This is to certify that the

dissertation entitled

An Approach to the Development of Intelligent Systems through Integrating Function-Based Reasoning with the Generic Tasks Methodology

presented by

Oleg Yurievich Lukibanov

has been accepted towards fulfillment of the requirements for

Doctor of Philosophy degree in Computer Science and Engineering

Major professor

Date 8/2/2000

AN APPROACH TO
THROUGH INTEGR
C

AN APPROACH TO THE DEVELOPMENT OF INTELLIGENT SYSTEMS THROUGH INTEGRATING FUNCTION-BASED REASONING WITH THE GENERIC TASKS METHODOLOGY

By

Oleg Yurievich Lukibanov

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Computer Science and Engineering

2000

AN APPROACH TO TH
THROUGH INTEGRAT
GEN

The research in K

three different schools

number of useful meth

However, these methods

complex problems requ

contributing to the overa

Integrated Knowledge-B

This dissertation

that could be used for t

core of the approach is

decomposition of a mo

relations the functional

This dissertation

traditional function-bas

methodology to the

ABSTRACT

AN APPROACH TO THE DEVELOPMENT OF INTELLIGENT SYSTEMS THROUGH INTEGRATING FUNCTION-BASED REASONING WITH THE GENERIC TASKS METHODOLOGY

By

Oleg Yurievich Lukibanov

The research in knowledge-based systems have advanced to a degree where different schools of thought have developed as well as described a number of useful methods that solve various kinds of problems. However powerful these methods are they cannot solve complex problems working alone. Complex problems require the cooperation of a number of problem-solvers each contributing to the overall solution. This dissertation refers to such systems as Integrated Knowledge-Based Systems (I-KBS).

This dissertation research defines a methodology, language, and tool set that could be used for the description of integrated architectures of I-KBS. The core of the approach is a modified function-based reasoning theory that enables decomposition of a modeled entity, its functionality, and a causal network that explains the functionality of the overall unit.

This dissertation describes the changes that have been made to the traditional function-based reasoning theory and application of the proposed methodology to the development of a shell for constructing integrated

nowledge-based systems

ctionality of S-Force is th

nowledge-based system.

The dissertation de

and S-Force to re-design m

systems for generating m

assemblies made of polym

In the conclusion,

other methodologies that

proposes a hypothesis f

development of complex.

knowledge-based systems – S-Force. The description of the features and functionality of S-Force is then given on an example construction of an integrated knowledge-based system.

The dissertation describes the application of the proposed methodology and S-Force to re-designing the knowledge-based core of the KBS Socharis – a systems for generating multiple conceptual manufacturing plans for mechanical assemblies made of polymer composite materials.

In the conclusion, this dissertation compares the developed approach to other methodologies that promote cooperation between multiple software entities and poses a hypothesis that the reported research may serve as a basis for the development of complex, reusable integrated problem-solving architectures.

Part of the resear

RCED program (ONR N

tiative NSF MIP-942-035

94-2763-613224).

I would like to than

encouragement during my d

Dr. Stoken this dissertat

I indebted to my co

discussions we had over the

Robert Hawkins and Dr. T

I most thankful to

support and help I would b

ACKNOWLEDGMENTS

Part of the research reported here has been supported by DARPA RADEO program (ONR N00014-96-1-0678), by the NSF Rapid Prototyping Initiative (NSF MIP-942-0351), and the Technology Reinvestment Program (eee-9412783 613224).

I would like to thank my advisor, Dr. Sticklen for his support and encouragement during my doctoral studies. Without stimulating discussions with Dr. Sticklen this dissertation would have not been possible.

I indebted to my colleagues from the Intelligent Systems Laboratory for debates we had over the ideas behind this research. I especially thankful to Mr. Robert Hawkins and Dr. Timothy Lenz.

I most thankful to my wife and colleague Iliana Martinez, without her support and help I would be able to start and finish this research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CCA	Conc
CVW	Capa
CORBA	Com
DARPA	Defe
DFD	Data
DSSA	Dom
[illegible]	Fram
[illegible]	Flex
[illegible]	First
[illegible]	Func
[illegible]	Gen
[illegible]	Gen
[illegible]	Het
[illegible]	Inte
[illegible]	Inte
[illegible]	Info
[illegible]	Inte
[illegible]	Kno
[illegible]	Kno
[illegible]	Kno
[illegible] (KAH)	Kno
[illegible]	Kno
[illegible]	Kn
[illegible]	Kn
[illegible]	Kn
[illegible]	Kn
[illegible]	Kn
[illegible]	M
[illegible]	Na
[illegible]	O
[illegible]	Pr
[illegible]	P
[illegible]	Pr
[illegible]	Pe
[illegible]	R
[illegible]	R
[illegible]	R
[illegible]	S
[illegible]	Ta

LIST OF ABBREVIATIONS

CCA	Conceptual Composite Assembly
CMM	Capability Maturity Model
CORBA	Common Object Request Broker Architecture
DARPA	Defence Advance Research Project Agency
DFD	Data Flow Diagram
DSSA	Domain Specific Software Architectureq
FB	Frame-Based
FC	Flexible Control
FOPC	First-Order Predicate Calculus
FR	Function-based Reasoning
GT	Generic Task
GT ITS	Generik Task Integrated Tool Set
HC	Hierarchical Classifier
IDL	Interface Definition Language
I-KBS	Integrated Knowledge-Based System
IPT	Information-Processing Task
ISL	Intelligent Systems' Laboratory
KADS	Knowledge Analysis and Design Support
KBS	Knowledge-Based System
KIF	Knowledge Interchange Format
KLA (KLAH)	Knowledge Level Architecture Hypothesis
KLH	Knowledge Level Hypothesis
KQML	Knowledge Query and Manipulation Language
KR	Knowledge Representation
KS	Knowledge Sourse
KSE	Knowledge Sharing Effort
MRD	Multiple Routine Designer
NSF	National Science Foundation
OMT	Object Modeling Technique
PS	Problem-Solver
PSA	Problem Solving Architecture
PSH	Problem-Solving Hierarchy
PSP	Personal Software Process
RC	Rigid Control
RD	Routine Designer
RTM	Resin Transfer Molding
SRC	Semi-Rigid Architecture
TSA	Task Specific Architecture

1.1. *Motivation*

Since the 1960s
produced many success
dissecting problem-solvin
number of general-purpos
are often used to solve a
problems require the use
hand. An alternative to th
incorporates all necessar
number of cooperating pr
a specific PSM. Often
separate knowledge-base
solving modules. In ord
problem-solving modules
Such integrated Knowle
dissertation. I define an
consists of multiple, disp
given problem.

Ontological resea
emerges as an inter-ag
communication between

1. INTRODUCTION

1.1. Motivation

Since the 1960s research in Knowledge-Based Systems (KBS) has produced many successful practical applications in various domains. Bu dissecting problem-solving patterns, several schools of thought developed a number of general-purpose problem-solving methods (PSM). These methods are often used to solve a particular task. Meanwhile, the majority of practical problems require the use of a number of different PSMs to solve the problem at hand. An alternative to the development of a monolithic, large-sized KBS which incorporates all necessary PSMs is the development of a KBS that consists of a number of cooperating problem-solving units each of which is an instantiation of a specific PSM. Often such an instantiation leads to the development of separate knowledge-based system resulting in a collection of separate problem-solving modules. In order to perform a designated function, this collection of problem-solving modules requires careful and precise assembling or integrating. Such Integrated Knowledge-Based Systems (I-KBS) are an emphasis of this dissertation. I define an Integrated Knowledge-Based System as a KBS which consists of multiple, disparate problem-solving modules co-operating to solve a given problem.

Ontological research has suggested that the use of shareable domain ontologies as an inter-agent language is a practical solution to the problem of communication between disparate modules. However, integration of a number of nowledge-based sys

ommunication media

and information flow a

derivatives of object-m

development as an ef

methods were often

nowledge-based sys

The focus of th

allows the integration

KBS.

1.2 Importanc

KBS developm

use, and integration

of a significant amo

KBS research is u

projects (such as E

Wielinga and Schre

by support from US

and NSF's[2] progra

to DARPA's High P

that

[1] Defense Advance

[2] National Science

knowledge-based systems in the I-KBS is more than the definition of a communication media and language; it requires the determination of the control and information flow as well. Software engineering practices suggest using the derivatives of object-modeling techniques (OMT) and component-based system development as an effective way to develop software systems. However, these methods were often neglected in the methodologies for development of knowledge-based systems.

The focus of this dissertation is the development of a methodology which allows the integration of the disjoint knowledge-based systems into a functional I-KBS.

1.2. Importance of the Research

KBS development methodologies, knowledge acquisition, knowledge re-use, and integration techniques for large-scale intelligent systems are the topics of a significant amount of research in the US and Europe. The importance of KBS research is underscored by the endeavors of European multi-national projects (such as ESPIRIT-I and ESPIRIT-II (van Heijst, Schreiber et al. 1997) (Wielinga and Schreiber 1994) and Vital (Motta, O'Hara et al. 1994)) as well as by support from US governmental agencies (such as DARPA[1] (DARPA 1998) and NSF's[2] program on Knowledge and Cognitive Systems). In the introduction to DARPA's High Performance Knowledge Bases (HPKB) project, it was noted that

[1] Defense Advance Research Project Agency
[2] National Science Foundation

"[The AI KBS co
from applying knowled
Techniques for creatin
Many fundamental pro
knowledge bases ha
specialized techniques
The potential payoff fr
program is tremendous

This progress in th
exploitation of systems inte
not much attention was
methodologies and develo
complex integrated knowled

1.3. Research Conte

Advances in Objec
development of the Unive
2000, that serves as a ve
complex software systems
components. Rational Rose
a "what you see is what yo
systems starting at the conc
the modeling process Ratio
object-oriented languages o

> “[The AI KBS community] reached a threshold where the power gained from applying knowledge based technology is much greater than in the past. Techniques for creating large knowledge bases have advanced significantly. Many fundamental problems of how to perform efficient inference on large knowledge bases have been solved (although others remain). Many specialized techniques for creating and applying knowledge have matured. The potential payoff from combining all this research progress into one program is tremendous. ” (DARPA 1998)

This progress in the KBS field made feasible the development and exploitation of systems intended to solve large and complex problems. However, not much attention was paid to the issue of development of practical methodologies and developmental environments that would support building complex integrated knowledge-based systems.

1.3. Research Context

Advances in Object Modeling Technique laid the basis for the development of the Universal Modeling Language (UML) (Fowler and Scott 2000) that serves as a very expressive and convenient medium for modeling complex software systems with intricate communication between its different components. Rational Rose® – a commercial tool based on the UML – provides a “what you see is what you get” environment for modeling complex software systems starting at the conceptual level of requirements definition. As a result of the modeling process Rational Rose® generates a skeletal code in one of the object-oriented languages of choice: C++, Java, Visual Basic, etc. This skeletal

ode provides guideline
terface of each object
communications. As u
terface, the software
structures without worry
and retrieving them on d
components. At the same
development lacks any s
high performance knowled
and a proof-of-the-concept

The Generic Task
templates for constructing
sense that its templates
accommodate a deviation
that governs the inference
defies the principle of h
templates for the problem-
identify parts of the over
the templates. This pos
resulting problem-solvers
problem-solving organizat

The alternative is to
Section 3.2.5) and decom

code provides guidelines for the designer team by defining the input-output interface of each object, object specifications, and the channels for inter-object communications. As long they do not break through the defined in UML interface, the software engineer can implement the objects and define data structures without worrying about integration issues. By storing object models and retrieving them on demand, UML not only allows but encourages reusing components. At the same time, however, the area of knowledge-based systems' development lacks any similar methodology applicable to designing large scale, high performance knowledge-based systems. The need in such a methodology and a proof-of-the-concept tool motivated this dissertation research.

The Generic Task (GT) approach (Section 3.2.3) provides large-grain templates for constructing knowledge-based systems. However, GT is brittle in a sense that its templates are rigid and pre-defined. Therefore, in order to accommodate a deviation from a template it is necessary to re-program the code that governs the inference and/or knowledge representation in the template. This defies the principle of having a certain number of reusable, unchangeable templates for the problem-solving needs. On the other hand, it is often possible to identify parts of the overall problem that could be effortlessly tackled by one of the templates. This poses the problem of how to effectively integrate the resulting problem-solvers so they not only function properly, but the overall problem-solving organization is understandable and manageable.

The alternative is to design KBSs following the CommonKADS approach (Section 3.2.5) and decompose the problem into a large set of primitives and then integrate them in

programmed separately

also relies on the deve

problem-solving modules

solving architecture conc

This dissertation d

unnecessary programmin

chunks, each of which ca

solving template. By a

problem-solver as at a b

and using the Knowledge

22), it is feasible to co

the integrated system.

specification of commu

specifying their activatio

of the integrated syste

problem-solver, its con

addition, the dissertatio

an KBS: first, to deco

handled by a single pre

and control flow betwee

practical application of a

possible in the light of

then integrate them in a knowledge-based system. Each primitive module is programmed separately and, on practice, can rarely be reused. In a nutshell, it also relies on the developer's programming skills to implement and integrate problem-solving modules. This approach results in the system with the problem-solving architecture concealed inside the code.

This dissertation describes a methodology that eliminates excessive and unnecessary programming for problems that could be decomposed into smaller chunks, each of which can be solved by application of the large-grain problem-solving template. By assuming the *black box* stance i.e., looking at every problem-solver as at a black box with known functionality, inputs, and outputs, and using the Knowledge-Level Architecture Hypothesis (Sticklen 1989) (Section 3.2.2), it is feasible to compose and describe the problem-solving architecture of the integrated system. The composition of the system includes explicit specification of communication channels between co-operating entities and specifying their activation condition, whereas the description of the functionality of the integrated system should be built on functionality of each participating problem-solver, its connections, and its supervising control structures. In addition, the dissertation methodology provides implicit guidelines for designing an I-KBS: first, to decompose the problem into the chunks that could be easily handled by a single pre-defined template and second, define the communication and control flow between such modules. It is important to understand that the practical application of a knowledge-level architecture to design I-KBSs became possible in the light of recent advances in software engineering (Section 3.4), communication technic

programming (section 3

The three main is

1. The Knowledge-Le

main idea of KLA

decomposed into sub

be understood at the

sub-agents and spe

structure. The KLA m

KBS behavior and fin

2. The application of

software systems

3. Advances in ontolo

communication med

the conceptualizatio

relations in the dom

threefold (Gruber 19

and to clarify possib

database backbone

interest. Third, the c

That is, agents that

using this ontology a

communication techniques and approaches (Section 3.4), and component-based programming (section 3.4.2).

The three mainissues that prompted this dissertation are listed below:

1. The Knowledge-Level Architecture hypothesis (KLA) (Section 3.2.2). The main idea of KLA is that if an agent [a knowledge based system] is decomposed into sub-agents [problem-solvers], then the composite agent can be understood at the knowledge level by a knowledge level description of sub-agents and specifying the sub-agents' integration and communication structure. The KLA model of an integrated KBS is then used for validating the KBS behavior and finding integration conflicts in the I-KBS architecture.
2. The application of Function-Based Reasoning (FR) theory to modeling software systems
3. Advances in ontology research suggest using a domain ontology as a communication media for inter-agent communication. A domain ontology is the conceptualization of facts, terms, and assumptions and their inter-relations in the domain of interest. The purpose of domain ontologies is threefold (Gruber 1992). First, it helps to organize the domain terminology and to clarify possible misreadings of a term. Second, it helps in creating a database backbone for the multitude of problem-solvers in the domain of interest. Third, the ontology serves as a basis for inter-agent communication. That is, agents that participate in the problem-solving process communicate using this ontology and interpret it in the same way.

I will show that
designing and modeling
control structures and
architecture (KLA). It is
are commonly used in th
1. Rigid Control archite
solvers (PS) and exe
time.

2. Semi-Rigid Control
determined during the
operation is decided o
3. Flexible Control arch
execution order are s
operation.

Modern methodol
systems often concentra
architectures. The metho
KBS developer to use a
closely fit the control f

1.4. Research Obj

The development
knowledge acquisition, a
proposed in this disserta

I will show that with some augmentations, the FR methodology allows designing and modeling integrated knowledge-based systems utilizing arbitrary control structures and with an emphasis on the system's knowledge level architecture (KLA). It is possible to recognize three major control structures that are commonly used in the development of integrated systems:

1. Rigid Control architecture (RC), i.e. all interconnections between problem-solvers (PS) and execution order are determined during system development time.
2. Semi-Rigid Control architecture (SRC), i.e. PS's interconnections are determined during the system development time. However, the exact order of operation is decided on during run time.
3. Flexible Control architecture (FC), i.e. neither PS' interconnections nor execution order are set *a priori* but emerge dynamically during the system's operation.

Modern methodologies for building integrated systems and multi-agent systems often concentrate on the implementation of one of the above control architectures. The methodology proposed in this dissertation aims to enable the I-KBS developer to use any combination of the control architectures listed above to closely fit the control flow of the problem at hand.

1.4. Research Objective

The development of an I-KBS poses problems of decomposition, of knowledge acquisition, and of the integration of pre-built modules. The approach proposed in this dissertation aims to resolve some of these problems by:

1. Providing the mean
sub-problems and

2. To organize the int
modules called to so

The research is
scale integrated know e
system generally include

- To decompose a prob
handle from the know
- To identify the metho
possible to locate pre
- To develop a doma
interrelations – to p
modules as well
communications.
- To implement individ
- To integrate individ
control information.
- To develop a front-e
to an end-user.

In spite of the f
addressed in often rese
with an explicit problem-

1. Providing the means to decompose the problem into the set of manageable sub-problems and
2. To organize the integration infrastructure based on the functionality of the modules called to solve each of these sub-problems.

The research is focused on the problem of developing medium-to-large scale integrated knowledge-based systems. The process of building such a system generally includes the following consecutive steps:

- To decompose a problem into a set of smaller sub-problems that are easier to handle from the knowledge-engineering perspective.
- To identify the methods that are capable of solving these sub-problems, and if possible to locate pre-built modules that serve these purposes.
- To develop a domain ontology – a vocabulary of domain terms and their interrelations – to provide a backbone for the individual problem-solving modules as well as to arrange means for inter-problem-solver communications.
- To implement individual modules.
- To integrate individual modules in the target I-KBS leveraging available control information.
- To develop a front-end to the I-KBS that presents the results of the I-KBS run to an end-user.

In spite of the fact that many of the above listed tasks have been addressed in often research, the issue of the development of functional I-KBS with an explicit problem-solving architecture, clear but flexible control structures, and adaptable access

methodology discussed

There are several

approaching these prob

- To correctly identify
- To correctly organize
- To unambiguously d

The traditional Fu

about a device in terms

functionality of a device

the branches of FR, wh

apply this metaphor to a

be necessary to augme

allow modeling and inte

information and control f

The core of my

problem-solver that is lo

functionality. Given the

could be solved by su

higher-level problem-so

these black boxes. The

boxes to form higher le

problem-solver is a co

and adaptable access to the information flow is an open question. The methodology discussed in this dissertation answers some of these questions.

There are several difficulties that have to be taken into consideration when approaching these problems:

- To correctly identify unitary problem-solving components.
- To correctly organize functional decomposition-composition of the I-KBS.
- To unambiguously direct the information and control flow through the I-KBS

The traditional Function-Based Reasoning (FR) gives the means to reason about a device in terms of the functions of its sub-devices and to compose the functionality of a device from the functions of its components. Following one of the branches of FR, which uses the metaphor of device on software system, we apply this metaphor to an Integrated Knowledge-Based System. To do so, it will be necessary to augment the traditional Function-Based Reasoning theory to allow modeling and integrating I-KBSs as well as the explicit assignment of information and control flow between the cooperating entities.

The core of my approach is the notion of function or role of a unitary problem-solver that is looked at as a *black box* with known inputs, outputs, and functionality. Given the decomposition of the problem at hand into chunks that could be solved by such unitary problem-solvers, the goal is to compose the higher-level problem-solver (capable of solving more complex problem) out of these *black boxes*. These higher-level problem-solvers in turn are used as *black boxes* to form higher level modules, and so on. The functionality of a larger problem-solver is a composition of the functionalities of lower-level problem-solvers. The result o

procedure is a knowle

solves the problem.

1.5. Deliverables

The outcome o

knowledge-based syste

dissertation. To prove

design a knowledge-bas

On the KBS side o

KBS-1. Augmentation

functional decompos

limited set of unitary

KBS-2. Addition of an

methodology which e

flow through the Fun

KBS-3. Addition of the

of the Functional M

arbitrary distributed

KBS.

KBS-4. The develop

described framewo

solvers. The result of the goal-driven recursive repetition of the composition procedure is a knowledge-based system that performs the desired function, i.e. solves the problem.

1.5. Deliverables

The outcome of this work is a shell that allows building integrated knowledge-based systems using the theoretical framework discussed in this dissertation. To prove the validity of the methodology the shell is used to re-design a knowledge-based planning system.

On the KBS side of the research the contributions are:

KBS-1. Augmentation of the Function-Based Reasoning methodology to allows functional decomposition of integrated knowledge-based systems using a limited set of unitary problem-solvers as base-line building blocks.

KBS-2. Addition of an Information-Processing layer to the functional modeling methodology which enables an explicit assignment of information and control flow through the Functional Model.

KBS-3. Addition of the capability of describing dynamic control over the parts of the Functional Model of the I-KBS, therefore enabling implementation of arbitrary distributed control structures and problem-solving units within the I-KBS.

KBS-4. The development of a shell which supports building I-KBSs in the described framework.

The domain o

composite materials[3]

planning system – So

the dissertation. The r

of the system, since S

modified only by an

features:

CMD-1. Possibility

architecture of its

the developed s

examine every p

Modification can

also the control f

CMD-2. Explicit d

components of t

strategy of Socr

for teaching ma

An importa

architecture hypot

allows developing

trivial tasks in a w

[3] Composite Manu

[4] Socharis is the C

and construction

The domain of choice is the domain of manufacturing with polymer composite materials[3], where I target the existing intelligent manufacturing planning system – Socharis[4] – and re-design it using the approach proposed in the dissertation. The main purpose of this re-design is to ease the maintenance of the system, since Socharis's control is directly coded in Smalltalk and can be modified only by an experienced software engineer. The redesigned system features:

CMD-1. Possibility of on-the-fly change in the system's architecture, the architecture of its components, and the knowledge content. Re-designed in the developed shell the new Socharis will provide graphical interfaces to examine every part of the system and the ability modify them if necessary. Modification can not only affect the knowledge encoded in the system, but also the control flow that governs the order of the problem-solvers' activation.

CMD-2. Explicit definition of the information and control flow between the components of the system that lets the user understand the problem-solving strategy of Socharis. This, in turn, may be used as an educational instrument for teaching manufacturing in polymer composites.

An important conjecture that follows from the leveraging Knowledge-level architecture hypothesis is that the methodology discussed in this dissertation allows developing reusable integrated problem-solving architectures for non-trivial tasks in a way that the second-generation expert systems (Section 3.2.2)

[3] Composite Manufacturing Domain - CMD

[4] Socharis is the Greek name of ancient Egyptian god Sekar – the god of crafts and construction

prompted the develop

tasks.

1.6. *Dissertation*

The rest of the

the domain problem tha

historical and theoret

analyses the underpinn

statement of the prob

implementation framew

framework to build I-KBS

the described domain p

architecture are addres

future research venues a

prompted the development of reusable problem-solving methods for common tasks.

1.6. Dissertation Organization

The rest of the dissertation is organized as follows: Chapter 2 introduces the domain problem that serves as a proving ground for the advocated approach. Historical and theoretical background is discussed (Chapter 3). Chapter 4 analyses the underpinning theory of the research. Chapter 5 gives a detailed statement of the problem, followed by a description of the approach and implementation framework. An illustrative example of using the developed framework to build I-KBSs is given in Chapter 6. Re-implementation of I-KBS for the described domain problem and discussion of the resulting problem-solving architecture are addressed in Chapter 7. In conclusion (Chapter 8) I address future research venues and general challenges in KBS research.

