1. Introduction

Engineering is the art of applying available technologies to arbitrary specifications in attempt to implement them as products. Implementation may be viewed as mapping concepts from a specification domain onto a technology domain. For example, a specification of a banking system is written in a language whose vocabulary contains clients, statements, transactions, branches etc. and is expected to be implemented in a database technology based on files, records, updates, reports etc. This same technology is also expected to implement a hospital management system with a significantly disjoint terminology, including patients, medical notes, laboratory tests and results, departments and more. A skilled systems analyst is capable of associating clients (or patients) with records, statements (or medical notes) with reports, and lab results (or transactions) with updates, as illustrated in Figure 1.

![Figure 1: Disjoint specifications mapped onto the same technology](image-url)
Implementation is the junction where knowledge resources meet and amalgamate into a product. It may be too much to expect engineering to be fully automated, however, knowledge resources may be computerized and inference methods are continuously developed. Therefore the representation of engineering as a formalized and computable process may lead towards the development of sophisticated and intelligent engineering tools.

In this paper we give a logical view on implementation. We use first order logic as the basis to our formalization but this basis is extended to host more sophisticated logical constructs and inference mechanisms. In section 2 we describe our knowledge based representation of the implementation process. Section 3 gives our logical view on implementation. Section 4 introduces inheritance relations between objects as one specific realization of the proposed logic. Section 5 contains conclusions and future work.

2. Knowledge Based Implementation

2.1. Glossary
In order to give a clear representation we will first introduce the following glossary.

A product is a feature required by a client. Initially the product is hypothetical and needs to undergo a process of implementation for realizing it in more concrete means. The implementation is done by an engineer. A specification is a descriptive theory about the product, whether hypothetical or real. The initial theory, describing the product prior to its actual existence, is the initial specification and the theory describing the final product is the final specification. The client's world—the domain in which the initial specification is described—is the application domain, which is regarded as the knowledge source of the client. The source of knowledge available to the engineer is the engineering domain or the technology domain.

Specifications, as theories, take different forms and may be formalized in a variety of models. Most of these models are used for software engineering and among them are logic (e.g. Turski [8]), algebra (e.g. Burstall & Goguen [3]), semantic networks (e.g. Bordiga et al [2]), set theory (e.g. Sorensen [5]) and graphical representation (e.g. Ross [4]). However, there are characteristics common to all forms, and therefore we extend our glossary with several more technical terms (rather intuitive than formal):

A language is a syntactic system in which sentences can be constructed. Consequence is a relation associating sentences in the language with sets of other sentences: When a sentence s is related by consequence to a subset S it is said that s is a consequence of S. A theory is any set of sentences in the language which is closed under consequence, i.e. every consequence from sentences in the theory is also in the theory. A theory is usually specified by a kernel set of sentences called axioms and the theory as a whole is the consequence-closure of its axioms. Th(A) denotes the theory based on a set of axioms A.
2.2. Knowledge Resources and their relationships

Upon completion of an implementation process the product has two specifications (according to the above definitions): the (hypothetical) initial specification and the ("real") final specification (denoted as IS and FS respectively). It will be easier to refer to these two theories if we assume that they are expressed in the same language. This is not a straightforward assumption, since implementation, in many cases, involves translation or interpretation. However, we claim that FS is not just an interpretation of IS, and if the interpretation component is isolated then the other concerns can be highlighted more easily. If an interpretation did take place then the common language is the one shared by FS and the transformed IS.

If the product is satisfactorily accepted by the client it is possible to claim that FS IS, i.e. the final specification contains at least all the information contained in the initial one. The "added value" to the specification (FS-IS) is a direct result of the implementation and may be considered as the set of technological contributions. The initial specification originates in the client's application domain (AD), whereas the technological contributions come from the technology domain (TD) (Figure 2). For practicality we restrict these domains to contain only the knowledge relevant to the concerned product. The intersection AD\cap TD is regarded as the common knowledge.

![Figure 2: Knowledge distribution in the final product specification](image)

3. First Order Logic Formalization of Implementation

In the following we formalize our concepts in first order logic.
Definition 1 (properties and requirements)

An *(initial) specification* is a theory $S$, where

1) $S = \text{Th}(P \cup R)$

2) For every sentence $s$ of $S$: if $s \in \text{Th}(P)$
   
   then $s$ is a *property*

   otherwise $s$ is a *requirement*.

Informally, $P$ is a set of initial *explicit* properties, and any consequence based on these properties *alone* is regarded also as a property. $R$ is an additional set of *explicit* requirements, and any consequence inferred using a requirement is also regarded as a requirement.

