of the "system life" is a time of intensive discovering its own abilities and the agile expanding of its collected experience. The system behaviour is obviously very stochastic (less predictable) at this time.

Learning time can be decreased in two ways:

- Preloading of knowledge collected by another exemplar of the system. The knowledge graph (or some subgraph) can be extracted from an experienced AGI system and loaded as extended *congenital knowledge* into a newly created system.
- The system can be taught, when decision made by system is overridden using remote operator intervention (*"do what I tell"* mode) as described above. The experiences collected at such teaching (performed actions and followed results) can then be used by the system in an autonomous mode.

Note that the last approach significantly distinct from the "classic" hard-coded programming of industrial robots; the AGI system don't reproduce predefined action sequence, but rather, combines fragments of many demonstrated scenarios with autonomously discovered acting patterns (*"creative behaviour"*). This mode is also distinct from the "classic" supervised learning in which the evaluation of an action result ("good"/"bad") is performed externally; in our case this evaluation is performed internally in the same way as in an autonomous mode. So our approach is aimed at expanding

the collected knowledge, not to form classification rules nor create some immutable behaviour.

After an initial period, in a stable environment, the system behaviour become more predictable, more successful and less exploratory. But changes in the environment automatically reactivate the *exploratory behaviour* as response to decreased forecasting quality and increased uncertainty.

The intensity of the exploratory behaviour as a whole is one of design parameters. A particular implementation of the decision-making module is a result of trade-off between the fast response to changes in the environment, in a combination with less predictable behaviour (*fluid* system character), and the more predictable behaviour in a combination with the slower response to changes in the environment (*conservative* system temper).

3 DISTINCTIVE FEATURES

- Structural knowledge is represented by the directed graph with unlabelled edges. The knowledge graph can incorporate knowledge expressed using many known technologies (semantic nets, anthologies, rules, predicates)
- Temporal knowledge are represented explicitly
- All mutable information is stored into the knowledge graph and the event sequence; the forecasting and decision-making algorithms are immutable
- A binary hierarchy of temporal patterns is used for detecting of cause-and-effect relationships, for the forecasting and for compressing of the past experience information
- The event sequence that keeps temporal knowledge also works as kind of *knowledge processing conveyor*: one side accepts symbols of a current input that are processed while moved to other side where they are expelled
- The system combines two learning modes: the permanent self-learning and the teaching mode that is initiated by an external operator or parent subsystem
- System' customization may be done by alteration of the preloaded (congenital) knowledge and by modification of parameters or/and code

of two immutable algorithmic unit (the decision-making unit and the system state evaluation unit)

- A complex system can be composed as a hierarchy of AGI cores that have identical structure but different capacity and different preloaded knowledge: the parent core plays the role of the supervisor of a few child cores and the child cores play the role of the intelligent sensor or actuator

- Any sensor data preprocessing (numerical preprocessing, discretization and feature extraction and so on) is assumed to belong to the sensor

4 ADVANTAGES

- The collected knowledge is totally explicable, it can be presented in a human readable form, can be extracted from the knowledge graph and can be added using a control channel

- The forecasting is totally explicable

- The decision-making is explicable and traceable

- The quality of forecasting and decision-making can be improved by increasing available computational resources (available memory and processor performance)

- The system can learn to avoid an unwanted situation after just a couple occurrences (this is impossible in case of statistical learning) because decision-making is based on using an explicitly expressed experience

- Learning efficiency can be improved using teaching mode

- The decision-making is finally reduced to well-developed multi-criteria optimization, so a correspondent system unit may incorporate existing optimization algorithms

- Knowledge forgetting is fully controlled so acquisition of new knowledge can't affect the previously collected experience in an unpredictable manner

- The trade-off between contradicting requirements for system behaviour (like adaptivity versus predictability) are explicit design parameters

- Implementations of "narrow AI" solutions can be integrated into system as *soft actuators*
- The hierarchical architecture provides a natural way to computing parallelization
- Any kind of sensor can be incorporated when response is provided in a symbolic form
- Immutability of the core algorithmic unit (system state evaluation, decision-making units) as well as the ability to protect any part of knowledge graph from modification eliminates the risk of unacceptable system behaviour
- The system architecture is really general: it can be used with different embodiments and different system missions

5 ANALOGIES WITH THE NATURAL BRAIN

Development of the described AGI architecture is not a result of an attempt to simulate or to clone the natural brain. Rather it is a result of expanding abilities of known control systems. However, some analogies with natural brain architecture can be found:

- The knowledge graph may be considered as analogue of the cortex
- The event sequence and immutable algorithms can be considered as analogue of lower sections of the brain
- Generalized state parameters can be considered as emotions
- Behaviour of the hierarchical AGI system can be considered as analogue of the consciousness

6 FIELDS FOR FUTURE RESEARCH

- Development of a technique for converting of two-dimensional sensor output (vision, tactile sensors) into symbolic scene description
- Comparative analysis of different ways to provide generalization of concepts in the AGI system
- Development of the technique of natural language incorporation

Acknowledgement

The author wish to thank Brett N. Martensen for his efforts to make the text more readable.

REFERENCES

1. Mykola Rabchevskiy. Artificial General Intelligence: Engineering approach. 2010.