Any research in
problem (e.g. to diag
malfunction) and the A
new or augmented A
particular problem so
methods, etc. to solve
researcher does not ha
approach and, therefor
theoretic philosophizing.

This dissertation
materials and demonstr
integrated KBS can be
Lukibanov et al. 1998; M

To create a syst
mechanical assemblies
and output specification

INPUT: The inp
composite assembly (C
A part of the CCA ca
hierarchically, depicting
relationships between

2. DOMAIN PROBLEM

Any research in KBS is usually composed of two problems: the domain problem (e.g. to diagnose a disease, to aid in design, or to find the cause of malfunction) and the AI problem. The latter is concerned with development of new or augmented AI theory. The former is concerned with the applying particular problem solving strategies, techniques, knowledge representation methods, etc. to solve a domain problem. Without a domain problem a KBS researcher does not have any means to support the validity of the advocated approach and, therefore, the described AI method is bound to be merely theoretic philosophizing.

This dissertation introduces a problem in the domain of composite materials and demonstrates how the proposed approach to the development of integrated KBS can be used to solve it. The example domain problem (Martinez, Lukibanov et al. 1998; Martinez, Lukibanov et al. 1999) is described as follows:

To create a system for generating conceptual manufacturing plans for mechanical assemblies made of composite materials which satisfies the following input output specifications.

INPUT: The input to the system is given in terms of a *conceptual composite assembly* (CCA) (Lenz, Hawley et al. 1998; Zhou, Lenz et al. 1999). A part of the CCA called the *configuration model* represents an assembly hierarchically, depicting not only assembly-component relationships but also the relationships between geometric parameters and design features of the assembly. Within suc

nclude structure obje

the base level of a

fastening one struc

expressing such feat

relationships between

component or assem

between objects and t

feature or subassembly-

of its type, rough geo

information about joining

OUTPUT: The c

hierarchical manufactu

generated manufacturi

the final product and its

machinery, etc. The pla

The knowledge-

created in the Intellig

Socharis was created

However, this tool se

developers to code a

closed source" archite

assembly. Within such a hierarchical configuration model, ontological members include structure objects (an assembly or a subassembly), component objects (the base level of atomic parts for the structural assembly), joining objects (fastening one structure or component to another), and feature objects (expressing such features as holes). An ontology of link types expresses the relationships between objects. Link types include part-whole links (assembly-component or assembly-internal joint), join links (expressing connectivity between objects and the joints between them), and feature links (component-feature or subassembly-feature). Each component node contains the description of its type, rough geometry and material class. Each joining node contains information about joining parameters.

OUTPUT: The output of the system should be a family of conceptual hierarchical manufacturing plans for the manufacturing of the assembly. The generated manufacturing plans can be used for estimating both the properties of the final product and its required manufacturing resources, such as labor, tooling, machinery, etc. The plan is represented as a directed graph.

The knowledge-based system that performs this function – Socharis - was created in the Intelligent Systems Laboratory at Michigan State University. Socharis was created using GT ITS (the Generic Task Integrated Tool Set). However, this tool set does not support complex control flow, which led the developers to code all the control information in VisualWorks Smalltalk. Such "closed source" architecture hinders the maintainability of Socharis especially in

an environment whe
changed and/or tuned

The next sectio
knowledge organizatio
provides an introductio
materials.

2.1. Introduction

A composite ma
materials (reinforcing
composition on a mac
maximizes specific per
materials or to the agg
been termed the "mat
flexibility is often acco
designers of composite
defined sub-areas beca
domain. Consequently
typically realized.

Design for manu
composites than meta
geometry, functional re
constrains or suggests
selection of a manufa

an environment where the knowledge base and the control flow should be changed and/or tuned for each particular deployment of Socharis.

The next section discusses the architecture of Socharis and describes its knowledge organization, content, and knowledge-level architecture as well as provides an introduction to the domain of manufacturing with polymer composite materials.

2.1. Introduction to Composites

A composite material is a heterogeneous combination of two or more materials (reinforcing elements, fillers and binders), differing in form or composition on a macro-scale. The combination results in a material that maximizes specific performance properties traceable to one of the constituent materials or to the aggregate composite material. Composite materials have been termed the "materials of the future". However, this increased design flexibility is often accompanied by an increased design complexity. Industrial designers of composite material systems and architectures work within narrowly defined sub-areas because of the vast expertise necessary to cover the entire domain. Consequently, the full design flexibility of composite materials is not typically realized.

Design for manufacture has been historically more prevalent for polymer composites than metals. In part, this is because many design factors (e.g. , geometry, functional requirements, production rates, and material system) either constrains or suggests specific composite fabrication technologies. As the selection of a manufacturing process can greatly affect both the functional qualities and cost of

purely design concerns

of available composite

level comparison of fa

bases can simply elim

among the list of availab

Over the past se

Intelligent Systems La

integrated design and fa

et al. 1998; Martinez,

supports the conceptua

structural assemblies

engineers in exploring

evaluating evolving solu

The suite produc

original metal part. E

meeting the original de

manufacturing planne

manufacturing plans

designs, the designer c

of the more effective p

and aesthetic requireme

qualities and cost of the final product, manufacturing concerns often dominate purely design concerns. Even using solely manufacturing constraints, the variety of available composites fabrication technologies can complicate any conceptual-level comparison of fabrication alternatives for a part. Additionally, designers' biases can simply eliminate possible alternate fabrication processes, artificially limiting the list of available options.

Over the past several years, a software suite has been developed in the Intelligent Systems Laboratory at Michigan State University to accomplish integrated design and fabrication planning at the conceptual level (Lenz, Hawley et al. 1998; Martinez, Lukibanov et al. 1999; Zhou, Lenz et al. 1999)). It supports the conceptual design/re-design process for the transformation of metal structural assemblies to polymer composites. This software suite assists engineers in exploring the space of design and manufacturing possibilities and evaluating evolving solutions without detailed design or analysis.

The suite produces a family of conceptual composite redesigns from the original metal part. Each member of this family is a valid redesign option, meeting the original design requirements. These redesigns are passed to a manufacturing planner for further evaluation. By reviewing alternative manufacturing plans suggested for the functionally equivalent conceptual designs, the designer can rule out those that are less effective and concentrate on the more effective plans. The resulting solutions will not only satisfy functional and aesthetic requirements, but can be easily manufactured.

The following

2000), the manufactu

mentioned above. It g

parts from a conceptu

manufacturing planning

upon which Socharis wa

2.2 High-Level P

Figure 1 reflects t

mirrors the domain expe

for polymer composite

1993). First, it is neces

product designer descrip

terminology. This proble

solvers that filter and co

problem-solving stage ta

several manufacturing p

assembly constraints th

the part-subpart relatio

connected with part B u

The next sub-prob

to rule-out unsatisfactor

enters the further stage

The following discussion presents Socharis (Lukibanov and Martinez 2000), the manufacturing fabrication planning portion of the software suite mentioned above. It generates a family of applicable conceptual manufacturing plans from a conceptual description of a composite assembly. The conceptual manufacturing planning strategy presented previously served as the framework upon which Socharis was built.

2.2. High-Level Problem-Solving Strategy of Socharis

Figure 1 reflects the high-level problem solving strategy of Socharis, which mirrors the domain expert view of choosing an appropriate manufacturing plan for polymer composite assembly (Committee 1987; Bickerton, Stadtfeld et al. 1998). First, it is necessary to convert the description of the problem from the product designer description (Zhou, Lenz et al. 1999) to manufacturing engineer terminology. This problem is taken care of by the application of several problem-solvers that filter and convert the data. Along with translating the input data, this problem-solving stage takes care of partial assembly ordering – by developing a skeletal manufacturing plan. This plan is a graphical representation of temporal assembly constraints that indicate assembly operation precedences based on the part-subpart relations in the structure description (i.e. part A can not be connected with part B until subparts of A: A1 and A2 are bonded to part A).

The next sub-problem is to process the converted description of every part to rule-out unsatisfactory technologies. The list of applicable technologies then enters the further stage of the problem-solving process which is the basis on each part's requir

used with the targe

The later st

results (a family o

between alternat

that rate the relat

solving process of

The problem

cooperating probl

Martinez, Luk ban

cooperation. Th

knowledge about p

The doma

Class→ Attributes

succinct descripti

definition for Sha

medium, high},....}

nonlinear struct

values of AspectR

term definition. S

adds meaning to t

each part's requirements generates sets of manufacturing parameters, which if used with the target technology will produced the desired part.

The later stages of the problem-solving process tend to produce multiple results (a family of applicable manufacturing plans). To help the user to choose between alternatives the manufacturing plans are ranked by a set of estimators that rate the relative merits of each process. This stage concludes the problem-solving process of Socharis.

The problem-solving architecture of Socharis consists of a large number of cooperating problem-solving agents. A composite materials *domain ontology* (Martinez, Lukibanov et al. 1999; Zhou, Lenz et al. 1999) is used to facilitate this cooperation. This domain ontology provides a vocabulary for representing knowledge about polymer composite materials.

The domain ontology is a four layer deep hierarchy: Class Category → Class→ Attributes → Parameters. Each concept of the ontology also contains a succinct description of any assumptions made. Take, for example, a term definition for Shape → Shell → {AspectRatio, WallThickness, ...} → {{low, medium, high},...}. An explicit definition of a Shell (e.g. , thin-walled planar or curvalinear structure) and its quantitative correspondence to the qualitative values of AspectRatio attribute minimize any potential ambiguity inherent in the term definition. Such comments are mandatory parts of the representation and adds meaning to the syntax of the term.

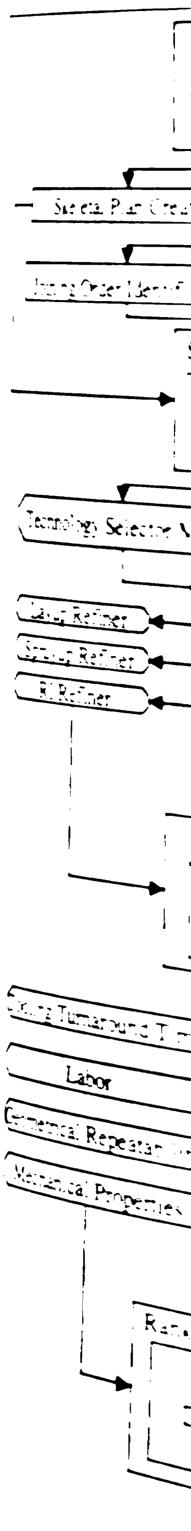


Figure 1. Problem-so

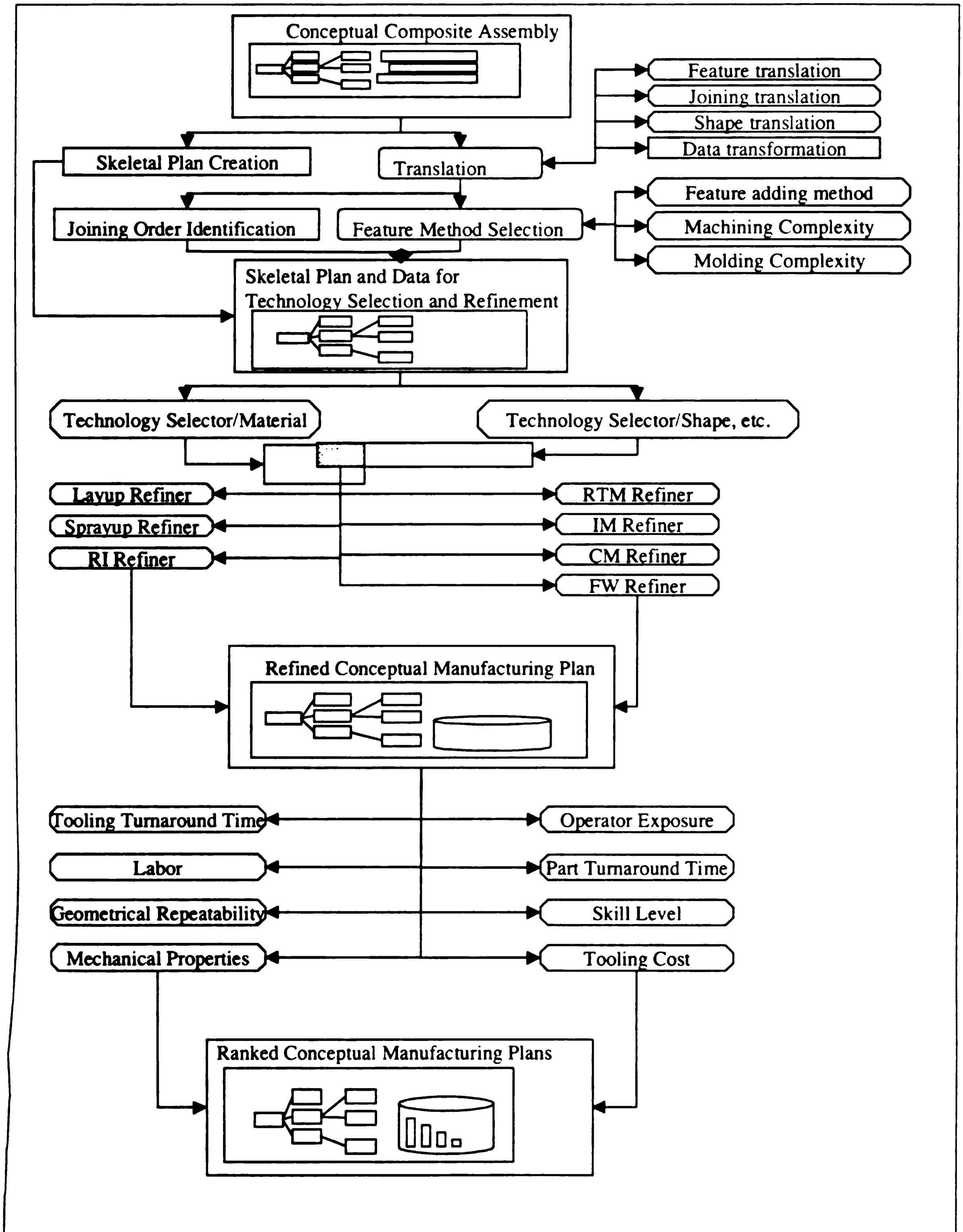


Figure 1. Problem-solving Strategy of Socharis.

The purpose of

domain terminology a

steps in creating a dat

domain of interest.

communication. That

communicate using th s

2.3. *Problem-Sol*

2.3.1. Translation

The task of this

prepare design para

selectors/refiners downs

of the skeletal manufac

addition methods.

Skeletal Plan Cr

requires the analysis

artifact. The generated

configuration model. A

structure, and any requ

manufacturing plan.

These additional

Data Translation. A

mechanical structure is

The purpose of these ontologies is threefold. First, it helps to organize the domain terminology and to clarify possible misreadings of a term. Second, it helps in creating a database backbone for the multitude of problem-solvers in the domain of interest. Third, the ontology serves as a basis for inter-agent communication. That is, agents that participate in the problem-solving process communicate using this ontology and interpret it in the same way.

2.3. Problem-Solving Architecture of Socharis

2.3.1. Translation and Skeletal Plan Generation

The task of this stage is to create a skeletal manufacturing plan and to prepare design parameters that will be processed by technology selectors/refiners downstream. This task is divided into three sub-tasks: creation of the skeletal manufacturing plan, data translation, and selection of the feature addition methods.

Skeletal Plan Creation. Construction of a skeletal manufacturing plan requires the analysis of the existing conceptual configuration model of the artifact. The generated skeletal plan is topologically analogous to the design's configuration model. At this point, the skeletal plan has only a rudimentary structure, and any required add-on features have not yet been included in the manufacturing plan.

These additional details are extracted from the configuration model in a **Data Translation.** At this stage, the descriptive information about the mechanical structure is translated from the terminology used by designers into that used by manufa

nology to the manuf

Feature Additi

a feature to a comp

machining, the feature

fabricated, whereas in

produced. The metho

based on that feature's

component.

2.3.2. Generati

After a skeletal p

fabrication technology t

each applicable fabrica

specific to each selecte

Technology Sel

stage as the approp

independent problem-s

geometrical features of

and (2) material feature

problem solvers are lis

produce a given compo

the list of the technolog

that used by manufacturers. The translation involves a mapping of the design ontology to the manufacturing ontology.

Feature Addition Method Selection. There are two methods for adding a feature to a component of a composite part: machining or molding. In machining, the feature is added to the component after the component has been fabricated, whereas in molding the feature is incorporated as the component is produced. The method for creating each individual feature must be decided based on that feature's tolerance allowances and the production quantities of the component.

2.3.2. Generating the Family of Technological Alternatives

After a skeletal plan has been generated, Socharis assigns one or more fabrication technology to each component within the manufacturing plan. After each applicable fabrication technology is selected, manufacturing parameters specific to each selected technology are set.

Technology Selection. The first use of the translated data occurs in this stage as the appropriate manufacturing technologies are selected. Two independent problem-solvers select applicable technologies based on (1) geometrical features of the component (shape, aspect ratio, wall thickness, etc.) and (2) material features (type of resin, fiber architecture). The results of both problem solvers are lists of manufacturing technologies that theoretically could produce a given component. The intersection of these lists, therefore, represents the list of the technologies that satisfy all criteria given in the input data. The technologies conta

for further process

Technolog

parameters for ea

requirements (e.

requirements (e.g

subsets of para

Component data

parameters (e.g.

define these par

different sets of

produced with m

generated manuf

2.3.3. Eva

Running S

the assembly. (

exponential incre

navigation of th s

as problem-solv

riments.

Merit tab

different design s

manufacturing m

technologies contained in this intersection are then passed to the refinement step for further processing.

Technology Refinement. After the technologies are selected, the parameters for each technology are defined. These parameters include curing requirements (e.g. , time, pressure, curing type, post curing), tooling requirements (e.g. , tooling complexity, tooling material), and so on. The specific subsets of parameters vary among the different generic technologies. Component data (e.g. , geometry, material, and add-on features) and global parameters (e.g. , production, and global tolerance allowances) are used to define these parameter values. Each generic technology may contain many different sets of parameter values. Additionally, each component may be produced with multiple technologies. Therefore, some way to compare the generated manufacturing options is required.

2.3.3. Evaluation

Running Socharis results in multiple manufacturing plans for every part in the assembly. Given the semi-deterministic assembly order, this leads to an exponential increase in the number of possible manufacturing plans. To enable navigation of this expansive space of possible manufacturing plans, Socharis's last problem-solving stage ranks the technologies according to a predefined set of merits.

Merit tables are traditionally used in engineering practice for ranking different design solutions. Every row in a merit table is associated with a critical manufacturing metric (e.g. , cycle time, tooling turnaround time, and operator skill

level). Each metric
an engineer. Altern
calculating the weight

The specific
evaluation of time (c
mechanical propertie
factors (operator skill
qualitative value from
then ranked according

The user can li
Socharis only displa
manufactured. This
significantly reduces t
user must choose.

2.4. Accomplis
Domain Prob

One of the m
conceptual design a
integration of knowled
system. This knowle
goals of Socharis:
technologies for a co
source could be used
assemblies.

level). Each metric is linked to a weighting factor that reflects its importance to an engineer. Alternative technologies are ranked according to these merits by calculating the weighted sum of the estimated metrics.

The specific metrics used in Socharis were selected to enable the evaluation of time (cycle time, tooling turnaround time), quality of the product (mechanical properties of the product, geometrical repeatability) and human factors (operator skill level, operator exposure, labor). This estimation assigns a qualitative value from one to ten for each metric. The alternative processes are then ranked according to the value of the weighted sum of the estimated metrics.

The user can limit the design space under examination by requesting that Socharis only display the best few options for each component to be manufactured. This is possible because of the merit table evaluations. This significantly reduces the number of refined fabrication options among which the user must choose.

2.4. Accomplishments and Shortcomings of Socharis and Application Domain Problem Statement

One of the most important contributions of Socharis to the area of conceptual design and manufacturing with composite materials was the integration of knowledge about eight generic manufacturing technologies in one system. This knowledge was specifically structured to suit the problem-solving goals of Socharis: to identify, instantiate, and evaluate manufacturing technologies for a composite assembly of interest. This concentrated knowledge source could be used to plan and estimate the manufacturability of the composite assemblies.

As an importa
intelligent Systems
manufacturing with c
ontology is a four-la
terminology used to
auxiliary information.
development of GT-ba

Establishing and
to the generalization f
solving strategy of c
suitable for developin
composite materials.

Noted, that a
Manufacturing ontolog
solvers through the
features a graphical
structure of problem-
and augment the kn
scholars. The same
In fact, this constitut
change mid- and high

The reason f
proceeded which was

As an important byproduct of the Socharis project, the research group at Intelligent Systems Laboratory developed a comprehensive ontology of manufacturing with composite materials (Lukibanov and Martinez 2000). This ontology is a four-layer deep class, sub-class hierarchy that accounts for the terminology used to describe both, the manufacturing processes and any auxiliary information. This ontology was used as a backbone for the development of GT-based problem-solvers that cooperate in Socharis.

Establishing and documenting Socharis's problem-solving architecture led to the generalization that these architectures representing a generic problem-solving strategy of conceptual manufacturing planning. This framework is suitable for developing planning systems in the domains other then polymer composite materials.

Noted, that advanced users of Socharis have easy access to its Manufacturing ontology and the knowledge content of the constituent problem-solvers through the Generic Tasks Integrated Tool Set (GT ITS). GT ITS features a graphical user interface for modifying knowledge and the internal structure of problem-solvers. As a result it does not take much effort to update and augment the knowledge about the processes that are already part of the Socharis. The same is not true about the overall problem-solving architecture. In fact, this constitutes the main drawback of Socharis - inability to effortlessly change mid- and high-level problem-solving architecture.

The reason for this is that the complexity of problem-solving by far exceeded which was presently supported in the GT ITS. The team of designers had to escape to th

Smalltalk programm

impossible for an

architecture. Even se

for evaluating add

knowledge of Smallta

code. The applicat

overcome these shortc

The task of e

comprises the applica

described in this dis

solving architecture o

shell for building in

presented methodolog

on the fly, adding and

the information flow.

knowledge-based co

research focus repor

had to escape to the coding of the global control architecture in VisualWorks ® Smalltalk programming language. This led to a situation where it is almost impossible for an outsider to make any changes in the problem solving architecture. Even seemingly simple tasks such as adding more problem-solvers for evaluating additional manufacturing technology requires not only the knowledge of Smalltalk, but also familiarity with existing GT ITS architecture and code. The application domain problem of this research, consequently, is to overcome these shortcomings of the present Socharis Implementation.

The task of exposing the problem-solving architecture of Socharis comprises the application domain goal of this research. The application of the described in this dissertation methodology allows representing the problem-solving architecture of Socharis in a graphical, easy to understand way. The shell for building integrated knowledge-based systems which supports the presented methodology allows changing Socharis's problem-solving architecture on the fly, adding and deleting components, reassigning control, and redirecting the information flow. It is necessary to note that the redesign will affect only knowledge-based core of Socharis ; other modules of Socharis lie outside the research focus reported in this dissertation.

3.1. Overview

The discussion

development precede

new line of compa

strategies) and exp

1989). The main idea

decomposable into su

knowledge level by a

the sub-agents' integr

greater detail in Sectio

3.1.1. Control

All currently e

related to one of the

as described below:

1. Rigid Control arc

solvers (PS) and

development time.

2. Semi-Rigid Cont

determined during

operation is decide

3. PREVIOUS RESEARCH

3.1. Overview of Knowledge-Based Systems

The discussion about control architectures commonly used in the KBS development precedes the overview of knowledge-based systems to provide a new line of comparison between different approaches: prevalent control strategy(ies) and explicitness of their Knowledge Level Architectures (Sticklen 1989). The main idea of Knowledge Architecture Hypothesis is that if an agent is decomposable into sub-agents, then the larger agent can be understood at the knowledge level by a knowledge level description of sub-agents and specifying the sub-agents' integration and communication structure. KLAH discussed in a greater detail in Section 3.2.2.