Definition 2 (implementation)

Let $S$ be an initial specification and let $F$ be a final specification (in the same language), if $S \subseteq F$ then $F$ is called an *implementation* of $S$.

At this first approximation we declare, in fact, that every alternative specification of the product is acceptable as an implementation as long as it can provide *at least* all the consequences inferable from the initial specification. The main practical difference between $S$ and $F$ is that in the former requirements are restricted to be *explicit axioms* whereas in the latter they may be obtained as *consequences* of the theory.

Definition 3 (Technology & Implementability)

Let $S = \text{Th}(P \cup R)$ be a specification and let $\text{Th}(T)$ be another theory (over the same language). If $\text{Th}(P \cup T)$ is an implementation of $S$ then $\text{Th}(T)$ is called a *technology* (in which $S$ is *implementable*).

The above is better explained by an example.

Example 1

Let $N_1, \ldots, N_4$ be four nodes through which a communication network should be constructed. The network requires that every node can communicate with all others. Let $\text{node}(x)$ be the predicate stating that $x$ is a node and let $\text{comm}(x,y)$ denote that the nodes $x$ and $y$ can communicate. The specification, then, is written as the following set of sentences ($S_1$ through $S_4$, and $R_1$):

$S_1$: \hspace{1em} \text{node}(N_1)$

$S_2$: \hspace{1em} \text{node}(N_2)$

$S_3$: \hspace{1em} \text{node}(N_3)$

$S_4$: \hspace{1em} \text{node}(N_4)$

$R_1$: $(\forall x)(\forall y) (\text{node}(x) \land \text{node}(y) \rightarrow \text{comm}(x,y))$
If all these sentences were axioms then this theory would satisfy the requirements. However, from the client's point of view, S1 through S4 are facts, whereas R1 is the requirement that should be obtained by means outside this theory. In other words, the client requires R1 to be a consequence of other axioms, rather than an axiom itself. To obtain this the theory may be enriched with more axioms. One possible solution is to organize the nodes in a star architecture (Figure 4(a)), with one node (N1) as the concentrator which is physically (bi-directionally) linked to every other node. Let link(x,y) denote the existence of a link from x to y, then the network is specified in the following sentences (T1 to T7):

T1: link(N1,N2) [physical links]
T2: link(N1,N3)
T3: link(N1,N4)
T4: (∀x)(∀y)(link(x,y) → comm(x,y)) [communication over links]
T5: (∀x)link(x,x) [link reflexivity]
T6: (∀x)(∀y)(link(x,y) → link(y,x)) [symmetry of bi-directional links]
T7: (∀x)(∀y)(∀z) (comm(x,y) ∧ link(y,z) → comm(x,z)) [transitive communication]

It is clear now that R1 is a consequence of {S1,...,S4}∪{T1,...,T7}. In our application context the set S={S1,...,S4} represent the properties of the specification, R={R1} is the set of requirements and T={T1,...,T7} is the technology which enables the requirements to be satisfied.

(a) star   (b) ring

Figure 3: Two communication network architectures

Another implementation to the same specification and the same requirement can be achieved using a (unidirectional) ring architecture (Figure 4(b)). The relevant technology is expressed by T′={T1′,...,T7′} as follows:
This time there are four physical links, however the complete communication is obtained using transitive communication. Here, again, the requirement R1 is a consequence of S∪T' and thus T' is a candidate technology for implementing R given S.

One point should still be clarified: by Definition 1 there are two types of requirements: explicit ones (axioms) and implicit ones (consequences). When formal implementation is considered it would be impractical if we had to prove that all the requirements are satisfied, both explicit and implicit. However it is fairly trivial to show that it is sufficient to satisfy the explicit requirements. This is stated by the following theorem, whose formal proof may be found in Tomer [6].

**Theorem**

Let S=Th(P∪R) be a specification and let T be a set of axioms.

If, for every r∈R, r is a consequence of P∪T then

- \(\text{Th}(P∪T)\) is an implementation for S, and
- \(\text{Th}(T)\) is a technology in which S is implementable.

The above theorem suggests a methodological approach towards automatic implementation: to substitute the set of requirements by another set of axioms (the technology). However, it gives no practical means to select or construct such a technology. In the following we introduce one particular constructive approach, based on inheritance relations.