3.1.1. Control Architectures

All currently employed KBS development integration schemes could be related to one of the control architectures or combinations of these architectures as described below:

1. Rigid Control architecture (RC), i.e. all interconnections between problem-solvers (PS) and their execution sequence are determined during system development time.
2. Semi-Rigid Control architecture (SRC), i.e. PS's interconnections are determined during the system development time. However, the exact order of operation is decided on during run time.

3. Flexible Control

execution order a

operation.

These control

respective knowledge

RC has an ex

information of the syst

KLA of any connecte

participating problem

Nevertheless, the app

relatively complex prob

since complex problem

covered by Rigid Contr

unpredictable results for

Similar to the R

could be expressed in

any interconnected s

cooperating problem-s

subplan will be active

process. However, th

that it does not prov

emergent behavior is

3. Flexible Control architecture (FC), i.e. neither PS' interconnections nor execution order are set *a priori* but emerge dynamically during the system's operation.

These control strategies differ significantly in the expressiveness of their respective knowledge level architecture (KLA).

RC has an explicit KLA that is understood through input and output information of the system. Moreover, one can always perform an analysis of the KLA of any connected subpart of the system by analyzing the roles of each participating problem solver, information flow, and execution order. Nevertheless, the application of Rigid Control to the development of KBS for relatively complex problem rarely results in the most elegant or optimal solution, since complex problems seldom follow a simple problem-solving strategy that is covered by Rigid Control. In addition, this approach suffers from brittleness (i.e. unpredictable results for unexpected inputs) and often.

Similar to the Rigid Control architecture, KLA of the Semi-Rigid Control could be expressed in terms of its input and conditional output. That is, given any interconnected subpart of the system and purposes/roles of each cooperating problem-solver one can determine the conditions in which this subpart will be active and the role of this subpart in the overall problem-solving process. However, the main drawback of the Semi-Rigid Control approach is that it does not provide enough run time flexibility, especially in case where emergent behavior is important. Another shortcoming is that this approach usually requires har

complicates mainten

KLA of Flexib

evel description of e

purpose of the whole

combination of me

dependencies are not

provides flexible, dy

Nevertheless, this app

- the termination pro

  and

- difficulty of know

  additional bookkee

This dissertatio

modeling KBSs that

combination. The

Chandrasekaran 198

KBS, however there

solvers.

Section 3.1.2

development method

level architecture as

strategies. I also co

usually requires hand coding of conditional activation of problem-solvers which complicates maintenance and debugging of KBS.

KLA of Flexible Control is difficult to determine. Given the knowledge-level description of each of the problem-solvers participating in KBS and the purpose of the whole system, one can hardly understand the role of an arbitrary combination of methods since their interconnections and input-output dependencies are not explicit and emerge only during run time. Flexible Control provides flexible, dynamic control over global problem solving process. Nevertheless, this approach also suffers from several shortcomings:

- the termination problem (i.e. how do we know that result has been achieved) and
- difficulty of knowledge debugging (the need to debug KBSs demands additional bookkeeping of activation records and problem-solving context).

This dissertation describes the framework that allows developing and modeling KBSs that is able to use any of these control architectures or their combination. The implementation supports use of Generic Task (GT) (Chandrasekaran 1983) problem-solvers as bottom level building blocks for an I-KBS, however there is no theoretical limitation for types of individual problem-solvers.

Section 3.1.2 provides an account of knowledge-based systems' development methodologies with respect to the openness of their knowledge-level architecture as well as to their ability to handle different kinds of control strategies. I also compare these methodologies along several directions, such as explicitness of

development shell.

3.1.2. Rule-B

Despite the se

Shortliffe 1984), th

overwhelming majorit

it. The main advanta

to do is to encode IF

available rule-based

Production System (P

It is easy to se

each rule is a separa

Major drawba

- Absence of expl

conflict resolution

- Lack of evident

rule in the KBS is

case of multiple f

structural analys

systems.

3.1.3. Hardw

Hardwired sy

KBSs. Following th

as explicitness of the knowledge-level architecture and the availability of the development shell.

3.1.2. Rule-Based Systems

Despite the serious critique of the rule-based methodology (Buchanan and Shortliffe 1984), this approach is extensively used in practice. In fact, the overwhelming majority of the commercially available KBSs and KBS shells follow it. The main advantage of this method is its simplicity. All the developer needs to do is to encode IF-THEN rules using one of a large number of commercially available rule-based system shells (e.g. CLIPS: C Language Integrated Production System (Riley 1998) or Jess (Friedman-Hill 1997))

It is easy to see that a conventional rule-based system uses the FC where each rule is a separate problem solver.

Major drawbacks of this approach (save for the usual critique) are:

- Absence of explicit control, which is usually programmatically encoded in the conflict resolution strategy.
- Lack of evident knowledge-level architecture. That is, the purpose of each rule in the KBS is unclear. Moreover, how rules are activated and selected in case of multiple firing is hidden in the conflict resolution module. This hinders structural analysis, troubleshooting, and effortless re-use of rule-based systems.

3.1.3. Hardwired Systems

Hardwired systems represent a sizable portion of modern industrially used KBSs. Following this approach, the system is built of a number of modules

whose ways of c
Communication is
channels. This is a

There are se
advantage is the e
architecture of the sy
easily determined by
communication chann
basis to judge the ro
provides a valuable a
the system. This, in tu

The major disa
KBS is the system's
for certain groups of
change global proble

3.1.4. Blackbo

A distinct fami
approach initiated in
a. 1988). One c
systems was Stanfo
generic blackboard a

whose ways of cooperation are decided on at the development stage. Communication is usually done through sending messages over pre-defined channels. This is a clear example of RC.

There are several benefits in using hardwired methodology. The main advantage is the explicit control and ability to assess the knowledge-level architecture of the system. The role of each component of the system can be easily determined by its input-output information. In addition, pre-established communication channels between different components of the system give the basis to judge the role of every inter-connected sub-part of the KBS. This provides a valuable assistance in determining knowledge flaws and conflicts in the system. This, in turn, enables easy maintenance of such systems.

The major disadvantage of the hardwired approach to the development of KBS is the system's brittleness. That is, the system can function unpredictably for certain groups of input. Another drawback is the system's inflexibility to change global problem-solving strategy without *ad hoc* methods.

3.1.4. Blackboard-Based Systems

A distinct family of KBS architectures is the successor of the "blackboard" approach initiated in mid-1970s by HEARSAY-II project (Erman, Hayes-Roth et al. 1988). One of the first available shells for development of blackboard systems was Stanford KSL's BB1 (Johnson and Hayes-Roth 1987). BB1 has a generic blackboard architecture that is represented in Figure 2.

Figure 2. Blac

The software

KSs), contribute in

usually, each KS is

may follow any su

1995).

The blackbo

state. It could be

participants of the p

The control

system. Based on

next. The control s

Carver et..a

blackboard appro

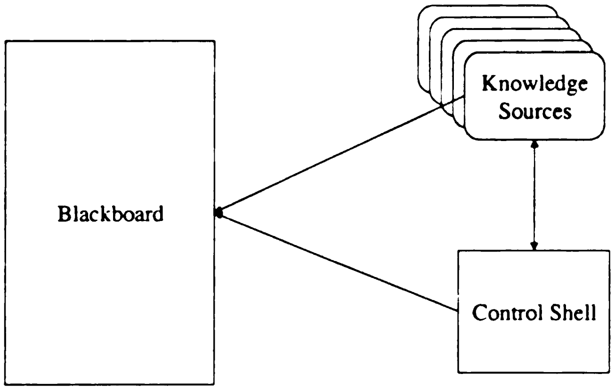


Figure 2. Blackboard Architecture

The software specialist modules, which are called knowledge sources (KSs), contribute information that would lead to the solution of the problem. Usually, each KS is a planar rule-based system, though KSs' internal architecture may follow any suitable problem-solving methodology (Vranes and Stanojevic 1995).

The blackboard is the database that keeps the current problem-solving state. It could be seen as a collection of global variables accessible by all participants of the problem-solving process.

The control shell governs the flow of problem-solving activity in the system. Based on the blackboard context it decides which of the KSs is to run next. The control shell is also traditionally implemented as a planar set of rules.

Carver et..al. (Carver and Lesser 1992) noticed a control problem with the blackboard approach: 'What makes blackboard control difficult is that it can be ighly problematic

because there may

The convent

modem augmentat

and RC methods at

source might be cons

(Hidium, Sade

the blackboard is pa

different sets of wor

different times. By

visible to knowledge

One of the a

hypothesis about th

1996). According

different aspects o

information. Thes

media, so differen

different modules.

Another

straightforwardnes

smaller chunks

simply implement

shared blackboa

highly problematic to determine expected value of KSIs [knowledge sources] because there may be complex interrelationships among the KSIs.'

The conventional blackboard approach follows the FC. Nevertheless, modern augmentations to this methodology leverage the convenience of SRC and RC methods at the knowledge source level. That is the separate knowledge source might be constructed by following these approaches.

(Hidlum, Sadeh et al. 1996) describe a blackboard-based system where the blackboard is partitioned into the number of contexts that correspond to different sets of working assumptions and different solutions and are active at different times. By doing this the developers limit the amount of knowledge visible to knowledge sources which, in turn simplifies the control.

One of the attractive features of the blackboard approach is that it fits the hypothesis about the nature of cognition (Laird, Newell et al. 1987), (Brooks 1986). According to this hypothesis, different modules that are responsible for different aspects of cognition are active all the time and constantly processing information. These modules post results of their processes on the common media, so different sources may use information collected and processed by different modules.

Another benefit of the approach is its implementational straightforwardness: given a problem, one needs to break the problem into smaller chunks without worrying about explicit chunks' interconnections, one simply implements a problem solver for each of the chunks and links them to a shared blackboard.

However, th

disadvantages. La

describe its know

blackboard-based K

solving role of any pa

From the de

blackboard KBS is b

Engineering (Schach

1. The coupling of

variable (the blac

affects:

1.1.Maintenance

another integ

variables;

1.2.Security: a F

allowing for p

2. The structure o

principle of coinc

which degrades

3. Data is not enca

access to the

security problem

However, the blackboard approach suffers from several significant disadvantages. Lack of explicit structural control knowledge and the inability to describe its knowledge-level architecture makes it difficult to maintain the blackboard-based KBS. Moreover it makes it impossible to describe the problem solving role of any part of the blackboard based KBS.

From the development point of view, the way in which a particular blackboard KBS is being created defies several major principles of Software Engineering (Schach 1997).

1. The coupling of different problem-solvers is achieved through the global variable (the blackboard) which is considered bad practice in SE because it affects:
   1.1. Maintenance: change is difficult, since with each instantiation of PS in another integrated KBS the developer needs to duplicate all global variables;
   1.2. Security: a PS may be exposed to more data than it needs therefore allowing for potential breach of security.
2. The structure of the generic control module (flat rule base) follows the principle of coincidental cohesion, i.e. it performs multiple unrelated actions, which degrades maintainability, and hinders reuse.
3. Data is not encapsulated, that is, every participating problem-solver has direct access to the variables on the blackboard, which potentially brings the security problems.

3.2 Task-Sp
System

3.2.1. Know

The importa
Leve Hypothesis (K
knowledge level in c
and its implementatio
should not be a to
relation to the repre
independent of the
consequently, sugg
also introduced the
terms of its knowle

KLH was cri
its lack of concern
only describe ex
recognizing an a
power. Neverthe
numerous resear
methods, behavio

3.2.2. Se

Clancey's
Classification, a
problem solving

3.2. Task-Specific Architectures and Integrated Knowledge-Based System

3.2.1. Knowledge Level Hypothesis

The important milestone in KBS research is certainly Newel's Knowledge Level Hypothesis (Newell 1980) (KLH), where he suggested the existence of a knowledge level in computer systems which lies directly above the symbol level and is implementation independent. Newell argued that the representation itself should not be a topic of interest, but rather the nature of knowledge and its relation to the representation. The use of the KLH allows systems description independent of their internal representation and implementation details, which, consequently, suggests a systematic approach to the system's analysis. Newell also introduced the notion of a rational agent that can be reasoned about in terms of its knowledge, goals, and behavior.

KLH was criticized in (Sticklen 1989; Velde 1991; Velde 1993) because of its lack of concern about control knowledge, its non-operational character (it can only describe existent systems and does not help build new KBS), its not recognizing an agent's ability to be decomposable, and its lack of predictive power. Nevertheless, the power that the KLH provides for KBS analysis guided numerous research thrusts in the KBS field that investigate problem solving methods, behaviors, and strategies.

3.2.2. Second Generation Expert Systems

Clancey's (Clancey 1985) analysis of MYCIN, his description of *Heuristic Classification,* and Chandrasekaran's (Chandrasekaran 1983) work on reusable problem solving models led to the crystallization of the idea that there are common inferenc

problem solving.

Architectures (TSA

Second Generation

then, methodolog

invariably rely on

architecture of the

McDermott's

components of ex

1992) are other e

methodologies d

knowledge from

chosen as basic

considers a task

tasks could be s

structure. Unf

decomposition c

decomposable e

Sticklen (

called Knowledg

hypothesis is th

agent can be u

[5] will discuss t

common inference strategies or inference structures that can be used for problem solving. This is the working hypothesis of the Task Specific Architectures (TSA) 'branch of' KBS research, which heralded the beginning of Second Generation of Knowledge-Based Systems in the early eighties. Since then, methodological approaches to second generation expert systems almost invariably rely on mapping the structure of a *knowledge level* model to the architecture of the application.

McDermott's (McDermott 1988) role limiting models, Steel's (Steels 1990) components of expertise, and KADS knowledge sources (Balder and Akkermans 1992) are other examples of generalized inference strategies[5]. The mentioned methodologies differ in several aspects: separation of the domain factual knowledge from the problem-solving knowledge, granularity of tasks that are chosen as basic inferences, etc. Nevertheless, each of the TSA approaches considers a task decomposable into a set of sub-tasks, where each of the sub-tasks could be solved by the application of a particular, generalized inference structure. Unfortunately, KLH does not provide insights on how this decomposition could be analyzed. On the contrary, it regards an agent as a non-decomposable entity.

Sticklen (Sticklen 1989) suggested an augmentation to KLH which he called Knowledge Level Architecture Hypothesis (KLAH). The main idea of this hypothesis is that if an agent is decomposable into sub-agents, then the larger agent can be understood at the knowledge level by a knowledge level description

[5] I will discuss the details of each mentioned methodology in the later sections

of sub-agents and
structure. Another
possible to create a
problem solver for
architecture. The p
majority of the criti
(Stocken 1989), altn
decomposability. Y
investigated by seve

Steels (Steels
which lies directly ab
decomposed into m
the sub-tasks execu
task layer where
structures. KLH h
agent's decomposa

All task-sp
decomposability in
pre-defined proble
Generic Tasks (C
task decompositi
methods (McDerm

of sub-agents and specifying the sub-agents' integration and communication structure. Another important contribution of KLAH is that it suggested that it is possible to create a model of a decomposable agent and use that model as a problem solver for validating the agent's behavior, finding conflicts in the architecture. The paper was critiqued in (Clancey 1989; Slator 1989), but the majority of the criticism was targeted at the philosophical issues raised in (Sticklen 1989), although not all the critics agreed with the idea of agents' decomposability. Yet, the similar ideas about agent's decomposability were investigated by several researchers in KBS community.

Steels (Steels 1990) defined a similar idea of *knowledge use* level: a level which lies directly above the knowledge level and which focuses on how a task is decomposed into manageable sub-tasks, what ordering should be imposed on the sub-tasks execution, etc. A similar idea underlines the concept of the KADS *task layer* where primitive inferences are combined in the problem-solving structures. KLH has evolved to the stage where it accepts the concept of an agent's decomposability.

All task-specific methodologies are based on the idea of task's decomposability into a number of subtasks, each of which could be handled by a pre-defined problem-solving method. However, the brittleness of large-grain Generic Tasks (Chandrasekaran and Johnson 1993) and vague approaches to task decomposition in small-grain problem-solving methods as in role limiting methods (McDermott 1988) and KADS (Wielinga and Schreiber 1994) caused researchers to

decomposition and

3.2.3. GENE

The idea

way in which ge

application doma

plants, or disease

problem solving

1990)

The Generic

focused on a numb

The GT approach

Ohio State Univer

70s – early '80s

This approach in

shells that sim

implementation.

- Taxonomic C

a situation as

(Chandrasek

classification

- Function-bas

works, ans

researchers to develop tools and methodologies that facilitate task decomposition and sub-task integration processes.

3.2.3. GENERIC TASKS

> 'The idea of generic task is not that interesting until we realize that the way in which generic tasks are executed shows many similarities across application domains. [for example] In the diagnosis of circuits, cars, power plants, or diseases, significant elements are in common, specifically, the same problem solving methods and the same types of domain models. ' (Steels 1990)

The Generic Task (GT) approach is a "large grain" view of problem solving focused on a number of primitive problem solving types - called "generic tasks". The GT approach was developed at the Laboratory for Intelligent Research at Ohio State University by a team of researchers led by Chandrasekaran in late '70s – early '80s (Chandrasekaran 1983; Chandrasekaran and Johnson 1993). This approach includes task –level implementation, reusable, and executable shells that simultaneously support knowledge acquisition and system implementation. The list of primitive generic tasks is as follows.

- Taxonomic Classification Task – Classify a (possibly complex) description of a situation as an element, as specific as possible, in classification hierarchy (Chandrasekaran, Mittal et al. 1979) (usually referred to as "hierarchical classification")
- Function-based Reasoning - Given a causal understanding of how a "device" works, answer "what would happen if" type questions (Sticklen and

Chandrasekara

abstraction")

- Knowledge Dir

obtain attribute

datum

- Object Synthes

satisfying certa

referred to as "ro

- Hypothesis mat

describe the pro

- Abductive Asse

explained by th

each associate

portion of the d

of the given hyp

It could be

GTs covers all po

a convenient mec

for the task at han

(Sticklen, Kamel

1995; El-Sheikh,

Penney et al. 1

1997), etc.

Chandrasekaran 1985) (this primitive task was originally called "state abstraction")

- Knowledge Directed Information Passing – Given attributes of some datum obtain attributes of some other datum, conceptually related to the original datum
- Object Synthesis by Plan Selection and Refinement – Design an object satisfying certain specification (Brown 1987) (this primitive task is usually referred to as "routine design")
- Hypothesis matching – Given a set of hypothesis and a set of datum that describe the problem state, decide if the hypothesis matches the situation
- Abductive Assembly of Explanatory Hypotheses – Given a situation to be explained by the best explanatory account, and a number of hypotheses, each associated with degree of belief, and each of which offers to explain a portion of the data, construct the best composite explanatory hypothesis out of the given hypotheses.

It could be said that one could never be sure that the list of above sets of GTs covers all possible situations. Nevertheless, the GT methodology provides a convenient mechanism for identifying an appropriate problem-solving method for the task at hand and has been effectively used for the development applied IS (Sticklen, Kamel et al. 1992; Sticklen, Kamel et al. 1992; Moy, McDowell et al. 1995; El-Sheikh, Sticklen et al. 1996; Lenz, McDowell et al. 1996; El-Sheikh, Penney et al. 1997; Kamel, Lukibanov et al. 1997; McDowell, Sticklen et al. 1997), etc.

ISL's Gen

Lukianov et al.
the Generic Task a
problems with we
KBSs built in this
participating specia
specified. Limitatio
the systems behav
leads to the nece
outside of the shell.

TIPS (Punc
GT-based problem
collection of proble
integrated KBS w
into GT-based pro
with a distinct sp
Finally, it is neces
multiple selectors
comparing to the
knowledge and it
directly call anoth
mechanism on ho

ISL's Generic Task Toolset (Sticklen, Kamel et al. 1992; Kamel, Lukibanov et al. 1997), supports the development of integrated RC KBS within the Generic Task approach. This tool supports development expert systems for problems with well-defined and run-time unalterable control flow. KLA of the KBSs built in this tool set is also easily identifiable, since the roles of each participating specialists and each connected group of specialists are explicitly specified. Limitations of this method are evident: inability to dynamically adjust the systems behavior and inability to adopt flexible control architecture, which leads to the necessity to hard-code control mechanisms programmatically outside of the shell.

TIPS (Punch 1989) is an architecture that allows integration of existing GT-based problem-solvers using sponsor-selector mechanism. TIPS is a collection of problem-solvers linked together via sponsor selectors. To create an integrated KBS within the TIPS framework the problem is first to decompose it into GT-based problem-solvers. Then each problem-solving unit is associated with a distinct sponsors-selector that controls the problem solver's activation. Finally, it is necessary to specify a priority order for conflict resolution in case of multiple selectors firing. There are definite advantages in the TIPS architecture comparing to the pure blackboard approach: it allows leveraging domain control knowledge and it supports vertical integration where one problem solver can directly call another. The main drawback of TIPS is that it does not specify the mechanism on how knowledge is shared between problem solving modules.

Another int

1991) is based o

author proposed

choose between a

achieve the given

approach is simila

CommonKADS co

methods for inte

approach (Glaser 1

3.2.4. PRO

PROTEGE-

tool that generates

problem solving m

KBS crea

architecture throu

connected collec

advocates flexible

planned dynamic

Nevertheless, the

model inside the

of C.

Another interesting GT-based integration approach **GT-SOAR** (Johnson 1991) is based on SOAR's problem spaces (Laird, Newell et al. 1987). The author proposed to use SOAR mechanism of problem spaces to dynamically choose between available small-grain problem-solving methods in order to achieve the given task described in terms of initial state and goal state. This approach is similar to that proposed in several research directions within CommonKADS community, where researchers report on development of methods for integration of small-grain problem-solvers using blackboard approach (Glaser 1996) and using specialized grammars (Brazier 1997), etc.

3.2.4. PROTÉGÉ

PROTÉGÉ-II (Tu, Kahn et al. 1989; Tu, Eriksson et al. 1995) is a meta-tool that generates task-specific expert-system shells from libraries of reusable problem solving methods, domain-ontology, and knowledge acquisition modules.

KBS created in PROTÉGÉ-II has strong feel of knowledge-level architecture through emphasizing explicit role of each participating module or connected collection of modules inside the expert system. PROTÉGÉ-II advocates flexible control architecture where the sequence of action could be planned dynamically during the run-time exploiting agenda-based control. Nevertheless, the implementation does not allow explicit stating of the control model inside the shell and requires additional hard-core programming in CLIPS or C.

3.2.5. KAD

The KADS

the view that the

developer defines

them. According

during the KBS

Task Model, Mod

Design model. M

1. Static doma

calculus. Th

objects etc.

2. Knowledge

methods su

knowledge

3. The *Task* le

can be com

4. The Strate

particular p

Numer

built over yea

were rooted

decompositio

KADS was
organization

3.2.5. KADS

The KADS[6] (Knowledge Analysis and Design Support) community takes the view that the development of KBS is a modeling activity during which the developer defines different layers of KBS and specifies connections between them. According to KADS there are several models that need to be specified during the KBS development cycle: Organizational model, Application model, Task Model, Model of Expertise, Model of cooperation, Conceptual model, and Design model. Model of expertise – the core of the KBS – consists of four layers:

1. *Static domain knowledge* is based on first-order order-sorted predicate calculus. This level contains domain factual knowledge, relations between objects etc.
2. *Knowledge sources* – canonical inference steps - are the traditional generic methods such as classification, abduction, etc. In addition, the methods for knowledge *transformation*, *selection* and *computation*.
3. The *Task* level contains knowledge about how elementary inference engines can be combined to achieve a certain goal.
4. The *Strategic knowledge* level determines what goals are relevant to solve a particular problem.

Numerous KBS development environments that follow this approach were built over years. This section overviews several KBS building environments that were rooted in the KADS methodology, but differ in their approach to task decomposition-integration problem.

[6] KADS was an ESPIRIT-I European multinational project that included organizations from France, Germany, Netherlands, and United Kingdom

The **Com**
promotes design
ontology (*static do*
connection betwe
communication ch
therefore implemen
difficulty that the K
small-grain problem
hand, having done
performed seamles

VITAL (Mc
KADS. VITAL tak
the initial problem
GDM (generalize
application of rule
represent basic
approach support
down sentence-re
channels betwee
established auton
formalism that the
at hand. This fo
adequately.

The **CommonKADS** methodology (Wielinga and Schreiber 1994) promotes design time integration of problem solving methods using domain-ontology (*static domain knowledge model*) as communication glue. By defining a connection between the layers of the KBS developer states that the explicit communication channels and control strategy during system developing time, therefore implementing RC or SRC global problem-solving strategy. The main difficulty that the KBS developer faces following this approach is how to combine small-grain problem-solving components into a consistent model. On the other hand, having done this the integration of different parts of the system is being performed seamlessly.

VITAL (Motta, O'Hara et al. 1994) is another approach originated within KADS. VITAL takes the iterative viewpoint on the KBS building process, where the initial problem statement is formalized using special kind of grammar called GDM (generalized directive model). Then the initial GDM is refined by application of rules of GDM grammar until it reaches terminal symbols that represent basic inference mechanisms. Unlike CommonKADS, the VITAL approach supports combination and integration of generic components by top-down sentence-refinement process. During this process, all the communication channels between the inference components along with their execution order are established automatically. The weakness of this methodology is in the extreme formalism that the KBS developer has to follow in order to describe the problem at hand. This formalism is not always capable of reflecting the given problem adequately.

Glasser (G

the integration o

CoMoMAS supp

instead of determ

argues for the bla

the described app

knowledge, which is

The Table 1

development I-KBS

framework I-KBS c

	E K
BB1, BBK	1
PROTEGE-II	3
Generic Task Toolset	5
TIPS	4
GT SCAR	3
CommonKADS	5
VITAL	5
CoMoMAS	2
Conventional FR	5
Augmented FR	

Table 1. Co

Glasser (Glaser 1996) proposed an opportunistic blackboard approach to the integration of CommonKADS inference engines. The described system **CoMoMAS** supports a model-oriented approach to the construction of KBS, but instead of determining the cooperation model during the design stage CoMoMAS argues for the blackboard-like run-time dynamic cooperation model. However, the described approach does not allow explicit representation of existent control knowledge, which is its main drawback.

The Table 1 demonstrates comparison between discussed system for development I-KBSs. The bottom row is reserved for the augmented FR framework I-KBS construction that is one of the targets of the reported research.

	Explicity of KLA (1 - 5)	Global control strategy	Local control strategy	Ability to handle multiple control structures	Shell
BB1, BBK	1	FC	FC	No	Yes
PROTÉGÉ-II	3	FC	All	Yes	Yes
Generic Task Toolset	5	RC	RC	No (programming is needed)	Yes
TIPS	4	SRC	RC	No	No
GT SOAR	3	RC	FC	No	No
CommonKADS	5	SRC	RC	Some	Several
VITAL	5	RC, Some SRC	RC	No	Yes
CoMoMAS	2	FC	FC	No	No
Conventional FR	5	SRC	RC	No	Yes
Augmented FR	**?**	**?**	**?**	**?**	**?**

Table 1. Comparison between the discussed approaches

3.3. Ontolo
Knowled

Knowledge

composed syste

ensures that the

for problem solvi

developed speci

applications. T

sharing.

3.3.1. Sy

One of th

translators that

knowledge repre

MacGregor 19

parts of proble

terminological r

information is tr

and back. Inte

from one repres

FRAME_BASE

Frame_name: C
Maker:
Model:
Year: 19
Color: B

Table 2.
representations

3.3. Ontologies and Other approaches to Knowledge Sharing in Knowledge-Based Systems

Knowledge sharing is one of the keys to the successful development of a composed system. Use of shared knowledge between cooperating agents ensures that the agents are able to understand common information necessary for problem solving. Moreover, it is likely that domain knowledge representation developed specifically for co-operated problem solving would be reused in later applications. The following sub-sections describe approaches to knowledge sharing.

3.3.1. Syntactic Mapping

One of the strategies to achieve the inter-agent understanding is to create translators that will transform knowledge representation of one agent to knowledge representation of another agent when needed. In LOOM, for example (MacGregor 1991), different type of reasoning engines were used for different parts of problem solving: semantic net reasoning engine was responsible for terminological reasoning, another – used logical representation. In this case, information is translated from semantic network representation to Horn's clauses and back. Interfaces between different types of agents translated information from one representation to another (Table 2).

FRAME_BASED REPRESENTATION	UNARY PREDICATS
Frame_name: Car Maker: Mercury Model: Mystique Year: 1995 Color: Blue	Frame_name(Car) & Maker (Mercury) & Model (Mystique) & Year (1995) & Color (Blue)

Table 2. A possible mapping between frame-based and unary predicates representations

Another e

Knowledge Bas

Frame-Based (F

(FOPC) represe

this case transla

Disadvan

Schreiber et al.

- Fixed syntac

example, if

of the repr

knowledge

ability of a p

- Sometimes

another.

- It is not feas

general.

Meanwh

procedures tha

developing hig

"hooking up" pr

characterized a

- small numbe

Another example of syntactic integration is CYC (Lenat and Guha 1990). Knowledge Base in CYC is divided into two levels: convenient for inference Frame-Based (FB) representation is backed by First-Order Predicate Calculus (FOPC) representation in order to "... provide the requisite expressiveness". In this case translation is be made in both direction FOPC ↔ FB.

Disadvantages of this type of integration are following (van Heijst, Schreiber et al. 1997):

- Fixed syntactical mapping restricts expressiveness of representations. For example, if we are to express some piece of knowledge in some way in one of the representational formalisms, this will put constraints on how this knowledge will be represented in the other KR. This also may limit reasoning ability of a problem-solver as well as expressiveness of knowledge model.
- Sometimes there is no obvious way to map one syntactical representation to another.
- It is not feasible to make such converters for every possible representation in general.

Meanwhile, the simplicity of this methodology allows *ad hoc* mapping procedures that can be written on the fly and do not require lead-time for developing high-level system architecture. This method can also be used for "hooking up" previously developed agents with new software. Such I-KBSs are characterized as follows:

- small number of cooperating agents;

- similar syntac

- no co-operati

KBS) is plann

3.3.2. KIF

An appro

developed unde

1992). As a pa

developed Kno

interchanging in

inherits LISP's

basic semantic

variables) and

world. Every

sentence is eit

the preset con

for handling no

supports First

additions.

This re

representation

The DARPA
conventions f
based system
supporting te
broadly funct

- similar syntactic structure of knowledge used by the agents; and
- no co-operation with "foreign" agents (agents, acting outside this particular I-KBS) is planned.

3.3.2. KIF: Knowledge Interchange Format

An approach to standardization of knowledge representation is being developed under DARPA support in the KSE project[7] (Genesereth, Fikes et al. 1992). As a part of this project, a group of researchers at Stanford University developed Knowledge Interchange Format (KIF) – a formal language for interchanging information among disparate computer programs. This language inherits LISP's syntax and is a declarative representation of knowledge. The basic semantics of KIF is a correlation between the terms (constants and variables) and the sentences of the language and a conceptualization of the world. Every term denotes an object in the universe of discourse, and every sentence is either true or false. In order to be true, the sentence should satisfy the preset conditions and axioms. The authors also introduce non-monotonicity for handling non-monotonic knowledge. In other words, KIF is a language that supports First Order Predicate Calculus representation of knowledge with few additions.

This research is, in fact, the first attempt to standardize knowledge representation (KR). The authors filed a draft with the American National

[7] The DARPA Knowledge Sharing Effort (KSE) is a consortium to develop conventions facilitating sharing and reuse of knowledge bases and knowledge based systems. Its goal is to define, develop, and test infrastructure and supporting technology to enable participants to build much bigger and more broadly functional systems than could be achieved working alone.

Standard (Gene

standard for repr

This meth

sound logic theo

experience with

representation h

Chandrasekaran a

method:

- Different interpr

satisfiable in one

- Non-logical type

represented usin

- Many KR schem

monotonic reaso

and so forth. Giv

impossible to cre

in KR without lim

- Assuming the hy

translation of an

For example, a t

an axiomatizatio

- Expressing arbit

can lead to tailo

Standard (Genesereth 1998) in which they suggested that KIF should be a standard for representation of shared knowledge.

This method has many attractive properties: clarity of representation; sound logic theory, which allows making proofs; with a few hundred years of experience with predicates handling and manipulating. Nevertheless, this representation has some noticeable shortcomings (Ginsberg 1991; Chandrasekaran and Johnson 1993) that can diminish the usefulness of this method:

- Different interpretations can treat the same sentence differently, e.g. what is satisfiable in one interpretation may not be in the other.
- Non-logical types of knowledge (e.g. images, video, and audio) can not be represented using KIF.
- Many KR schemes involve different extensions of FOPC, such as non-monotonic reasoning, reasoning with limited data, reasoning with noisy data, and so forth. Given all the differences in modern approaches to KR it seems impossible to create a language that will capture all current and future trends in KR without limiting the latter.
- Assuming the hypothesis that everything can be expressed in FOPC, the translation of an arbitrary knowledge into this form may be a very difficult task. For example, a translation of a probabilistic database into FOPC must include an axiomatization of the theory of probability itself.
- Expressing arbitrary knowledge using any fixed and standardized KR method can lead to tailoring knowledge to this representation.

3.3.3. K

The sco

interactions be

environment.

communication

between a kno

system. The

Labrou et al.

the external in

The ba

common kno

necessary) to

pragmatic iss

- Which age
- How to in
- What dom

KQML

KQML

capable of p

registry of se

use these fa

Facilitators m

Domain Nam

3.3.3. KQML: the pragmatic approach

The scope of the External Interfaces Group within KSE is the run-time interactions between knowledge-based systems and other modules in a run-time environment. Special attention has been given to two important cases – communication between two knowledge-based systems and communication between a knowledge-based system and a conventional database management system. The Knowledge Query and Manipulation Language (KQML) (Finn, Labrou et al. 1997) language is one of the main results, which has come out of the external interfaces group of the KSE.

The basis for the KQML approach is the understanding of the fact that common knowledge representation is not enough (or may not even be necessary) to make the communication between agents work. Some important pragmatic issues should be addressed first:

- Which agent to communicate with and how to find them.
- How to initiate and maintain exchange.
- What domain ontology to use.

KQML is concerned primarily with such pragmatic issues.

KQML introduces a new class of agents - *communication facilitators* - capable of performing tasks for communication purposes, such as maintaining registry of service names, forwarding messages, routing messages, etc. Agents use these facilitators when they need to send a message to another agent. Facilitators may access other agents using <machine: agent name> with use of Domain Naming Service across the Internet.

KQML is

list is performat.

In the me

about PRICE IF

language LPROL

Figure 3

Much e

used KQML f

distributed CY

Object Mana

(CORBA) (Be

3.3.4.

In orde

of discourse

conceptualiza

KQML is based on the balanced parenthesis list. The first element of the list is performative and the remaining are parameters.

In the message in Figure 3 the agent `stock-server` asks for one reply about `PRICE IBM`, and wishes the value to be returned in variable `?price` using language `LPROLOG` and ontology `NYSE-TIKS`.

```
(ask-one
        :content (PRICE IBM ?price)
        :receiver stock-server
        :language LPROLOG
        :ontology NYSE-TICKS)
```

Figure 3. Example of KQML query.

Much experience has been gathered from the variety of projects which used KQML for inter-agent communication: Microsoft OLE2, experiment with distributed CYC based agents over local network at the University of Maryland, Object Management Group's Common Object Request Broker Architecture (CORBA) (Ben-Natan 1995), Xerox's ILU etc.

3.3.4. Ontologies and Ontolingua

In order for an agent to reason and exchange information about a domain of discourse, it must use a conceptualization of that domain. This conceptualization should provide a vocabulary for representing knowledge about the domain.

ontologies[8] (C

Ontolo

Once create

- provides

knowledg

- provides

and the K

- serves as

Resea

the developm

Server (Fike

maintenance

KIF (which

representatio

between clas

domain know

The C

over the Inte

load, edit, a

[8] This def
is understo
will use term

the domain. These conceptualizations are usually called domain ontologies or ontologies[8] (Gruber 1992)

Ontologies play different roles in knowledge-base development cycle. Once created for a particular domain the ontology:

- provides the developer with a domain dictionary for representing domain knowledge;
- provides with the glossary that is available for interacting between the user and the KBS
- serves as an interoperation dictionary for agents that comprise KBS.

Research of Knowledge System Laboratory at Stanford University led to the development of *Ontolingua* (Gruber 1992; Gruber 1993) - the tool (Ontolingua Server (Fikes 1997)) and a methodology that helps in the development, maintenance, and use of the domain ontologies. The backbone of *Ontolingua* is KIF (which is augmented with the frame-base representation). This representation allows expressing class – subclass hierarchies, relationships between classes, functions on relations, and instances of classes that describe domain knowledge.

The Ontolingua Server enables collaborative development of ontologies over the Internet. Once logged into the server the user has the ability to create, load, edit, and save ontologies. The tool also gives possibilities to maintain the

[8] This definition is different from the definition of *Ontology* in philosophy where it is understood as a theory about that what can exist. Throughout this proposal, I will use term ontology in the KBS sense.

ontology, che

IDL, etc.

Three

idea of inter

collaborative

ontologies ov

vocabularies

CommerceN

specification

development

(Gennari, D

vocabulary.

Simil

project. De

Schreiber 1

3.3.5

The

the softwar

of building

developme

during whic

and then v

system.

ontology, check inconsistencies, translate to and from KIF, LOOM, CORBA's IDL, etc.

Three main impacts of the Ontolingua research are (i) crystallizing the idea of inter-agent communication through use of ontology, (ii) enabling the collaborative development of the domain ontologies, and (iii) sharing the domain ontologies over the Internet. Different research groups for the development of vocabularies are using the Ontolingua Server. Examples of such projects are CommerceNet - providing Internet accessibility to products' descriptions and specifications, support of Enterprise project (Uschold and King 1995) in development ontologies description of business process, and InterMed project (Gennari, D. E. Oliver et al. 1995) – providing Internet-based medical vocabulary.

Similar approach to knowledge sharing was suggested in CommonKADS project. Detailed examples of this approach can be found in (Wielinga and Schreiber 1994).

3.3.5. Tools for Developing Domain Ontologies

The development of domain ontology often precedes the development of the software systems that uses it. However, it is often the case that the process of building domain ontology stretches for the duration of the software development cycle. An example of such development is the Socharis project, during which the ontology of manufacturing with composite material was built first and then was augmented many times during the development of the Socharis system.

tured

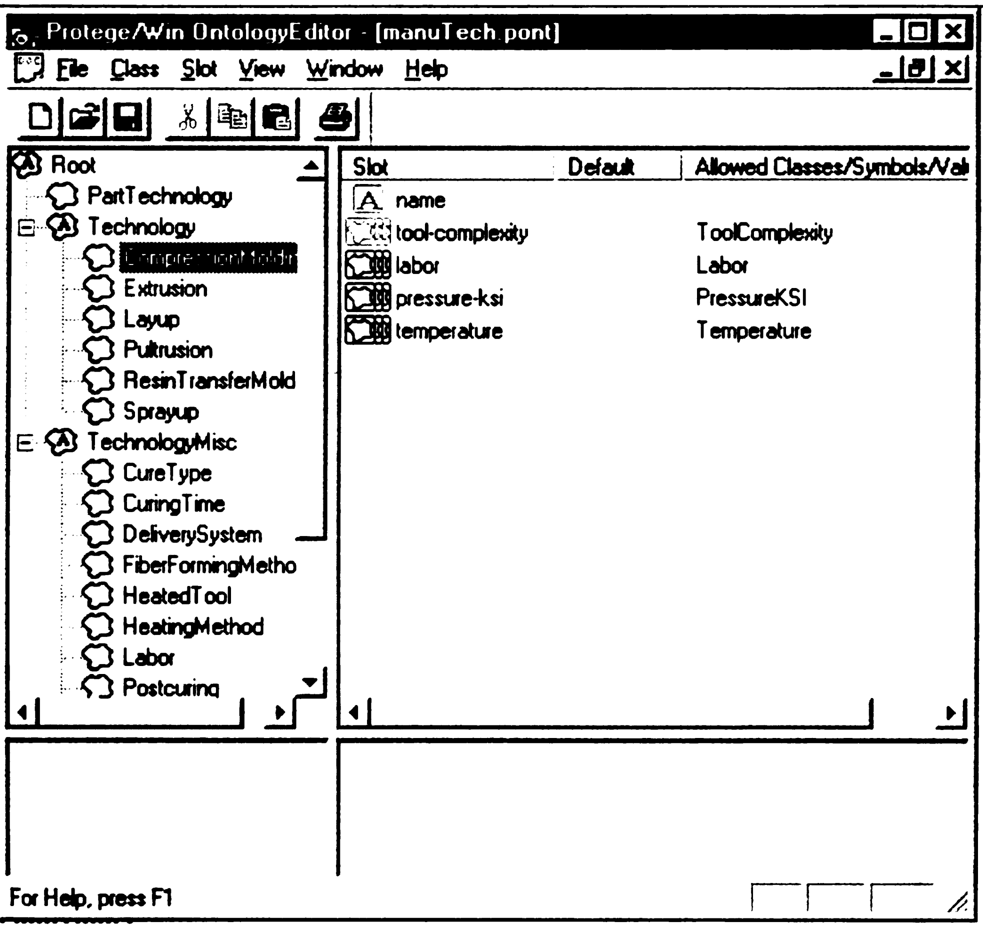
dissert

Append

of too s

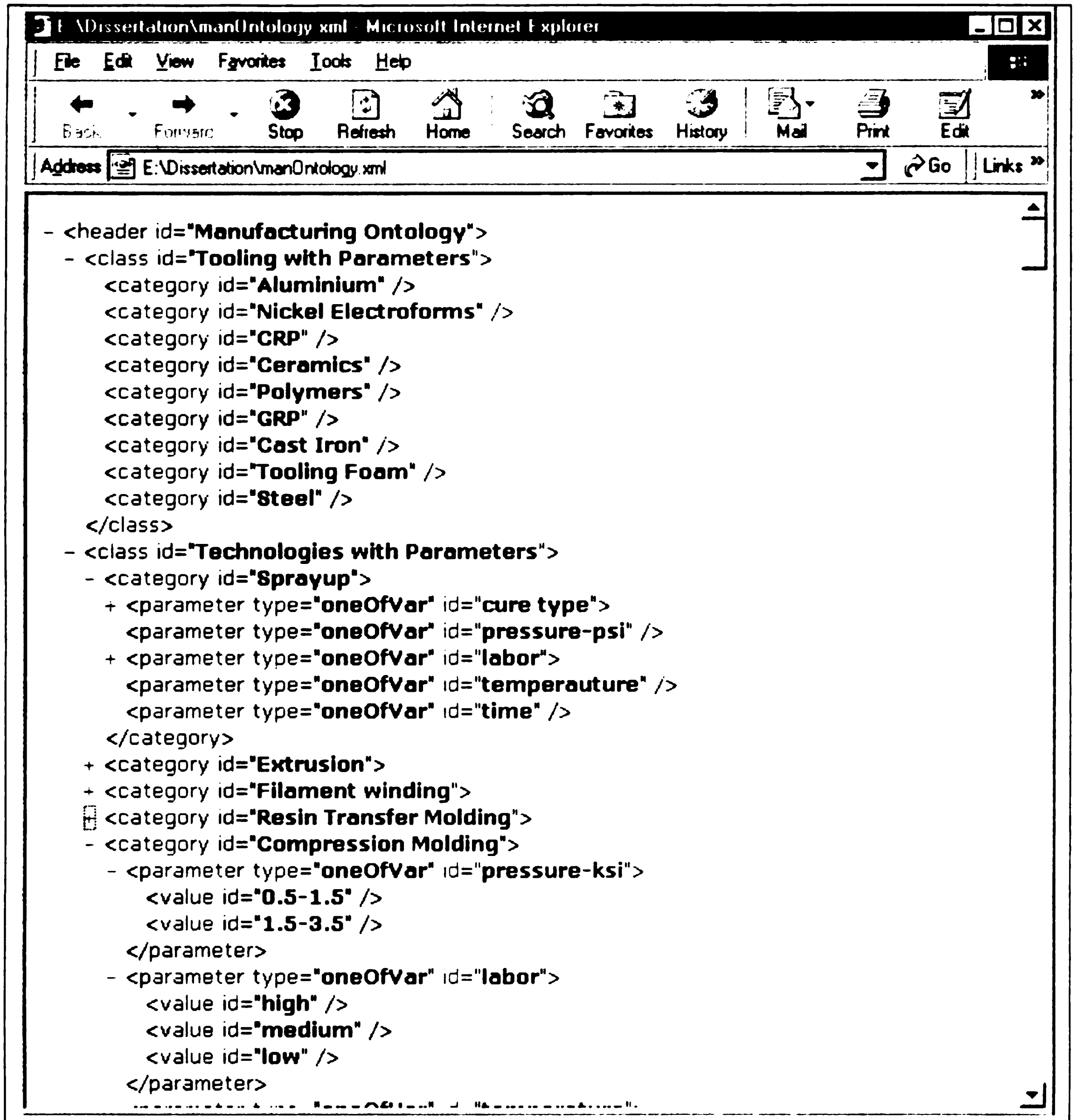
Figu

The authors of Socharis developed Ontology Editor that was specifically tuned to represent composite manufacturing ontology. As a part of the dissertation research, this ontology was duplicated using Protégé (Figure 4, Appendix C) and XML (Figure 5, Appendix D) to demonstrate interchangeability of tools and methods.



Slot	Default	Allowed Classes/Symbols/Val
name		
tool-complexity		ToolComplexity
labor		Labor
pressure-ksi		PressureKSI
temperature		Temperature

Figure 4. Screenshot of the Manufacturing Ontology in Protégé shell



```
- <header id="Manufacturing Ontology">
  - <class id="Tooling with Parameters">
      <category id="Aluminium" />
      <category id="Nickel Electroforms" />
      <category id="CRP" />
      <category id="Ceramics" />
      <category id="Polymers" />
      <category id="GRP" />
      <category id="Cast Iron" />
      <category id="Tooling Foam" />
      <category id="Steel" />
    </class>
  - <class id="Technologies with Parameters">
    - <category id="Sprayup">
      + <parameter type="oneOfVar" id="cure type">
        <parameter type="oneOfVar" id="pressure-psi" />
      + <parameter type="oneOfVar" id="labor">
        <parameter type="oneOfVar" id="temperauture" />
        <parameter type="oneOfVar" id="time" />
      </category>
    + <category id="Extrusion">
    + <category id="Filament winding">
    + <category id="Resin Transfer Molding">
    - <category id="Compression Molding">
      - <parameter type="oneOfVar" id="pressure-ksi">
          <value id="0.5-1.5" />
          <value id="1.5-3.5" />
        </parameter>
      - <parameter type="oneOfVar" id="labor">
          <value id="high" />
          <value id="medium" />
          <value id="low" />
        </parameter>
```

Figure 5. screenshot of MSIE 5 displaying re-developed in XML Manufacturing Ontology

Both methods proved adequate for developing domain ontologies. However, there are several differences between them:

- Protégé is a tool for developing knowledge-acquisition shells and the developing working ontology is a part of the process. As a result of the deve

also

C

ontology

Since th

design o

the doma

3.4.