## 4. Implementation by Constructive Inheritance

Inheritance relations play an important role in object-oriented design (see, for example, Blair et al [1]). The intuitive meaning of the inheritance relation between two objects \(α\) and \(β\) (often expressed as "\(α\) is a \(β\)") is that \(α\) has all the properties of \(β\), in addition to some other properties of its own. In our approach we use a simple form of inheritance between objects. If objects \(α\) and \(β\) are represented by first-order theories A and B, respectively, then the statement "\(α\) is a \(β\)" means \(A ⊇ B\). If the inheritance relation is determined in advance then the properties of \(α\) may be derived from the properties of \(β\). However, in the engineering context, we need to implement the object \(α\) according to the required properties. We attempt to do so by searching for
another object $\beta$ from which $\alpha$ may inherit these properties. When such an object is found we simply construct the relation “$\alpha$ is a $\beta$”. The mechanism of searching for objects and constructing the inheritance relation is noted here as constructive inheritance. We will give here no formal definition of constructive inheritance—a detailed formalization and discussion may be found in Tomer [6] and in Tomer & Hogger [7]. Instead we will revisit an extension of Example 1.

Example 1'

Both types of communication networks (star and ring) are instances of a general network (GNW) object, possessing the following properties:

- **GNW1**: $(\forall x)\text{link}(x,x)$ \hspace{1cm} [link reflexivity]
- **GNW2**: $(\forall x)(\forall y)(\text{link}(x,y) \rightarrow \text{comm}(x,y))$ \hspace{1cm} [communication over links]
- **GNW3**: $(\forall x)(\forall y)(\forall z) (\text{comm}(x,y) \land \text{link}(y,z) \rightarrow \text{comm}(x,z))$ \hspace{1cm} [transitive communication]
- **GNW4**: $(\forall x)((\forall y)\text{link}(x,y) \rightarrow \text{master}(x))$

The last axiom is a new one, defining a master as a node that is directly linked to every other node. By stating that “a star is a general network” and “a ring is a general network” we let star and ring inherit (and thus possess) these axioms, i.e. each one of these specific theories already contains the axioms \{GNW1,...,GNW4\}. In addition each architecture has got its own specific axioms, as follows:

- **STAR1**: $\text{link}(N1,N2)$ \hspace{1cm} [physical links]
- **STAR2**: $\text{link}(N1,N3)$
- **STAR3**: $\text{link}(N1,N4)$
- **STAR4**: $(\forall x)((\forall y)\text{link}(x,y) \rightarrow \text{link}(y,x))$ \hspace{1cm} [symmetry of bi-directional links]

- **RING1**: $\text{link}(N1,N2)$ \hspace{1cm} [physical links]
- **RING2**: $\text{link}(N2,N3)$
- **RING3**: $\text{link}(N3,N4)$
- **RING4**: $\text{link}(N4,N1)$

In order to deal with more complicated cases we introduce another type of communication architecture—a tree (Figure 5)—which is also an instance of general network. The tree architecture is specified by the following axioms:

- **TREE1**: $\text{link}(N1,N2)$ \hspace{1cm} [physical links]
- **TREE2**: $\text{link}(N1,N3)$
- **TREE3**: $\text{link}(N1,N4)$
The specification of the site, as before, contains the definition of the nodes:

SITE1: node(N1)
SITE2: node(N2)
SITE3: node(N3)
SITE4: node(N4)

This time, we specify two requirements: the first one, as before, is that all the nodes communicate with each other, and the second one is the existence of a master node.

REQ1: \((\forall x)(\forall y) (\text{node}(x) \land \text{node}(y) \rightarrow \text{comm}(x,y))\)

REQ2: \((\exists x)(\text{node}(x) \land \text{master}(x))\)

We may try now to satisfy the requirements by letting the object site inherit any of the objects ring, star and tree (one at a time) and try to prove the theorems REQ1 and REQ2. The following table summarizes the results:

<table>
<thead>
<tr>
<th>inheritance</th>
<th>REQ1</th>
<th>REQ2</th>
</tr>
</thead>
<tbody>
<tr>
<td>site is a star</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>site is a ring</td>
<td>√</td>
<td>X</td>
</tr>
<tr>
<td>site is a tree</td>
<td>X</td>
<td>√</td>
</tr>
</tbody>
</table>

Thus it is shown here that in order to satisfy both requirements the communication network at the site must be implemented in a star architecture.

5. Conclusions and Future Work

In this paper we introduced a logic representation of knowledge-based engineering and suggested one practical approach—constructive inheritance—which implements the theory. The general formalization, as was presented in section 3 above, is general and schematic and may be interpreted and implemented in various ways. The constructive inheritances scheme, that was
Artificial Intelligence in Engineering 245

introduced in section 3, is just an example to how the general concepts may be applied in various contexts.

Future work should introduce alternative implementation schemes, together with other proposed application contexts. It is also suggested to investigate other engineering contexts that are more complicated than the simple mapping of section 1 above.

We believe that there are engineering fields which are, better than others, candidates for automated knowledge-based engineering. A deep case study in one of these fields can yield good understanding of the subject and lead to future research directions.

6. References