3.4

Th

introduce

represent

the repres

decompos

functiona

in the soft

denote pri

represent o

The

metaphor o

methods for

published in

developed in Protégé framework, ontology conforms to the internal format but also could be exported in to a Lisp like balanced parenthesis list.

On the other hand, a number of commercially available XML editors let the ontology designer create ontology by allowing defining hierarchical structures. Since the focus of XML editors is on the generality, it could take more time to design ontology in any of them. However, the result – an XML representation of the domain ontology – is far more accepted than this of Protégé shell.

3.4. Related Research in Software Engineering

3.4.1. From Data Flow Diagrams to the Object Modeling Technique

The data-flow diagram (DFD) (Stevens, G. J. Myers et al. 1974) was introduced in the early seventies to aid in a structured design of software, representing data and the processes that transforms data. The DFD supports the representation of data and processes that transform the data by systematic decomposition of the system, therefore providing means for describing the functionality of the system. The DFD allows tracing information and control flow in the software system by following links in a directed graph where vertices denote primitive operators (true/false test, assign, etc) and directed edges represent data and control flow.

The Object-oriented paradigm, the successor of the DFD, is based on the metaphor of a software system as an object that encapsulates data, and has methods for manipulating this data. Different objects can communicate through published interfaces. This approach has numerous advantages over the DFD

(abstraction, m
expressively re

The prob
of constructing
description of sc

OMT is c
complimentary v
system. A forma
language based
system through
of the process, a
dynamic model
diagrams. The
mechanism is
Cheng 1998).

Universa
derivative of O
the problem de
functional parts
responsibilities
channels and
wire-frame co
used as an ob

(abstraction, modularity, encapsulation, reuse, etc.). Nevertheless, it fails to expressively represent the overall functionality of the system.

The problem attacked by the Object Modeling Technique (OMT) is the one of constructing formal software specification from an informal high-level description of software requirements and specifications.

OMT is composed of object, dynamic, and functional models to provide complimentary views that graphically describe different aspects of a software system. A formal software model is usually described in terms of a specification language based on process algebra (e.g. LOTOS) that allow description of the system through its input, output, a behavioral expression that models the activity of the process, and a set of post-conditions. State diagrams usually describe the dynamic model of the system, and the functional model is captured in data flow diagrams. The main flaw of this approach is that a well-defined integration mechanism is virtually non-existent (Wang, Ritcher et al. 1997; Wang and Cheng 1998).

Universal Modeling Language (UML) (Fowler and Scott 2000) is a derivative of OMT that enables the development of software systems starting at the problem definition stage. The development starts with identifying – actors – functional parts of the system (a frame for future objects) and use cases (i.e. responsibilities of the actors). Next the developer defines communication channels and refines object definitions. The final steps are the generation of wire-frame code in an object-oriented language (such as C++ or Java) that is used as an object interface definition.

The des

software produ

development. T

1. During the c

deals with n

shelf"). Ther

which underli

2. The main co

map informa

in KBS dev

already code

manageable

3. The main bo

spends the

reflected in a

Neverthe

cooperating par

It may also be a

further analysis,

3.4.2. Co

Reuse of

productivity and

The described methods while suited for the developing of a common software product unfortunately cannot be applied directly to the area of KBS development. The reasons for this are as follows:

1. During the development of second generation KBS, the engineer usually deals with modules of well-defined functionality (often available "off-the-shelf"). Therefore making it unnecessary to formally describe the algorithm, which underlies the problem-solving process.
2. The main concerns for KBS developers are similar to the OMT task: how to map informal specification onto formal methods. The difference is as follows: in KBS development, usually all methods for information processing are already coded and the problem is how to decompose the original task into manageable pieces that will allow direct application of the existent module.
3. The main bottleneck in the KBS development cycle, where the KBS engineer spends the most time, is knowledge acquisition. This major problem is not reflected in any of formal methods for software development.

Nevertheless, OMT provides a convenient metaphor for representing the cooperating parts of KBS and defining information transfer between those parts. It may also be advantageous to use OMT formalism to describe the system for further analysis, maintenance, and reuse.

3.4.2. Component Reuse

Reuse of existing components in constructing software systems improves productivity and the quality of the developed products. Much of the effort of software engin

that allow effect

The esse

"right" compone

description are :

keywords cannot

the semantics an

formalize compo

Authors suggest u

components. Lar

assumptions (serv

by a component), a

This metho

to choose a comp

conditions, and se

aspects of the con

example, it is often

therefore, it would

component can pe

possibly required du

- various hardwa

requirements, pro

software engineering lately was concentrated on the development of methods that allow effective reuse of existing modules (Mili, Mili et al. 1995).

The essential step in the component reuse process is the selection of the “right” component. It is often the case that interface specification and keyword description are the only available criteria for choosing a component. However, keywords cannot convey all useful information, unless a special agreement on the semantics and pragmatics of the keywords exists. One of the attempts to formalize component description was reported in (Chen and Cheng 1997). Authors suggest using the Larch family of specification languages to describe the components. Larch languages allow description of the component in terms of *assumptions* (services required by a component), *capabilities* (services provided by a component), and *domain theory* (an algebraic model of a domain).

This method enables the software design process (automated or manual) to choose a component according to its logical specifications, pre- and post-conditions, and services, provided or required. However, the other important aspects of the component's functionality are not taken into consideration. For example, it is often required to perform a certain task in a given amount of time; therefore, it would be desirable to know at least approximately, how fast a component can perform on a given platform. Other examples of information possibly required during selection time include:

- various hardware concerns (required networking bandwidth, memory requirements, processor architecture, etc.),

- software c
protocol use
- KBS conce
maintainab.

3.4.3. Pe

The Pers
developed at Sc
provides a softwa
of software produ
the senses that
formalizing the pr
Maturity Model (C
practices in softwa

PSP cons
recording data, a
size counting. F
development.

The cyclic p
software-engineeri
divide-and-conque

PSP was in
projects in several c

- software concerns (e.g. implementation language, deployment platform, protocol used), and
- KBS concerns (domain ontology, explanatory possibilities, knowledge base maintainability, and so forth).

3.4.3. Personal Software Process

The Personal Software Process (PSP) (Ferguson, Humphrey et al. 1997) developed at Software the Engineering Institute at Carnegie Mellon University provides a software engineer with a methodology for the consistent development of software products. This methodology is different from that described above in the senses that it does not require formalizing of an algorithm, but rather formalizing the programming discipline. The authors of PSP use the Capability Maturity Model (CMM) (Hayes and Zubrow 1995) which addresses management practices in software development.

PSP consists of a series of scripts that define tasks and forms for recording data, and standards that govern such things as coding practices and size counting. Figure 6 represents the general cycle of PSP for software development.

The cyclic process in the middle of Figure 6 is built on several well-known software-engineering principles: the modular design concept, versioning, and the divide-and-conquer principle.

PSP was introduced in 1994 and has been put to practice with various projects in several companies including Motorola and Union Switch, and Signal.

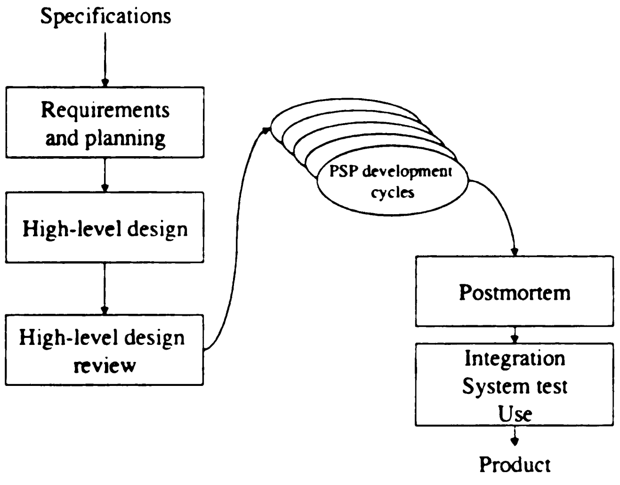


Figure 6. PSP flow diagram.

3.4.4. Domain Specific Software Architecture

Another approach to software development and reuse was developed as a result of a five-year research program sponsored by DARPA (Mettala 1992). It is called Domain Specific Software Architecture (DSSA) and is based upon the observations that

1. Distinct software applications can have common architectures.
2. Such common architectures can enable efficient reuse of components across such applications.
3. Such common architectures are easily recognized in the specific application domains, in part, because the body of widely understood concepts for a particular problem domain helps to overcome substantial differences in the representation of the applications.

There v

program clea

architectures

through re-use

specifications.

I would i

DICAM for Vehi

Erman et al. 1

methodology an

design, and va

combined techn

three principle w

1. Developing a

from real tin

concepts of k

2. The applicati

directly in the

domain analy

blackboard m

design, cons

[1] Avionics Naviga
Distributed Intellig
Management, Inte
Prototyping Techn

There were six independent projects in the program[9]. The results of this program clearly shows that the identification of general trends in software architectures of a particular domain significantly increases the productivity through re-use of software, documentation, and formalization of the initial specifications.

I would like to draw attention to one of the projects in the DSSA program – DICAM for Vehicle Management performed by the Tecknowledge (Hayes-Roth, Erman et al. 1992). To achieve maximum results in the development of a methodology and implementation of a suite of supporting tools for specification, design, and validation of DICAM applications researchers of Tecknowledge combined techniques of knowledge engineering and software engineering in three principle ways:

1. Developing a hybrid control technology that combines important concepts from real time software engineering with the knowledge-based reasoning concepts of knowledge engineering.
2. The application of concepts, methods, and tools from knowledge engineering directly in the software development process. (Knowledge - based models for domain analysis, classification for taxonomies, abstraction and specialization, blackboard methods of incremental problem-solving for system development design, constraint specification and processing in software requirements

[9] Avionics Navigation, Guidance and Flight Director, Command and Control, Distributed Intelligent Control and Management (DICAM) for Vehicle Management, Intelligent Guidance, Navigation and Control, Hybrid Control, and Prototyping Technology

manage

devel'op

3. The ap

(e.g.g.,

hierarc

softwar

Acc

80-90% of

Table 3 pr

be produce

Requirement requirements
Design: auto design speci simulation pr
Implementat code, automa implementat
Testing: code genera avoid re-test
Document reuse docum
Total

Tab

3.5.

The

computer s

management, and knowledge-based expert systems for providing software development and design assistance.)

3. The applications of software engineering methods to knowledge engineering (e.g.g., real-time systems development, database centered design, hierarchical systems, distributed systems, reference architectures, multi-tool software engineering etc.).

According to (Honeywell 1999) automation and reuse can be applied to 80-90% of life cycle activities and can reduce effort by a factor of 2 to 10. The Table 3 provides examples of estimates for various life cycle activities that might be produced by particular development groups.

	Life cycle activity and changes	Percent Reduction
Requirements: reuse requirements, avoid iterations on requirements	50% of 20%	80% of 20%
Design: automate analytic model generation, reuse design specifications, avoid coding simulation/prototype, avoid redesigns	90% of 20%	95% of 20%
Implementation: automate code generation, reuse code, automate software/system integration, avoid re-implementations	95% of 20%	95% of 20%
Testing: reuse test specifications, automate test code generation, automate trace/coverage analysis, avoid re-testing unmodified units.	90% of 20%	95% of 20%
Documentation: automate document generation, reuse document boilerplate	50% of 20%	80% of 20%
Total	75% of 100% (4X reduction)	90% of 100% (10X reduction)

Table 3. Impact of applying DSSA methods to DICAM

3.5. Function-Based Reasoning

The core idea of the described approach is based on associating a computer system with the engineered device and applying methods of function-based reaso

programs. T

reasoning and

3.5.1. D

The not

When a desig

required *functio*

reasoning helps

of the device ca

The work

and Chandrase

causal reasoning

research resulte

function and re

methodology is

device is modele

with clearly defin

devices is targete

The core

understanding of

explanation of a

decomposition of

device decompos

based reasoning and modeling developed for such devices for computer programs. This section overviews the theory and practice of function-based reasoning and presents current state in this field.

3.5.1. Description of Function-based Reasoning

The notion of function is in many ways central for practical engineering. When a designer designs a device, he/she is concerned with the delivery of a required *function*. When a system does not perform its *function,* diagnostic reasoning helps identify the reason for the *malfunction*. Predicting the behavior of the device can include reasoning about its *functionality*.

The work of Sembugamoorthy and Chandrasekaran (Sembugamoorthy and Chandrasekaran 1986) was rooted in research in qualitative physics and causal reasoning (de Kleer 1977; Hayes 1979; Kuipers 1986; Forbus 1988). This research resulted in the theoretical framework for representation of a device's function and reasoning about a device in terms of its functions. This methodology is called Function-based Reasoning (FR). In this framework, a device is modeled through its decomposition into constituent sub-devices, each with clearly defined function(s)/role(s). The combination of functions of all sub-devices is targeted to achieve the stated function of the overall device.

The core idea of the approach is to enable capturing the causal understanding of the device in modular packets where each packet is an explanation of a purpose/role of the device. A device hierarchy captures the decomposition of the device into interacting subsystems. The layers in the device decomposition are linked by annotations in the causality packets to roles of lower leve

for which such

That is.

device at the

functions of the

values (or cha

its function, and

then points to a

been in the form

device goes in

pointers to the fu

device→ f

sub-

that explain

There are

research. Below i

these terms, which

1. A **device** is a

device. A devi

outside word. I

it handles (e.g.

of lower level sub-devices. This plays a central role to any computation purpose for which such a functional representation is used.

That is, when one "reads" a functional representation, one starts with a device at the highest level in the device representation. One is then shown the functions of the device. These functions are abstract statements of start and end values (or changes to values) of state variables obtained when the device meets its function, and the terms under which the function is applicable. Each function then points to a causal fragment (a behavior of the device) which has typically been in the form of a state variable description of the changes through which the device goes in order to achieve its function. This device is annotated with pointers to the functions of lower level sub-devices:

device➔ function ➔ behavior ➔

sub-device ➔ function ➔ behavior

➔ . . .

that explains the device functionality from top to bottom.

There are several major notions and terms often-used in FR related research. Below is the presentation of a compilation of definitions of a subset of these terms, which will be used throughout this paper.

1. A **device** is a decomposable entity. Each component of a device is also a device. A device has associated with its **loci** or **ports** of interaction with the outside word. In many particular cases a port is associated with a **substance** it handles (e.g. fluid, gas, electrical current, or data of a certain kind).

2. **Connecti**

designate

3. A partial c

device. A s

4. Types of f

suggested t

Control to M

FR methodo

1998).

5. A function c

devices or th

6. A behavior i

process des

device funct

7. Every devic

Functior

Precond

Postcon

By <c

condition

Tradition

the function it p

starts, postcond

2. **Connections** between devices use ports as points of interface and carry a designated substance.
3. A partial description of a device or its environment is called **state** of the device. A state is defined through the collection of **state variables.**
4. Types of **functions** of a device are distinguished according to the set suggested by (Keuneke 1989) Control, to Prevent, Control to Make, and Control to Maintain. This set is being augmented with new applications of the FR methodology (Lukibanov, Martinez et al. 1998; Martinez, Lukibanov et al. 1998).
5. A function of a device is achieved by the **behaviors** of its constituent sub-devices or the laws of the domain of discourse.
6. A behavior is a description of state changes in the device and is called causal process description (**macroexpansion**) and it captures causal story of the device functionality.
7. Every device in the FR representation is shown through the quartet of slots:

   Function <ToMake I ToMaintain I . . . > of device <device>

   Precondition <precondition(s) on state variable values>

   Postcondition <change(s) in state variable values after function>

   By <causal fragment which produces desired post-condition>

Traditionally, the device's functionality is represented through the name of the function it performs, preconditions that should be satisfied before the device starts, postcondition expression which becomes TRUE after the device finished its operation.

device, whic

subordinate c

The no

reasoning para

being carried t

device. The L

changed, modif

perform these

path of the slug

Function

causal story of

FR that has be

- FR method

device in or

- Functional

question, a

through the

- FR supports

there is r

decomposit

[1] The "world k

of discourse

its operation, and By clause. By clause is the pointer to the behavior of the device, which achieves the device function through the functions of the subordinate devices or by means of world[10] knowledge and definitions.

The notion of *slug* of matter is in many ways central for the functional-reasoning paradigm. The slug is an indivisible fragment of a substance that is being carried through the causal network that explains the functionality of the device. The Laws of Conservation dictate that the slug of matter cannot be changed, modified, or destroyed without applying specific operators that explicitly perform these operations on it. One "reads" the causal network by tracing the path of the slug of substance through the functional representation of a device.

Functional representation of a device along with the ability to understand causal story of the device's function brings to light several important qualities of FR that has been leveraged in numerous FR applications:

- FR methodology supports functional and structural representation of the device in one model, where one is naturally related to another.
- Functional representation supports reasoning which answers a 'What if?' question, and allows tracing consequences of an arbitrary action, manifested through the changes in state variables.
- FR supports variable granularities of knowledge in the same model. That is, there is no bottom level granule for device decomposition. The decomposition can go on until infinity or until the developer decides that a

[10] The "world knowledge" term here refers to the knowledge about the universe of discourse

particular

can be lev

- The ability

chain from

required by

FR meth

describe examp

3.5.2. FR

Pegah's

Bond and Pega

an example of

devices for sim

found in (Kamel

is an automatic

Figure 7. In or

authors extends

calculate and i

model.

Hawkins

as a tool for the

approach for dia

station Freedom

outer space and

particular level is sufficient for the description of the device's behavior. This can be leveraged in another attractive quality of the FR approach:

- The ability to explain achieved results by backtracking through the causal chain from the state that is needed to be explained at the level of explanation required by the user or a level maximally achievable by a system.

FR methodology is flexible enough to be adapted to various areas. Next, describe examples of application of FR techniques in different domains are:.

3.5.2. FR for Devices

Pegah's and Bond's (Pegah, Bond et al. 1992; Bond and Pegah 1993; Bond and Pegan 1993) FR model for jet fighter FA-18 fuel system can serve as an example of the application of the FR methodology to modeling physical devices for simulation purposes. Another example of such modeling can be found in (Kamel, Sticklen et al. 1989; Sticklen, Kamel et al. 1991). The test bed is an automatic cruise control system, which is schematically represented in Figure 7. In order to model the device for numerical and qualitative simulation authors extended FR approach with the addition of a new type of function *to calculate* and integrate qualitative and quantitative reasoning in a functional model.

Hawkins (Hawkins, McDowell et al. 1993) has shown that FR can be used as a tool for the troubleshooting devices. The authors used an augmented FR approach for diagnosis in the External Active Thermal Control System for space station Freedom which transfers accumulated heat from inside the station to the outer space and maintains climate inside the station. The core idea of the

approach is t
device and,
necessary dia
example of us
(Price 1996), w
a car is used to

Figure 7.

(Modarres
the functional vi
thousand functio
combined in a fu
that of the mai
issues: the scala

approach is that the inherent properties of FR enable causal reasoning about the device and, therefore, once the FR model of the device is complete, the necessary diagnostic information could be automatically generated. Another example of using the FR approach to troubleshooting devices was described in (Price 1996), where the author showed how FR model of the electrical system of a car is used to identify sneak electric paths.

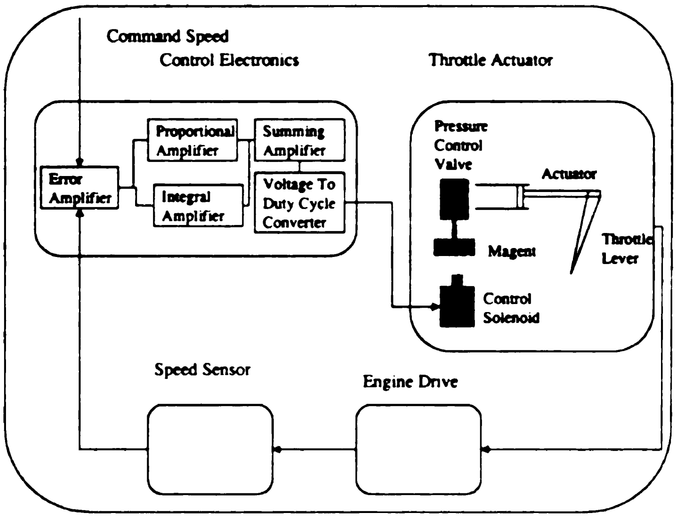


Figure 7. Schematics of automatic cruise control

(Modarres 1998) approached the problem of modeling nuclear plants from the functional viewpoint. The model of a plant consists of several hundred thousand functional units, which perform main and support functions that are combined in a functional model. While his approach is somewhat different from that of the mainstream FR community, it demonstrates important pragmatic issues: the scalability of functional approach, use of libraries of devices, and fun

app

beh

cov

199

a de

the c

the c

in (Lo

and (

librari

rudim

comm

approa

1987; S

1995).

3

I

that pro

process

functional decomposition of a device through the ‘goal-tree success-tree approach’.

The FR approach was traditionally focused on modeling dynamic behaviors of devices. A theoretical augmentation allowing FR methodology to cover the modeling of static devices was proposed in (Lukibanov, Martinez et al. 1998). The authors demonstrated how knowledge of the mechanical structure of a device coupled with the available functional information could not only capture the design intent, but also aid in the re-design and in the reverse engineering of the device.

Another attempt to use FR techniques for design purposes was presented in (Lossak, Yoshioka et al. 1998). The paper reported joint efforts of Japanese and German researchers, which resulted in a framework that allowed the use of libraries of devices for design. Devices are represented functionally with rudimentary geometrical data that allowed further manual detailing when design commitments are made.

All previous examples dealt with the engineered entities. However, the FR approach was successfully applied to modeling biological systems (Sticklen 1987; Sticklen and Tufankji 1992) and medical diagnosis (Tsumoto and Tanake 1995).

3.5.3. FR for Processes

It was noted (Chandrasekaran 1993; Chandrasekaran and Kaindl 1996) that process modeling is an open problem for the FR approach. However, processes (engineered or biological) that deal with the transformation and/or transport of s

example, La

approach app

extension of t

al. 1998). T

parameters of

describe how a

not only determ

the manufactura

The main

al., models the

On the other ha

itself is the subje

3.5.4. FR

A compu

functionality and

that could be und

Computer prog

subprograms (fo

oriented program

device from the

(Hartman and

decomposition to

transport of some kind of substance could benefit from the FR approach. For example, Lambert in (Lambert, Riera et al. 1997) presented a functional approach applied to modeling nuclear reprocessing. Another application and extension of the FR methodology was demonstrated in (Martinez, Lukibanov et al. 1998). The center of the authors' attention is the dependencies between parameters of a manufacturing process and manufactured artifact. The authors describe how a functional model of a manufacturing process can be leveraged to not only determine the parameters of a manufactured artifact, but also estimate the manufacturability of the design.

The main difference between these two approaches is that Lambert *.et. al..* models the process through modeling the objects that perform this process. On the other hand, in the approach proposed by Martinez *et. al..* the process itself is the subject of modeling through FR techniques.

3.5.4. FR for Software Design and Understanding

A computer program is a special kind of device with well-established functionality and interfaces. Software systems have strong flavor of causality that could be understood by following the input data through the software system. Computer programs in general have well-established decomposition into subprograms (for procedural or functional programming) or objects (for object-oriented programming). These properties allow treating a software product as a device from the FR point of view. (Allemang and Chandrasekaran 1991) (Hartman and Chandrasekaran 1995) applied the ideas of functional decomposition to computer programs. The goal of the above research is to aid in

automatic pr
applied the re
functional arc
aware of its re
for agents ca
agents operate
approach using
Sorenson 199
designing and
properties of F
implementation
applying the FR
of ZD system (
based software
describe the fun
during system in
the software ar
ultimately, captur

3.5.5. Sta

The curren
two major points
time into functiona
issues:

automatic program understanding and debugging. (Murdoc and Goel 1998) applied the results of these studies and augmented them in research towards a functional architecture of reflective agents. A reflective agent is an agent that is aware of its reasoning. The authors claim that leveraging the FR representation for agents can greatly improve performance of the system within which these agents operate. The authors also demonstrate the performance of the proposed approach using meeting scheduling software as a test bed. Stroulia (Stroulia and Sorenson 1998) demonstrated the applicability of the FR methodology to designing and re-designing software. The authors leverage the diagnostic properties of FR models to find "bottlenecks" in software systems, where the implementation does not meet the design specifications. Another example of applying the FR methodology to software design can be found in the description of ZD system (Liver and Allemang 1995). The authors advocate component-based software engineering. They show how the FR approach is used to describe the functionality of software components and make binding decisions during system integration. The FR approach also gives the possibility to describe the software architecture, automatically generate a Data Flow Diagram, and ultimately, capture the design rationale.

3.5.5. State of the Art

The current state of the art research on functional reasoning is focused on two major points – automatic generation of functional models and incorporating time into functional models. The first point is directly associated with following issues:

- Model completeness, which is a twofold problem itself:
  - The causal model must be "true", that is every "rule" in the model must be correct with respect to the variables of the model.
  - At every possible state, the model has to have enough information to generate the next set of possible states, i.e. the model should be self-sufficient.
- Libraries of devices. This issue has been addressed from both theoretical and practical points of view in many researches and applications. However, there are still problems remaining. These problems are concerned with library organization, device instantiation, property inheritance in the hierarchical libraries, etc.
- Model description. This is the problem of 'faceplates', i.e. how to describe the device with the "right" amount of information for the model assembler to make an educated decision on using a particular device from the library in the assembled model.
- Incorporating explicit time into the functional representation is an open problem. Yet, resolving this problem is very important for many different theoretical and practical purposes, and is essential for the practical applicability of FR methodology for great many different real life problems. Such as combining different time sensitive devices (e.g. a model of the cruise control and a model of climate control) with different time scales into a bigger model (e.g. a model of a car).

H

re

s

fa

d

p

2

3

4

Problems of designing libraries of device were addressed in (Pegah, Hawkins et al. 1994) and now are being revisited and extended in ongoing research at Intelligent Systems Laboratory at Michigan State University. In the same research, Hawkins addresses the problem of incorporating time into the functional models and associated sub-problems.

There are a number of problems directly or indirectly associated with the described venues of the research. Some of them will be addressed in the proposed research.

3.6. Conclusion

I would like to re-iterate the important theoretical and practical sources that prompted and contributed to this dissertation research.

1 Introduction of reusable problem-solving methods and task-specific architectures in the KBS research allows concentrating on the decomposition of a problem into the set of manageable units, where each unit has predefined functionality and inference strategy.

2 The Generic Task branch of TSA and the legacy software package (GT ITS) proved to be an effective tool for building practical small-scale KBSs.

3 Advances in integration methodologies and inter-object communication techniques allowed identification of generic control architectures and methods that allow integration of distributed units into a functional system.

4 Ontological research enabled the development of domain ontologies that are used to share knowledge between separate parties.

a

p

6

pe

Fl

th

gr

Th

methodol

systems.

5 Universal Modeling Language and component-based software engineering take a *black box* point of view at the software modules, which allows concentrating on the modules' functionality and treating the problem of integration of this modules separately.

6 The Function-Based Reasoning paradigm provided a functional perspective on any engineered or biological device. The extension of the FR paradigm on software systems enables their functional decomposition thus allowing concentrating on the function of every participating module, gradual problem decomposition, and progressive knowledge acquisition.

The next section will show how the listed above theories are used to build methodology that supports the development of integrated knowledge-based systems.

4.1.

To

create a

i.e. solve

this mea

experts' t

domain k

solved du

building a

the task

problem-s

be divided

1. Assess

   problem

2. Organiz

   problem

3. Acquire

4. Determi

   implicitly

   designat

   passed b

4. ANALYSIS AND A THEORY BEHIND APPROACH

4.1. Introduction

To build a knowledge-based system that solves a domain problem is to create a software system that will model the functionality of the domain expert, i.e. solves the tasks usually performed by domain expert. More often than not this means that an internal I-KBS architecture should copy (at least partially) experts' thought structure. The process of eliciting this structure and underlying domain knowledge – knowledge acquisition – is the core problem that has to be solved during the construction of a knowledge-based system. In the case when building a knowledge system involves reuse of existing problem-solving modules, the task of system building expands to include issues of integrating disparate problem-solving modules. Generally, the whole system building process could be divided into five major stages:

1. Assess the problem at hand and decompose it into manageable sub-problems.
2. Organize them in problem - sub-problem structure (so the solution of sub-problems leads to the solution of a larger problem)
3. Acquire problem-solving units responsible for solution of sub-problems
4. Determine information flow between problem-solving units by explicitly or implicitly connecting problem-solvers' inputs and outputs as well as designating a working vocabulary and structure of messages that are being passed between units

5. Define necessary control that would manage activation of each participating problem-solving unit.

The next Section lists issues that are not covered in this research and its reasons. The rest of this Chapter describes how the approach developed as a part of the research effort and covers the described above stages.

4.2. What is Not Covered by the Approach?

This dissertation intentionally omits problems of developing domain ontology. A domain ontology serves, as a backbone for building Integrated Knowledge Based Systems by providing working vocabulary that is syntactically and semantically understood by all participating units. The reason for leaving out such an important issue is that recent progress in ontological research resulted in the number of methods and tools for generating robust domain ontologies (Section 3.3). In addition, eXtensible Markup Language (XML) became *de facto* industry standard for developing domain vocabularies that in many cases play a role of ontologies by allowing store and reuse domain factual knowledge in flexible pre-defined structures. The application side of the dissertation (redesign of Socharis) is based in part on the ontology of manufacturing with composite materials that was developed as a part of the Socharis project.

Another major KBS problem that is excluded from the scope of the research: is the problem of building individual problem-solvers (a substantial part of I-KBS building process). The reason for this is that research in Task-Specific Architectures (TSA) already produced a well-populated group of problem-solving methods that could be successfully applied to solve isolated problems of ordinary complexity. However, the practical issues of integration of multiple problem solvers in to a working system were not in the focus of the research community. This research instead, centers on methodology that uses these problem-solving methods to develop large integrated KBSs. In fact, this dissertation is built on the foundation of one of schools of TSA – Generic Task approach – which supplies the unitary problem-solvers for building I-KBS in the described framework.

4.3. Theoretical Premise

Sticklen's *Knowledge-Level Architecture Hypothesis*, Steels's *Knowledge Use Level*, and Van de Welde's *Modeling Libraries* have one thing in common: all three of them consider the structural organization of problem-solving units (methods) as a crucial step in designing, describing, or modeling an intelligent system. The main thought behind these theories is that given the functionality of each agent and all the connections among the agents in the system it will be possible to understand the behavior of the system, and its sub-parts. While intuitively accurate, neither of these theories suggests how to organize agents, how to provide communication channels, and how to control execution order. , except for Van de Velde in (Velde 1994) where he suggested using any of the available methodologies (CommonKADS, Vital, etc.) to do so. However, he stopped short of describing a mechanism that could lead to the realization of the *Modeling Libraries*. This dissertation is based on the premise that Function-Based Reasoning/Function-Based Modeling is able to model such structures in all their richness.

The Generic Task methodology serves as another important theoretical cornerstone of the described approach. Generic tasks proved to be a convenient instrument in knowledge acquisition along with development and deployment of knowledge-based systems. However the brittleness of GT templates caused the developers of KBSs programmatically alter the pre-defined inference engine and/or knowledge representation to adapt them to the problem at hand. Extending generic tasks, this dissertation capitalizes on the large-grain view of the problem-solving process at the same time allowing modifying control among the cooperating entities without re-programming problem-solving templates. In a sense, this approach enforces guidelines for designing an I-KBS using the set of pre-defined templates. The basic principle is to divide the problem into a number of smaller more manageable parts that could be solved by an application of a single template without altering it. Then integrate the resulting knowledge-based systems into a functioning I-KBS by defining control and information flow between them.

However, it is important to understand that the only quality of Generic Tasks that has been heavily leveraged here is the possibility to apply the *black-box* point of view on any GT-based problem-solvers. This is because of the fact that each GT-based unitary KBS is self-sufficient in the sense that it contains all knowledge necessary for problem-solving as well an inference module. The significant conclusion is that any self-contained problem-solver could be used in the extension of the described methodology as long as its functionality could be *"black boxed"*.

re
as
Al
(F
sys
bec
sof
an
mo
By
ge
RU
ab
kno
wh
does
for k

prob
struct
In ad
decom

4.4. FR as a Structure Modeling Methodology

Inherent properties of the FR methodology – that sees a device as a recursive hierarchy of sub-devices – allow considering a software system (such as an integrated knowledge-based system) as a hierarchy of sub-systems. Along this line, the FR modeling mirrors the Rational Unified Process (RUP) (Fowler and Scott 2000) that considers structural decomposition of the software system as one of the stages of software development. RUP recently has become an industry standard for development of large, well documented software systems. Having its roots in Capability Maturity Model (Section 3.4.3) and Object Modeling Technique (Section 3.4.1), RUP goes beyond structural modeling and defines documenting policy, use-case models, block diagrams, etc. By following the Rational Unified Process the designer ideally finishes by generating skeletal code for the software system being designed. One of the RUP's shortcomings in modeling knowledge-based systems comes from its ability of transforming the design intent into the code. Often, when dealing with knowledge-based systems the code is not important (inference engine is already written and being reused), but knowledge representation is not. However, RUP does not have facilities to operate with complex knowledge models often intrinsic for knowledge-based systems.

Meanwhile, the use of the Function-Based Modeling techniques for problem decomposition allows the I-KBS designer concentrating on knowledge structure and representation, rather than on low-level methods for data handling. In addition, the natural for FR ability to arbitrarily choose the level of decomposition and explicitly define functionality of every participating subsystem com

com

KB

•

•

•

S

S

D

a

comes particularly handy when modeling the system out of the readily available components.

4.5. Limitations of the Traditional FR Approach

In spite of the adequacy of the traditional FR methodology in modeling I-KBS' structure this theory suffers from several limitations:

- The lack of an adequate ontology capable of describing the functionality of problem-solving units.
- Inability to distinguish between different parts of the substance that is being passed through the functional model, i.e. treating it as an inseparable *slug* of matter.
- Weakness of the pre-condition clause that made it very difficult to state complex conditions on the activation of parts of the functional model, which adds to the absence of complex control structures within the functional model.

These drawbacks limit application of the FR techniques to modeling software systems to primitive ones. Systems with a simple information flow and straightforward functionality.

4.6. Extension of Function-Based Reasoning Methodology: Functional Ontology

The implemented in this dissertation extension to the function-based reasoning methodology deals with the issues outlined in previous section. The problem of insufficient ontology is managed thorough augmentation of functional ontology with new types of functionality and behavior (Section 5.2.1). The addition to the functional ontology in the presented research agrees with the use of GT-based problem-solvers as bottom-level modules. That is, the functional ontology is extended to describe functionality and behavior of specific types of problem-solving methods: hierarchical classifier and multiple routine designers. It is possible to carry on this extension further to include any other types of problem-solvers should they be chosen as a building block, therefore extending the usefulness of the methodology beyond the Generic Task framework.

The use of augmented functional ontology not only helps to accurately describe the functionality of every component of modeled I-KBS, but also directs the problem decomposition process by suggesting available problem-solving methods and, by doing that, implicitly guiding the I-KBS designer to dissect the problem into the chunks manageable by available problem-solving methods.

Another advantage of using such augmented functional ontology is as follows. According to the theory of task-specific architectures, every problem-solving method has a specific knowledge structure associated with it. Therefore, definition of a method automatically specifies an associate with this method knowledge representation. If we turn this argument around, than it could be said that one could identify an applicable problem-solving method by analyzing available knowledge and its representation. Consequently, a fixed set of problem-solving methods (an explicit enumeration in the functional ontology) enables matching of knowledge structures in domain of interest to available knowledge representation. This also could be used as an instrument in problem decomposition process.

The implementation of this extension to the FR methodology is described in detail in Section 5.2.1 of this dissertation.

4.7. Extension of Function-Based Reasoning Methodology: Information Processing

Conventionally, a FR model deals with a *slug* of substance (Section 3.5) that travels through a causal model. This slug of matter cannot appear, disappear, or transform (by changing in nature and magnitude) without application of specifically stated operations. While this is sufficient for the majority of engineered devices, a software system requires different approach. Information, that travels through the software system is not a monolithic *slug* but rather a collection of information streams each of which travels by a different (possibly independent) trajectory and could be created or destroyed without contradicting to any of conservation laws (which are of concern when dealing with physical devices).

This dissertation introduces *Port Managers* that address both of the listed above concerns. *Port Manager* is an auxiliary device associated with every component of functional decomposition. Its main role is to supervise input and output variables of the device, translate them into internal representation if needed, connect with other port managers, and control activation of the corresponding device depending on the current problem-solving context. Port managers envelop their respective devices, therefore enforcing the *black box* point of view. The only way to access a device is through the input and output variables that are published in the device's port manager.

Any part of the I-KBS could be connected (dynamically or statically) to any number of co-operating modules. Any sub-set of input or output variables can participate in such connections. To support this quality of software systems port

managers should be capable of performing multiple variable mappings, linking output variables of one device/problem-solver to input variables of another. Exercising multiple mapping mechanisms, the I-KBS developer can effectively model arbitrary information flow among the participating problem-solving modules. This feature could be related to the object-oriented programming techniques where the designer encapsulates data and methods and provides interfaces to access them from outside.

Another problem that was listed as a shortcoming of the traditional FR, is the weakness of the control structures – preconditions – a Boolean expression on model's state variables. Precondition is the only way to gain dynamic control over the execution: when precondition is satisfied the sub-device activates. While satisfactory for modeling engineered devices where the amount of modeled through variable substances small enough, this form of control seriously hinders adequate modeling of software systems.

Port managers can considerably strengthen FR techniques in this respect. Port managers could not only perform all types of precondition checking, but also add controls, specific for problem-solving types participating in an I-KBS construction. This could be justified by the fact that different problem-solvers produce results of different structure that may require specific attention (e.g. enumerating and filtering). These different kinds of results might in turn be used by other problem-solvers. It is the task of port managers to recognize these structures and act correspondingly so that the problem-solver would receive input parameters of structure that agrees with it interface specification.

Idea

to achieve

is compron

transform

In

adapter tr

different

so his or

different

4.8.

T

the trad

of the a

function

explicit

descript

the infor

methodo

to expre

as *defici*

A

designer

chunks tha

Ideally, a well-planned I-KBS should conform to a single domain ontology to achieve uniform *understanding* among participating entities. In case this rule is compromised, port managers should provide translating capability that would transform the outputs of one module to the input of another.

In short, a port manager could be compared with a plug adapter/transformer kit that a savvy traveler caries around the world to plug to different electrical and telephone outlets with different voltages and configuration, so his or her device keeps receiving energy and data in different locations from different sources without being concerned of device's malfunction.

4.8. Conclusion

The combination of the described in previous sections augmentations to the traditional Function-Based reasoning paradigm allows adequate description of the architecture of an Integrated Knowledge-Based System capitalizing on the functionality of the participating units, functional task-subtask decomposition, and explicit direction of the information flow. The major change that allows description of the I-KBS is the addition of the facility of *port managers* that handle the information and control flow. Without this change the application of the FR methodology falls short in competent modeling of I-KBSs because of its inability to express multiple flows of the substance through the functional model as well as deficiency of the control mechanism.

Augmentation of the functional ontology, on the other side, lets the I-KBS designer concentrate on the directed decomposition of the problem at hand into chunks that could be handled by the units whose functionality is expressed in the augme

permit

therefo

result

introdu

augme

augmented functional ontology. A flexible nature of this functional ontology permits its extension to include description of different problem-solving modules therefore adapting to new problems.

The next step is to present a consistent methodology that if followed, will result in the I-KBS that solves the problem of interest. The next chapter introduces such a methodology and describes each proposed theoretical augmentation from the practical, implementational standpoint.

5.1

deploy

the fol

1. *Rap*

arch

2. *Guid*

the c

the

solve

3. *Integ*

know

5.2.

Fo

three sta

models:

1. Task-

recurs

by unit

2. Functio

problem

5. PROBLEM STATEMENT AND APPROACH

5.1. Problem Statement

The goal of this research is to develop a framework for design and deployment of integrated knowledge-based systems. This research addresses the following problems:

1. *Rapid modeling of system architecture* for development of problem-solving architectures and on-the-fly modifications of existing systems.
2. *Guiding the task decomposition process* by assisting in the identification of the constituent sub-parts of the system, mapping the identified sub-tasks to the existing problem-solving methods, and re-using of existing problem-solvers.
3. *Integrating pre-built problem-solvers into the target system,* leveraging control knowledge and using the domain ontology for the inter-agent communication.

5.2. Approach

Following the proposed approach, building of an I-KBS is divided into three stages. Each stage corresponds to the construction of one of three I-KBS' models:

1. Task-subtask decomposition model or Problem-Solving Hierarchy (PSH): a recursive division of the problem into smaller problems that could be tackled by unitary problem solvers.
2. Functional Model: shows a function (or a role) of each part of the I-KBS in the problem-solving process.

3. Inf

is u

overall

sub-pr

Solving

decom

conven

all prob

of the ta

T

node of

particula

intermed

functions

macroex

different

functiona

the comp

Th

that pass

matter is i

to anothe

3. Information-Processing Model: shows how information (variables and control) is used and is passed between the different parts of the I-KBS.

PSH can be understood as a recursive, domain dependent division of an overall problem into sub-problems that bottoms at the level where every identified sub-problem can be solved by some *a priori* identified method. This Problem Solving Hierarchy does not reflect a problem-solving flow but rather is a static decomposition, similar to the device-sub-device decomposition of the conventional FR. PSH allows the user and the designer to immediately assess all problem-solving units of an I-KBS, to see problem-sub-problem decomposition of the task at hand, and to access other models of the I-KBS.

The Functional Model enables the designer to look at the function of each node of the PSH, where the function/role of every leaf of PSH is tied to a particular unitary problem solver or a formula. The function(s) of every intermediate node and the function(s) of the root of the PSH are composed of the functions of the other nodes of PSH. The Functional Model is used to generate a macroexpansion of the I-KBS: a network that shows a causal chain of firing different parts of PSH. This macroexpansion gives a high-level picture of the functionality of an I-KBS by imposing partial temporal order on the execution of the components of PSH.

The traditional FR is usually concentrated on a particular *slug of matter* that passes through the functional model. In the case of I-KBS, this *slug of matter* is information that is being passed from one node of the Functional Model to another. The Information-Processing Model is used to unambiguously deter

some

the ap

proble

charac

charac

softwar

F

follows:

P

Th

granulari

one from

On

the overa

and know

necessary

difficult to

description

determine the direction of each stream of the information flow as well as to define some control data used to manage the nodes of the Functional Model.

5.2.1. Integrated Knowledge-Based System as a Device: Functional Ontology

I use a metaphor of a device to describe a problem-solver. According to the approaches described in previous section, its input data, output data, and a problem-solving behavior (which achieves a function of a problem-solver) can characterize any problem-solver. This characterization mirrors the functional characterization of a device, thus enabling use of the FR methodology on software systems including problem solvers.

Following FR methodology an integrated KBS can be decomposed as follows:

Problem-solver ➔ function ➔ behavior➔

Sub-problem solver ➔ function ➔ behavior. .

...

The decomposition goes further on until the developer reaches the level of granularity, where a type of each leaf-level sub-problem solver is matched to the one from the pre-defined set.

One of the advantages of the FR approach is the possibility of estimating the overall behavior of the device, based on the behaviors of the sub-devices, and knowledge of connectivity of devices. To leverage this property of FR it is necessary to define bottom level behavior of every problem solver. However, it is difficult to describe precise behavior of an arbitrary problem-solver. Such description is bound to be very general and therefore, will not give good leverage durin

precis

defini

ontolo

proces

1. su

situ

2. by

ma

I

a sub-

Designe

on the t

T

shown c

during task decomposition. Focusing on specific types of problem-solvers allows precise defining of function and behavior for each of selected PS. These definitions are reflected in the proposed augmentation to the current functional ontology. The changed functional ontology will aid in the task decomposition process by:

1. supplying the guidelines for choosing a particular PS method for a given situation and,
2. by directing decomposition of a task into sub-tasks whose PS architecture matches to that of predefined problem-solving methods.

In the research, I focus on classes of problem-solvers that are covered by a sub-set of GT problem-solvers: Hierarchical Classifier, Multiple Routine Designer, and Structured Matcher. However, there are no theoretical limitations on the type, size, etc. of bottom level problem-solver.

The proposed functional ontology for the set of GT problem-solvers is shown on Table 4.

Probl Solvei Type
Hierar l Class
Routin Design
Structu Matchei
Algorith c PS
A Proble Solver

Ta

Problem Solver Type	**Function**	**State Variables**	**Pre-condition**	**Post-condition**
Hierarchical Classifier	To Classify To Select	A set of Symptom variables; Class Variable	All symptom variables are set Or specific set of symptom variables is set	Class Variable is set Or classification failed
Routine Designer	To Design To Set Parameters	A set of design requirements variables; A set of design parameters	Design requirements variables are set	All Design parameters are set Or Design Failed
Structured Matcher	To Establish	A set of characteristic variables; Set of hypothesis	All characteristic variables are set Or Specific set of characteristic variables is set	Matched hypothesis found Or Matching failed
Algorithmic PS	To Calculate	A set of input variables; A set of output variables	All input variables are set Or Specific set of input variables is set	Output variables are set
A Problem Solver	To Solve	A set of input variables; A set of output variables	All input variables are set Or Specific set of input variables is set	Output variables are set

Table 4. Function ontology for GT based problem solvers

modeli

implem

to cont

is descr

1. **Fun**

2. **Prec**

3. **Post**

4. **By:**

Th

variables

postcond

engineere

more com

Boolean p

methodolo

relationshi

problem-so

these issu

associated

functions o

5.2.2. Augmenting Precondition Clause

In order to seamlessly apply the FR methodology to designing and modeling I-KBSs it is necessary to overcome several theoretical and implementational shortcomings.

In the traditional FR approach, the set of preconditions is the only method to control a device. According to this methodology, every function of the device is described as follows:

1. **Function:** <ToMake | ToMaintain |. . . > of device <device>
2. **Precondition:** <precondition(s) on state variable values>
3. **Postcondition:** <change(s) in state variable values after function>
4. **By:** <causal fragment which produces desired postcondition>

The *precondition* generally takes the form of logical propositions on state variables. If the precondition is satisfied then the function is performed and the postcondition is set. In many cases, when FR is used to model a simple engineered device this control method is adequate and sufficient. However, for more complex devices with elaborate behavior(s) it is not enough to state simple Boolean preconditions. If we are to model moderate complexity KBS using FR methodology, we need to be able to express complex conditional and causal relationships between different parts of the system, conditional activation of problem-solvers, translating outputs of one module to another, etc. To tackle these issues I introduce the notion of port managers, an auxiliary mechanism associated with every device in the FR model, that would control its. Control functions of a port manager include, but not limited to: converting input to a devic

proce

solvin

etc.

solvin

flexib

distrib

develo

manag

execute

one mi

satisfac

port ma

again.

activatio

at the pa

Th

integrate

advanced

shortcom

cycles.

device from one ontology to another, performing elaborate problem-solving procedures on inputs (e.g. deciding on defaults according to a specific problem solving context), conditional transferring of control to parent or children devices, etc.

Where each module performs self-activation based on the current problem solving-context, the use of port managers will enable an adequate modeling of flexible and semi-restricted control. In fact, port managers will play the role of distributed control units associated with each device. In the cases, where the developer chooses to use a centralized control scheme, the function of port managers will be reduced to the traditional checking of Boolean preconditions.

In addition to performing operations on inputs, the port manager might execute some actions on the output of the device. An example situation where one might use the output port manager is when the result of the device is not satisfactory and it would be preferable to run the device again. In this case, the port manager would modify the input data for the device and run the device again. Another use of a port manager for output may be the conditional activation of subordinate devices, when the developer elects to center the control at the parent device instead of transferring it to the sub-devices.

The introduction of port managers will enable the modeling and designing integrated KBS with arbitrary architectures. Nevertheless, in order to provide advanced design and modeling capability it is necessary to overcome another shortcoming of the current FR implementation: its inability to perform internal cycles.

5
I

develop

tradition

is built t

This cre

a unitar

behavior

shows th

knowledg

Th

is usually

then com

functional

causal ne

net define

source ar

important

necessari

node dep

the port m

Fo

defines t

conditions

5.2.3. Tying it All Together

In the proposed framework, the construction of an I-KBS starts with developing a problem-solving hierarchy, which is a direct application of a traditional FR methodology augmented as described in Section 5.2.1. The PSH is built top-down by specifying a root of hierarchy first and then working down. This creates branches and sub-devices where the leaf nodes are associated with a unitary problem-solver, executed through "by knowledge" or "by definition" behavior, or simple formula. The result of this process is the hierarchy that shows the decomposition of the problem into subproblems as perceived by a knowledge engineer.

The next step is the development of a Functional Model of an I-KBS. This is usually done bottom-up by assigning the functionality to the leaf nodes and then composing the functionality of the intermediate nodes from already defined functional chunks. A macroexpansion of a Functional Model gives a view on the causal network of the interoperation of the parts of the I-KBS. In fact, the causal net defines partial execution order; where the nodes located farther from the source are being executed after those nodes located closer to the source. It is important to know that the execution order defined in macroexpansion does not necessarily express the real activation of the nodes, since the activation of the node depends also on preconditions and conditional operations defined through the port managers.

Following the development of the Functional Model the I-KBS designer defines the Information-Processing Model that determines the activation conditions for every node of the functional model as well as variable mappings between different parts of PSH. Described in Section 5.2.2 port managers perform these tasks by allowing the designer to access the variables of each port connect output variables of another port and arrange for their values to be passed to the internal variables of the node. Port managers enables the designer to create elaborate preconditions that not only include a series of predicates but also define collections of variables that have to be determined before the node is being activated. In addition, the port managers allow prescribing sets of default values to the variables to support standard reasoning as well as reasoning with incomplete data. On top of that, port managers provide a number of features that aid the designer in the process of developing I-KBS.

The detailed illustrated example of building I-KBSs using the developed shell is given in the Chapter 6. In that Chapter I discussed not only the particularities of implementation of the proposed methodology, but also give a tutorial on building an I-KBS according to the described approach.

5.3. Framework Restrictions

To correctly design an I-KBS using the described approach it is important to understand restrictions that it applies to the design process and the resulting architectures. The nature of the restrictions lies in the position that the described approach takes towards a unitary problem-solving unit. The unitary problem-solver is considered a *black box.* That is, a problem-solver is being looked at as an entity with known inputs, outputs, and functionality. There is not any other information known about a particular problem-solver. That means that an I-KBS designer manipulates problem solvers as LEGO® blocks without knowing their internal structure, knowledge, and etc. This point of view is close to object-oriented and component-based software engineering from the object or component handling perspective. Therefore, it is reasonable to expect that the benefits of these methodologies to be extended to the described framework. These benefits include ease of components' reuse, painless upgrade of components' functionality as long as it does not affect its input and output specifications, ease of modeling and developing. However, software-engineering practices do not necessarily cover all the possible approaches to the designing of an I-KBS.

Lets take for instance a generic problem that can be solved by problem-solver A, that in turn needs the result of problem- solver B to solve one of the intermediate steps. Three possible problem-solving architectures would perform a task at hand:

1. *Large Black Box*. To incorporate problem solver B into problem solver A therefore creating a bigger problem solver AB that performs functions of both problem solvers, A and B (Figure 8)

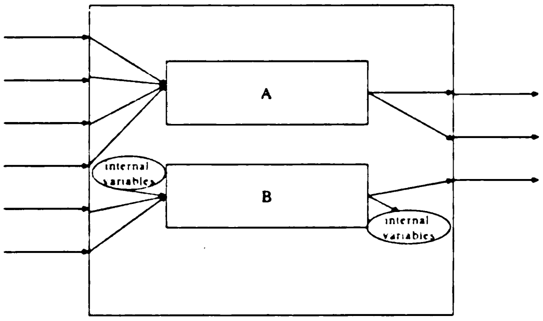


Figure 8. Large Black Box

2. *M*

p

g

w

3. *G*

va

of

F

2. *Multiple Black Boxes.* To divide problem solver A into a number of smaller problem-solvers {A11 ... A1i; A21 ... A2j}. Where problem solvers A11 ... A1i generate inputs for problem solver B and problem solvers A21 ... A2j work with the results of A11 ... A1i and B to produce final result (Figure 9).

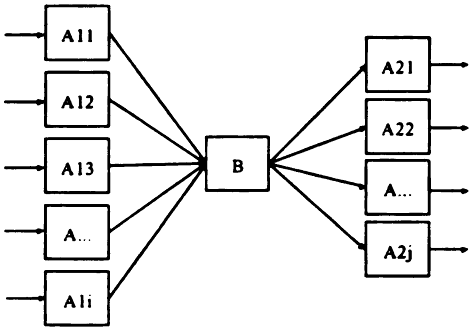


Figure 9. Multiple Black Boxes

3. *Gray Box.* Using this architecture the I-KBS designer can access the internal variables of the problem solver A and re-rout them to the inputs and outputs of problem solver B (Figure 10Figure).

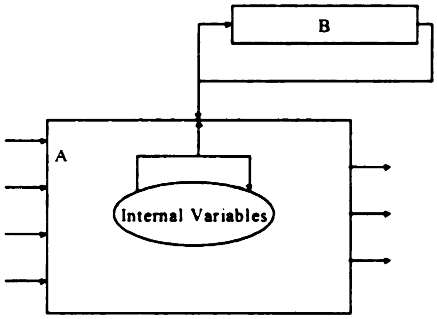


Figure 10. Gray Box

The first method, *Large Black Box*, however convenient, poses three major problems.

- First, this architecture makes it difficult to reuse problem solver B in another I-KBS. That is, the reuse of B can be effected by two ways: a) by re-implementing it using acquired knowledge or b) by calling the problem solver AB with the set of inputs that will allow run of problem solver B and then filter the results (if possible). Both methods require considerably more efforts than case of separate implementation of B.
- Secondly, knowledge acquisition for one complex problem solver often is more elaborate than knowledge acquisition for two separate problem solvers. The reason for this lies in the shift in knowledge engineering where instead of developing problem solver A and B separately and then integrating them, the designer needs to work on these three issues simultaneously. Consider for example a problem of diagnosing of a malfunction (of a Mars probe) where several sub-problems require determining of some value. Suppose that a PS for determining such a value exists and provides an answer in metric units. Then the designer needs to incorporate this PS into the structure of the diagnostic PS remembering not only to provide all necessary integration, but also to convert units from metric to SAE within the same problem solver.
- This leads to the third problem.
- Problem-solving architecture of AB problem solver is more complex then the respective architectures of problem solvers A and B separately, which may lead to difficulties with I-KBS's maintenance.

The second architecture, *Multiple Black Boxes*, addresses three above problems by chunking the problem solver into smaller black boxes with simpler internal architecture. Nevertheless, in turn, states another two.

- First, knowledge engineering for developing *Multiple Black Boxes* architecture for this problem might be "unnatural" for customary domain problem-solving practice. That is, the problem at hand might need a finer slicing (in knowledge and problem-solving methods chunks) then usual for the domain specialist. Lets look at the problem, introduced in the previous example and consider how it could be solved using *Multiple Black Boxes* architecture. First, every part of the diagnostic PS that uses the results of value setting PS becomes a separate problem solver and is being dealt with separately. Secondly, the part of diagnostic PS that prepares the data to be run by value setting PS also becomes a separate PS. Lastly, it is necessary to introduce an additional module that will convert units from metric to SAE.
- Secondly, the growth of the number of cooperating problem solvers in an I-KBS may lead to a more complex architecture, which may hinder the system's maintainability.

The *Gray Box* solution is very popular nowadays in multi agent architecture. However, its implementation requires knowledge of internal structure of problem solver A and changing a point of view at a unitary problem solver from *black box* to *gray box* with the possibility to access internal variables and manipulation with them. Yet, in many cases the insides of a unitary problem solver are not visible to an I-KBS designer, especially in the case when the system is being built by integrating readily available components. In addition to that, changing of the internal structure of a problem solver that is being used in several I-KBSs leads to an avalanche of changing code and/or knowledge in all co-operating problem solvers.

Constant trade-off between ease of implementation, ease of maintenance, and ease of re-use drives the I-KBS designer to choose the most appropriate integration structure. The described approach leans towards the *black box* point of view at the unitary problem solver, therefore excluding *Gray Box* structures from the list of alternative solutions to an integration problem. Consequently, implementational framework that supports the development of I-KBSs using the described approach drives the designer to use *Large Black Box* or *Multiple Black Boxes* solutions to the integration problems. Yet, it is possible to escape to the *Gray Box* architecture using legacy implementation of MRD and HC.

6. ILLUSTRATED APPROACH

To better understand the advocated approach it is helpful to walk through the process of building an I-KBS step-by-step using the developed Shell for Constructing Integrated Knowledge-Based Systems (S-Force). As an example, I will use the simplified problem statement for Socharis: to generate a family of applicable conceptual manufacturing plans from a conceptual description of a composite assembly. The knowledge-based kernel of this system deals with the problem of selecting, instantiating, and estimating a number of modern manufacturing technologies used to produce composite parts and consists of more than thirty unitary problem-solving modules. To make a walk through the system building process using S-Force readable I will limit the number of technologies that are considered for instantiation to Hand Lay-up and Resin Transfer Molding (RTM). In addition, I will reduce the number of estimation metrics to Part Turnaround Time and Tooling Cost metrics. Such limiting by no means lessens the control complexity of the I-KBS but instead it only reduces the number of problem-solvers that are being governed by the similar control and mapping structures. The description of full version of re-designed Socharis can be found in Section 6 and Appendix B.

Similarly to the original, the problem of Socharis* - an abridged Socharis - is divided into three major parts: the technology selection, the technology refinement, and the technology estimation. At the first stage Socharis* chooses the manufacturing process according to the description of the composite part:

shape, material, tolerances, etc. Then, if RTM and/or Hand Lay-up technologies are selected, Socharis* sets the manufacturing parameters suitable to manufacture the composite part (multiple results are likely). Finally, for each of the parameterized manufacturing technologies, Socharis* calculates normalized metrics for the part's turn around time and the tooling cost to help the designer to decide between multiple alternatives.

The number of GT-based problem-solvers was developed to solve different subtasks of the overall Socharis's problem. All separate problem-solvers were built using a consistent ontology of manufacturing with composite materials constructed as a part of the Socharis project. A list and short description of problem-solvers that are re-used in this example are shown in the Table 5

Name	**Type**	**Responsibilities**
OperationSelector	Hierarchical Classifier	select manufacturing technology
RTMRefiner	Multiple Routine Designer	Set parameters for the RTM process
Lay-upRefiner	Multiple Routine Designer	set parameters for the hand lay-up process
PartTT	Multiple Routine Designer	calculate metrics for the part turn around time
ToolingCost	Multiple Routine Designer	calculate metrics for the cost of tooling used to manufacture the part

Table 5. Problem-solvers, participating in Socharis *.

Building of an I-KBS using S-Force is done in stages, by constructing three models:

2

3

I-

e

co

so

of

du

it h

so

ch

act

de

1. Task-subtask decomposition model or problem-solving hierarchy that shows a recursive division of the problem into smaller problems that could be tackled by unitary problem-solvers.
2. Functional Model that shows a function (or a role) of each part of the I-KBS in the problem-solving process.
3. Information-Processing Model that shows how information (variables and control) is used and is passed between the different parts of the I-KBS.

In the following sections, I will describe the steps that the developer of the I-KBS takes to build these models and completes I-KBS in the S-Force environment.

6.1. Building the PS Hierarchy

The first step in building of an I-KBS using the developed shell is to construct a decomposition of the I-KBS into PS-subPS hierarchy: problem-solving hierarchy (PSH). This PSH is not a functional I-KBS, but rather a model of it that shows all available parts of the I-KBS organized in the meaningful clusters. These clusters represent task-subtask decomposition of the problem as it had been elicited from the field expert. Another way to look at PSH is that the solution of the problem defined for a node in PSH depends on the solution of its children (but may depend as well on the solution for other nodes). However, the actual order of subtasks' execution and the restrictions on variable passing are defined in other models: functional and information passing.

6.1.1. Socharis* Example

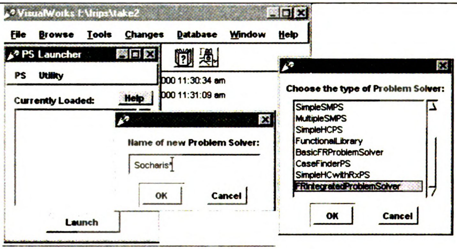
The problem-solving process for Socharis* could be divided into three parts: Technology Selector, Technology Refiner, and Estimator. Two latter also are divided into smaller parts: Lay-up Refiner and RTM Refiner, Tooling Cost Estimator and Part Turnaround Time Estimator respectively. This decomposition reflects problem-solving procedures usually performed by the manufacturing engineer while assessing alternatives for the part production:

1. Assess all possible spectrum of applicable technologies.
2. Consider possibility of altering different parameters. It is possible that one needs to consult different experts to set the correct parameters (in our case RTM and Hand Lay-up experts.)
3. Evaluate choices according to the set of pre-defined metrics. The set of metric has to be pre-defined and normalized in order to provide fair estimation of alternatives (e.g. Tooling Cost and Part Turnaround Time).

To start working in the S-Force environment, the I-KBS designer launches a correspondent VisualWorks image FRIPS. then clicks on the Γ button in the launcher toolbar to start Generic Task Integrated Toolset. Figure 11 shows the cascaded set of interfaces that guide the user through the process of creating new I-KBS:

1. Select `PS` → `New` from the menu bar
2. Enter the name of the new I-KBS
3. Select the type of problem-solver "`FRIntegratedProblemSolver`"
4. Click `Ok`

Following, the designer creates the root of the hierarchy and subsequently adds subs to it in the "FR → Device Hierarchy" editing window. Addition of a root is done by selecting "Relation → Add root" from the top menu followed by entering the name of the node in the correspondent dialog window (Figure 12). PSH can have multiple roots if this is required by the particularities of the problem at hand.

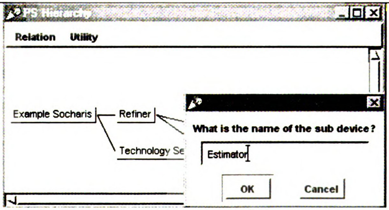


000 11:30:34 am
000 11:31:09 am

Name of new Problem Solver:
Socharis

Choose the type of Problem Solver:
SimpleSMPS
MultipleSMPS
SimpleHCPS
FunctionalLibrary
BasicFRProblemSolver
CaseFinderPS
SimpleHCwithRxPS
FRIntegratedProblemSolver

Figure 11. Cascaded set of interfaces guiding the developer through the process of creating new I-KBS



What is the name of the sub device?
Estimator

Figure 12. Adding a sub to the node of PSH

Attaching subs to a parent node is done by action-clicking[11] on the node where the sub is to be added, selecting `Create Sub Device` from the pop-up menu and entering the desired name (Figure 12Figure).

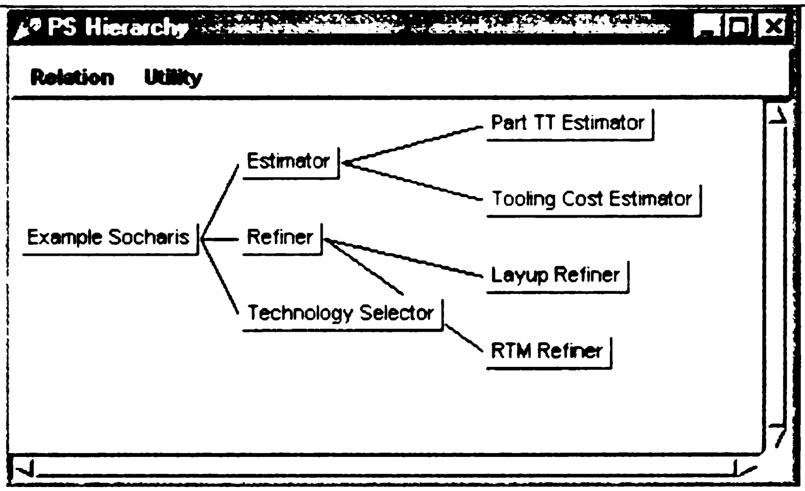


Figure 13. Problem-Solving Hierarchy of Socharis *

Figure 13 shows complete PSH of Socharis* that mirrors task decomposition described earlier in this section. However, it is impossible to make any conclusion about run time order of activation of PSH nodes. This order is determined through functional and information passing models

6.2. Functional Model: Assigning Functionality to the Nodes of PS Hierarchy

The next step in building an I-KBS is to define the Functional Model for its every part. Traditionally the list of behaviors' types consisted of "by Knowledge", "by Definition", "by Function of Device", and "by Behavior of Device". I augmented this list with the "by Problem-solver" type that allows associating a GT-Based problem-solver with a particular node of PSH. This behavior type

[11] Action-click is middle button on 3-button mouse or Ctrl + right button in Win X environment, or command Click in Macintosh environment.

enriches the current implementation of FR modeling framework by allowing complex problem solving procedures being executed at the unitary device level instead of simple one-step operations. The distinction between types of behaviors is explained in Table 6.

Type	**Description**
By Knowledge	Unitary behavior. Is based on the common to particular domain knowledge.
By Definition	Unitary behavior. Is based on the domain definition of particular device.
By Function of Device	Complex behavior. Calls other device's function
By Behavior of Device	Complex behavior. Calls particular behavior of another device.
By Problem-solver	Unitary behavior. Calls an associated with the node GT-based problem-solver

Table 6. Description of types of behaviors employed in Functional Model of I-KBS

To facilitate I-KBS building and maintaining, every node of PSH is associated with a variable – <node name> Number (stage counter) – which serves as a counter of problem-solving stages. This helps in supporting necessary bookkeeping as well as marks the meaningful milestones. As a rule, before the node is activated for the first time during problem-solving process, the value of stage counter is set to zero (unless otherwise specified by the system developer or the user). Each step in problem-solving process (or state change in traditional FR) is being marked by incrementing stage counter value. For example, if a device `A` has a function `fnA` with a behavior `bhA` of the type "by

Problem-solver: psA", then the state change that occurs in the bhA correspondent to the function fnA of the device A is as follows:

A Number = 0 → by ProblemSolver psA → Set A Number to: 1

This information can be used to check if a particular node had been invoked during the problem-solving process and to check the status of the problem solving in case of multiple state-changes in a behavior of a node.

On practice, the assignment of functionality is the bottom-up process. The developer associates the unitary functions with the lower level of PSH and then, builds the functional definitions of the upper nodes using the lower nodes functions and behaviors as building blocks.

In this implementation, I limited the types of invoked GT-Based problem-solvers to Hierarchical Classifier and Multiple Routine Designer to support legacy software. On the other hand, these two types of problem-solvers cover broad enough spectrum of classes of problem to be exclusively used in the development of Socharis. However, as was noted earlier, there is no theoretical limitation on the type of unitary problem-solver that could be used in the advocated approach.

6.2.1. Socharis* Example

To define the functionality of each of the leaves of PSH the developer performs the following sequence of actions:

1. Action-clicks on the correspondent node and chooses "Browse device" from the pop-up menu
2. Action-clicks on the upper pane and selects "Add function" and Enters the name of the function

3. Calls the behavior interface by double clicking on the added function name in the upper pane
4. Adds behavior similarly to adding function
5. Invokes behavior relation interface by double clicking on the name of the behavior
6. Finally, action-clicks on the Name Number node and chooses "`Add state change`" from the pop-up menu and then, selects the type of the state change link.

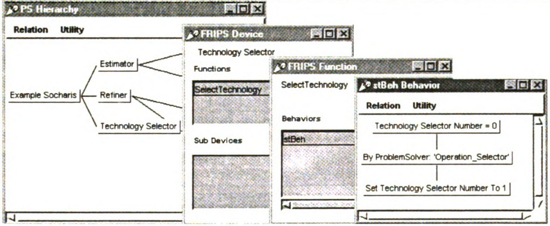


Figure 14. Sequence of interfaces leading to behavior definition

Figure shows the cascaded sequence of interfaces that result in the definition of the behavior for the Technology Selector. PS Hierarchy window shows PSH of Socharis*. `FRIPS Device` window provides interface to information about a device (in this case, to the device `Technology Selector`) its functions and sub-devices. By selecting browse function (or double clicking on a function name), the user invokes function interface: a window with title `FRIPS Function` that shows all constituent behaviors of the function. The last window – `stBeh Behavior` – was called from `FRIPS Function`. This interface represents behavior that leads to achieving a correspondent function.

Below are the descriptions of functionality for each node of PSH for Socharis*.

Device	Function	Behavior
Operation Selector	Select Technology	by problem-solver Operation_Selector_New. SimpleHCP Set operationSelector Number to 1
Lay-upRefiner	Refine Lay-up	by problem-solver: Lay-up. BasicRDProblemSolver Set Lay-upRefiner Number to 1
RTMRefiner	Refine RTM	by problem-solver: RTM. BasicRDProblemSolver Set RTMRefiner Number to 1
PartTT	Estimate PTT	by problem-solver: PTT. BasicRDProblemSolver Set PartTT Number to 1
ToolingCost	Estimate Tooling Cost	by problem-solver: ToolingCost. BasicRDProblemSolver Set ToolingCost Number to 1.

The functionality of upper level node is assembled from the functions of the lower level node as follows.

Device	Function	Behavior
Estimator	Estimate Technology	by function Estimate PTT of device PartTT and by function Estimate Tooling Cost of device Tooling Cost Set Estimator Number to 1
Refiner	Refine Technology	by function Refine Lay-up of device Lay-upRefiner and by function Refine RTM of device RTMRefiner Set Refiner Number to 1

It is easy to change the functionality of the device by editing the correspondent behavior. For example, to accommodate the particularities of problem-solving process in Socharis* it is necessary to perform estimation of the technological process right after the refining stage. Consequently, the updated functions of Lay-upRefiner and RTMRefiner are represented as follows:

Device	Function	Behavior
Lay-upRefiner	Refine Lay-up	By Problem-solver: Lay-up. BasicRDProblemSolver Set Lay-upRefiner Number to 1 by function Estimate Technology of device Estimator Set Lay-upRefiner Number to 2
RTMRefiner	Refine RTM	By Problem-solver: RTM. BasicRDProblemSolver Set RTMRefiner Number to 1 by function Estimate Technology of device Estimator Set RTMRefiner Number to 2

Finally, the description of the functionality of the root device is described through the functionality of its children nodes:

Device	Function	Behavior
Socharis Example	Socharis Function	by function Select Technology of device operation Set SocharisExample Number to 1. by function Refine Technology of Device Refiner Set SocharisExample Number to 2.

6.3. Functional Model: Macroexpansion

The functional description of each node does not give the full picture of the functionality of the overall I-KBS: to look at the full-blown Functional Model of the I-KBS it is necessary to generate its macroexpansion. Macro-expansion is the net (a directed graph with one or more sources and sinks) that represents causal (and therefore partial temporal) relationships between different parts of the PSH. There is an explicit succession in the execution of the nodes that are located on different levels of the network: nodes closer to the source are being executed before those farther from it. However, the execution order of the nodes that are located on the same level of the macroexpansion cannot be determined from this

W

c

P

kr

I-K

Mac

Fig

Mac

Figu

ase

ex

ws

I

an

n

se

a

es

ng

model. In addition, macroexpansion does contain neither information about variables passing between different parts of I-KBS, nor control knowledge about conditional nodes invocation, conditional looping, etc. The role of Information-Processing Model described in the next section is to explicitly define this kind of knowledge.

6.3.1. Socharis* Example

In order to assess the global causal story correspondent to the developed I-KBS the designer builds macroexpansion by selecting `"PS → Build Macroexpansion"` in the menu. The macroexpansion of I-KBS Socharis* (Figure 15) can be browsed by selecting `"Browse → Browse Macroexpansion"` from the top menu.

To better understand what macroexpansion represents, let us examine Figure 15 more closely. This macroexpansion is read top-down (or left-to-right in case the user flipped the macroexpansion using one of the available commands). An explanation to this macroexpansion of I-KBS Socharis* could be given as follows.

If Example Socharis Number equals 0 (i.e. the system has not been run yet) and Technology Selector Number equals 0 (Technology Selector has not been invoked before) then S-Force runs problem-solver Technology Selector which sets the Technology Selector Number to 1 (i.e. acknowledges that this node has been invoked once.) According to the behavior of the Example Socharis node, this finalizes the first stage of the problem solving process by setting the Example Socharis Number to 1. Then the systems checks if Refiner, Lay-up Refiner, and RTM Refiner have not been called yet and if this is true, invokes the problem-solvers Lay-up and RTM. Further, after testing the invocation of Estimator, Part TT Estimator, and Tooling Cost Estimator, the system runs problem-solvers Part_turnaround_time, and ToolingCost. Finally, after all nodes finished successfully the system sets correspondent <node name> Numbers: Part TT Estimator Number and Tooling Cost Estimator Number to 1, Estimator Number to 1, Lay-up Refiner and RTM Refiner Numbers to 2, Refiner Number to 1, and at last Example Socharis Number to 2, therefore finishing problem-solving process.

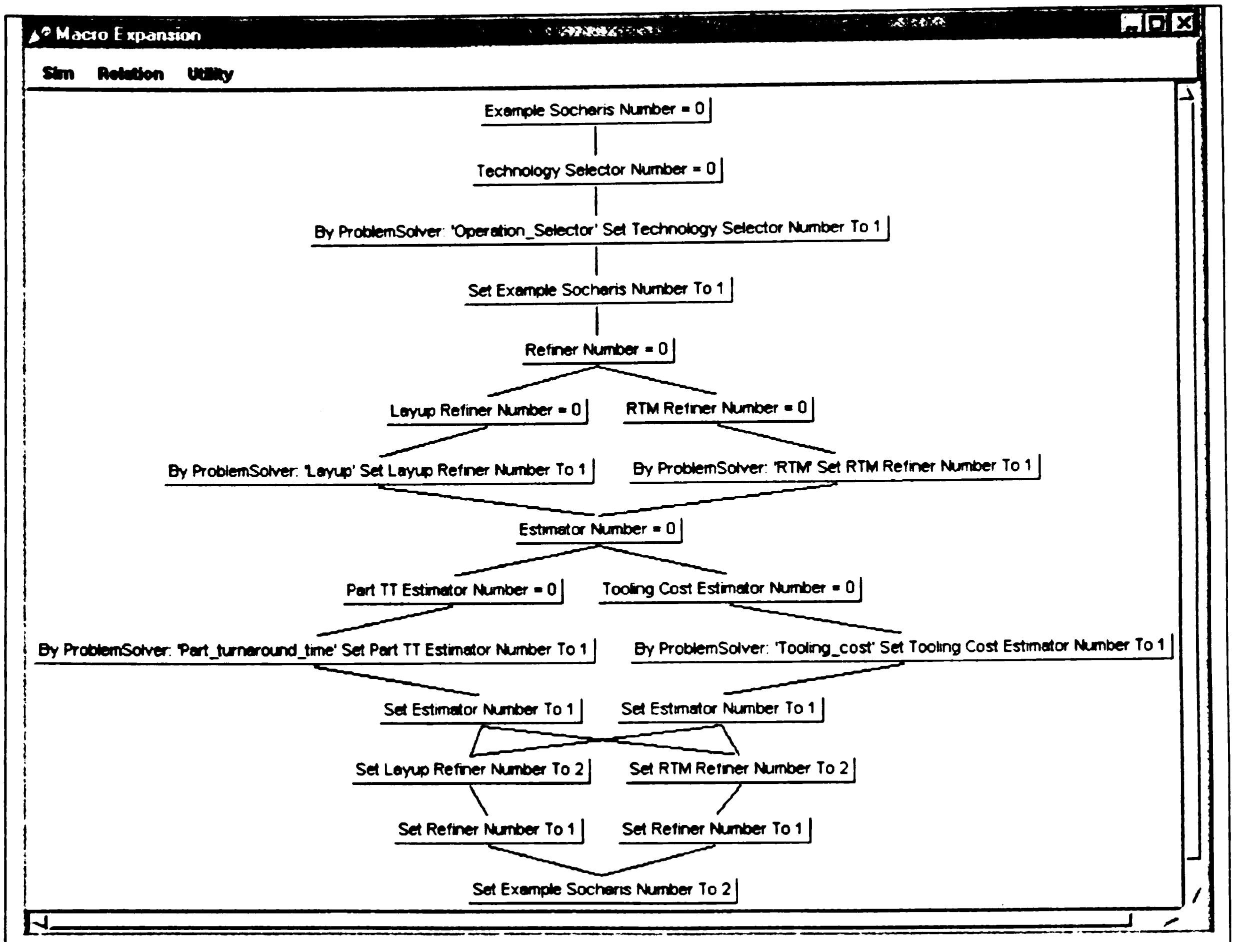

Macro Expansion
Sim
Relation
Utility
Example Socharis Number = 0
Technology Selector Number = 0
By ProblemSolver: 'Operation_Selector' Set Technology Selector Number To 1
Set Example Socharis Number To 1
Refiner Number = 0
Layup Refiner Number = 0
RTM Refiner Number = 0
By ProblemSolver: 'Layup' Set Layup Refiner Number To 1
By ProblemSolver: 'RTM' Set RTM Refiner Number To 1
Estimator Number = 0
Part TT Estimator Number = 0
Tooling Cost Estimator Number = 0
By ProblemSolver: 'Part_turnaround_time' Set Part TT Estimator Number To 1
By ProblemSolver: 'Tooling_cost' Set Tooling Cost Estimator Number To 1
Set Estimator Number To 1
Set Estimator Number To 1
Set Layup Refiner Number To 2
Set RTM Refiner Number To 2
Set Refiner Number To 1
Set Refiner Number To 1
Set Example Socharis Number To 2

Figure 15. Detailed macro-expansion of Socharis*

This is a detailed macroexpansion that shows both, pre- and post-assertions on <node name> Numbers. On practice however, it is more convenient to use condensed form of macroexpansion that displays only pre-assertions as shown on the Figure 16.

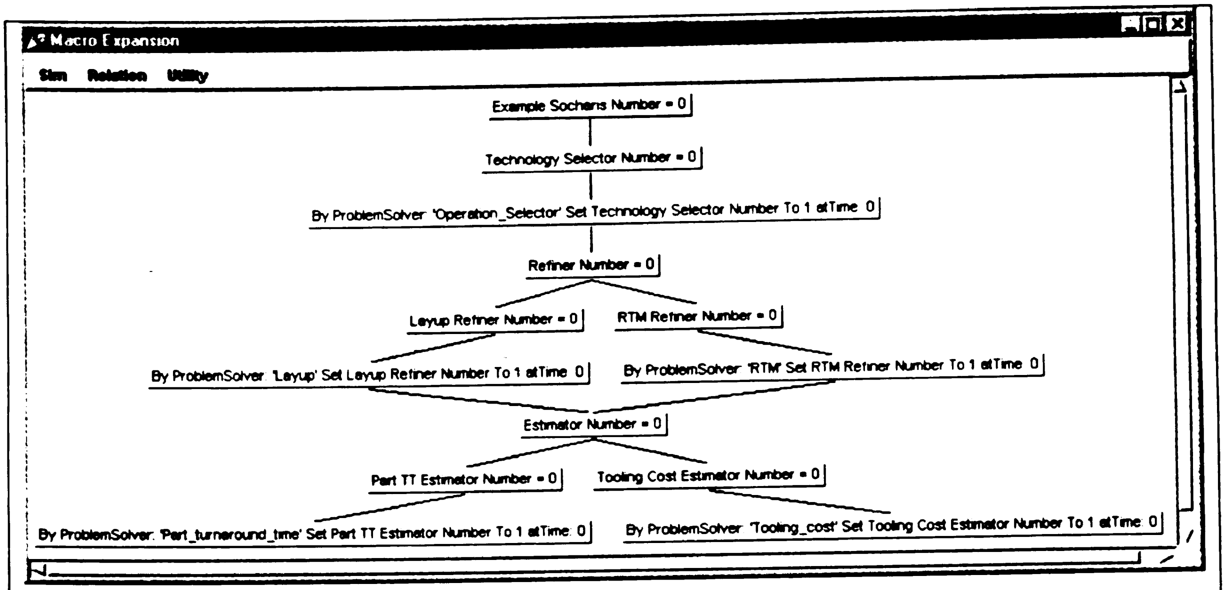

Macro Expansion
Sim
Relation
Utility
Example Socharis Number = 0
Technology Selector Number = 0
By ProblemSolver 'Operation_Selector' Set Technology Selector Number To 1 atTime 0
Refiner Number = 0
Layup Refiner Number = 0
RTM Refiner Number = 0
By ProblemSolver: 'Layup' Set Layup Refiner Number To 1 atTime 0
By ProblemSolver: 'RTM' Set RTM Refiner Number To 1 atTime 0
Estimator Number = 0
Part TT Estimator Number = 0
Tooling Cost Estimator Number = 0
By ProblemSolver: 'Part_turneround_time' Set Part TT Estimator Number To 1 atTime: 0
By ProblemSolver: 'Tooling_cost' Set Tooling Cost Estimator Number To 1 atTime: 0

Figure 16. Macroexpansion of Socharis*

6.4. *Information-Processing Model*

In order to define Information-Processing Model of an I-KBS it is necessary to perform six tasks:

1. Define input and output for every node of PSH. I.e. explicitly specify all variables that participate in a particular node prescribing whether the variable is input, output, or neither.
2. Determine whether or not the result should be passed to an upper device in PSH.

3. Define default values of input variables. I.e. assign values to the variables that will be used in this node in case a user or previous problem-solving activity has not assigned them.
4. Define mapping between different nodes. If a node takes its input values from the other node(s), it is necessary to explicitly specify that variables of the node take their values from a different node(s).
5. Define all preconditions to the nodes. I.e. specify qualitative, quantitative, and existential predicates that should be satisfied in order for the node to be activated. Qualitative and quantitative predicates are the tests on qualitative and numerical I-KBS variables respectively. In case of multiple predicates the system performs logical AND operation for all qualitative and quantitative predicates. Existential predicate is the table in which the designer indicates what variables should have values (be set) for the initiation of the node. It is possible that a particular node could be initiated with different sets of input variables therefore, S-Force performs logical OR operation for all existential predicates of the node.
6. Define the node's activation control that depends on multiple output of other node(s). There are two types of this kind of control implemented in S-Force: iteration on the multiple output of other node(s) or conditional activation of a node using multiple output of other node(s) as a test expression.

Port managers introduced in this dissertation resolve these tasks through convenient interface and automation of I-KBS building.

6.5. Information-Processing Model: PSH

S-Force gives two ways to define input and output information for every node of PSH:

1. Manually: by filling database with variables as its normally done within Generic Task Integrated Toolset and then updating port managers' contents from this database.
2. Semi-automatically. In this mode S-Force automatically imports variables from the databases of cooperating problem-solvers and fills in internal variables for the ports. The designer then adds more variables if necessary, distributes variables into input and output variables' lists, etc.

Semi-automatic mode takes part when the I-KBS designer associates a behavior of a node with a particular existing GT-based problem-solver. S-Force imports all the variables from this problem-solver, converts variable types to the necessary standards, updates I-KBSs database with these variables, and adds these variables to the set of internal variables of the node. After all the variables of a problem-solver are imported into the I-KBS model, each variable is given a flag that indicate whether this variable is input, output, or neither.

Finally, for each node of the PS hierarchy the designer identifies default output variable: a special kind of variable that holds values of the result of the node's execution and whether or not this variable should be passed up the problem-solving hierarchy.

There is no strict limitation on the direction of the variable assignment process: top-down or bottom-up traversal of PSH. On practice, the designer uses a mix of these strategies to see which study matches better to a specific part of the PSH. Generally, it is a good idea to start with the node that is called first in the macroexpansion, and works its way down the macroexpansion network.

6.5.1. Socharis* Example

Updating I-KBS's database with variables imported from a problem-solver is done automatically at the time the designer develops the Functional Model of the I-KBS and uses state change link type "by Problem-solver", as was explained in previous section. In order to access information about node's variables it is necessary to action-click on the node in the `PS Hierarchy Window` and select `Browse Port Manager` as shown in the Figure 17.

The Figure 18 represents the top pane of the `Port Manager Interface` for `Lay-up Refiner` node that provides front end to Port Manager's functions for managing variables.

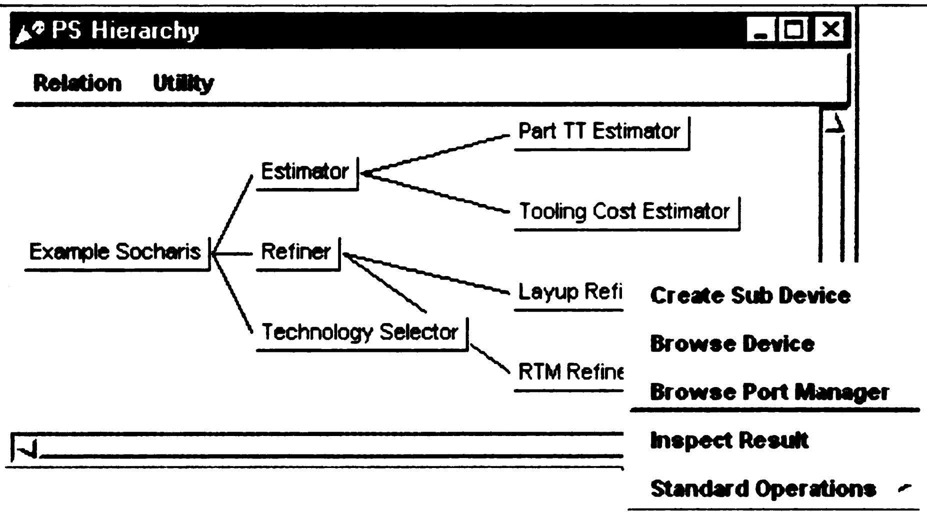
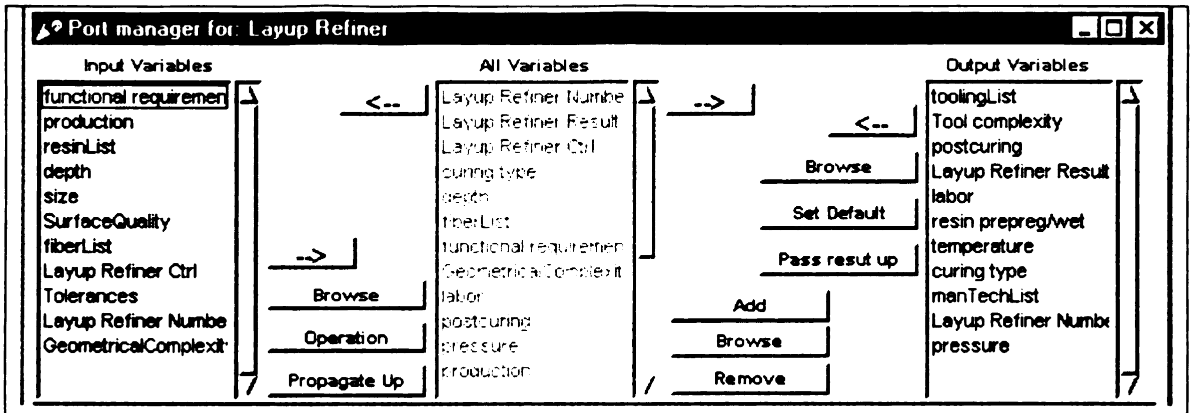


Figure 172. Opening Port Manager interface



Port manager for: Layup Refiner

Input Variables

functional requiremen
production
resinList
depth
size
SurfaceQuality
fiberList
Layup Refiner Ctrl
Tolerances
Layup Refiner Numbe
GeometricalComplexit

All Variables

Layup Refiner Numbe
Layup Refiner Result
Layup Refiner Ctrl
curing type
depth
fiberList
functional requiremen
GeometricalComplexit
labor
postcuring
pressure
production

Output Variables

toolingList
Tool complexity
postcuring
Layup Refiner Result
labor
resin prepreg/wet
temperature
curing type
manTechList
Layup Refiner Numbe
pressure

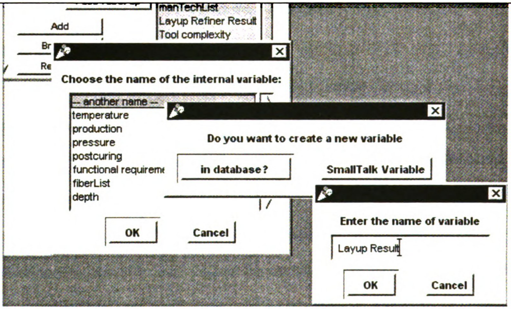
Figure 18. Upper pane of the Port Manager Interface that allows manipulating with the node's variables

The central list is the list of all variables of the node: input, output, and internal. By clicking on the buttons at the right lower corner of this list the user can browse or remove a selected variable. To add a variable to the list of internal variables the designer clicks on `Add` button and chooses a variable from the database or by selecting '—another name—' from the list has a possibility to add a new variable to the I-KBS. A variable from the central list could be redirected to the input or output lists by selecting a variable and clicking on the left (to transfer to the input list) or right (to transfer to the output list) arrow.

Button `operation` is associated with the input variables' list. It enables the designer to assign actions that have to be done with the variables before the node is being activated (preconditions, defaults, mappings, and controls). The detailed description of each of mentioned operation would be given in the next Sections. Button `Propagate Up` is mostly used in the development stage when it is necessary to propagate the variable to the upper level node (for variable mapping purposes for example.) To unload a variable from the input list the designer selects it and click on the right arrow button located above `Browse` input variable button.

Right hand side of the variable pane contains the list of output variables and four buttons:

1. left arrow for unloading a selected variable from the output list
2. `Browse` to browse a selected variable
3. `Set Default` to set the selected variable a default variable for the resulting value of node's operation, and
4. `Pass result up` button. This button sets the flag on the output variable that it should be passed to the upper node after the node finished operation.



Choose the name of the internal variable:

Do you want to create a new variable

Enter the name of variable

Layup Result

Figure 19. Adding new variable to a node

If the node functionality is achieved through the "By Problem-solver" behavior, then result of, problem-solver is saved in to a variable of Smalltalk type Dictionary. In this case the designers executes the following sequence of actions to determine where the result of problem-solver is being stored and how it is being passed up.

1. Create new variable by clicking Add button and Selecting "`another name`'
2. In the following dialog window, click on the Smalltalk variable button and enter the name of this variable. Figure 19 shows the cascaded set of interfaces that result in creating new variable Lay-up Refiner Result
3. The next step is to move this variable to the output variables' list. This is done by selecting Lay-up Refiner Result variable in the list of all variables and clicking on the right arrow at the upper right corner of the all variables' list.
4. Then, it is necessary to make this variable the Default output variable. Selecting this variable in the output variables' list and clicking Set Default does this.
5. Finally, if the designer decides to pass this variable up one level he or she selects this variable and clicks `Pass result up` and then, selects the output variable of the upper node where to pass the result. Note 1: the output variable of the upper node should already exist. Note 2, if a node receives results from more than one of its children it concatenates them, no intermediate result is lost. Figure 20 shows the set of interfaces that result in passing the value of `Lay-up Refiner Result` to the `Refiner Result`.

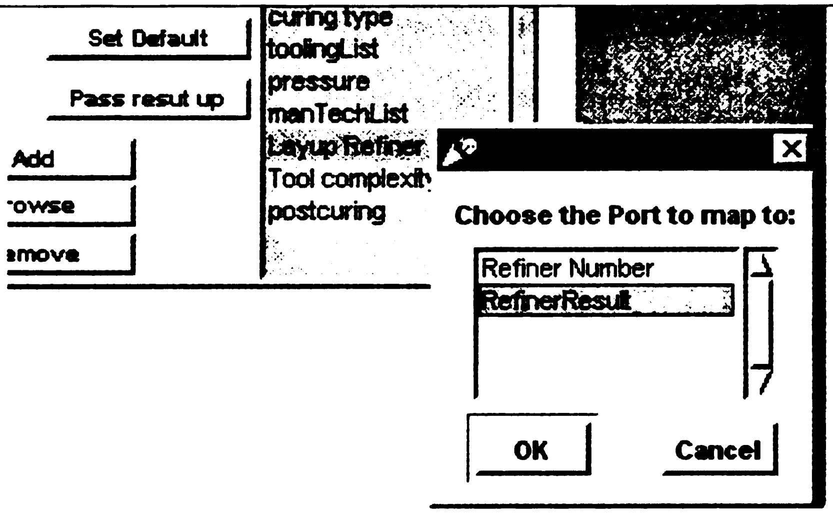


Figure 20. Passing the result variable up one level

The description of input and output variables for port manager for Lay-up Refiner is shown in the Table 7.

Input Variables	**Internal Variables**	**Output Variables**
Lay-up Refiner Number	Lay-up Refiner Number	Lay-up Refiner Number
Lay-up Refiner Ctrl	Lay-up Refiner Result	Lay-up Refiner Result
Depth	Lay-up Refiner Ctrl	curing type
FiberList	curing type	labor
Functional requirements	depth	postcuring
Geometrical Complexity	FiberList	pressure
Production	functional requirements	resin prepreg/wet
ResinList	Geometrical Complexity	temperature
Size	labor	Tool complexity
Surface Quality	postcuring	toolingList
Tolerances	pressure	manTechList
	production	
	resin prepreg/wet	
	ResinList	
	size	
	Surface Quality	
	temperature	
	Tolerances	
	Tool complexity	
	toolingList	
	manTechList	

Table 7. Variables of Lay-up Refiner Port Manager.

All variables but Lay-up Refiner Ctrl, Lay-up Refiner Result, Lay-up Refiner Number, and manTechList are the variables from the correspondent problem-solver operation_Selector_New. SimpleHCPS. Lay-up Refiner Result contains the result of the execution of this node. In this case, it will hold the name of the established specialists of hierarchical classifier. Lay-up Refiner Number is a bookkeeping variable that helps keep track of PS stages. Lay-up Refiner Ctrl is the control variable that controls the execution of this node. Finally, manTechList is the output control variable that holds input control value that activated this node.

Descriptions of input/output/internal variables of other port managers for Socharis* can be found in Appendix A.

6.6. Information-Processing Model: Input and Output

Default values of input variables are used in cases where an existential predicate requires the presence of this variable but it was not supplied to a node. This feature of S-Force gives the designer an ability to develop robust systems, which are able to work with incomplete data. However, it is not a necessary step in the I-KBS development and might be skipped.

6.6.1. Socharis* Example

To access an interface that allows various manipulating with input variables including definition of default variables the designer clicks on the operation button. This opens the bottom pane of the Port Manager Interface (Figure 21). At the lower right corner of this pane located the Default Values interface: a table with the left column containing all input variables and right column representing their default values, where question mark ('?') means undefined.

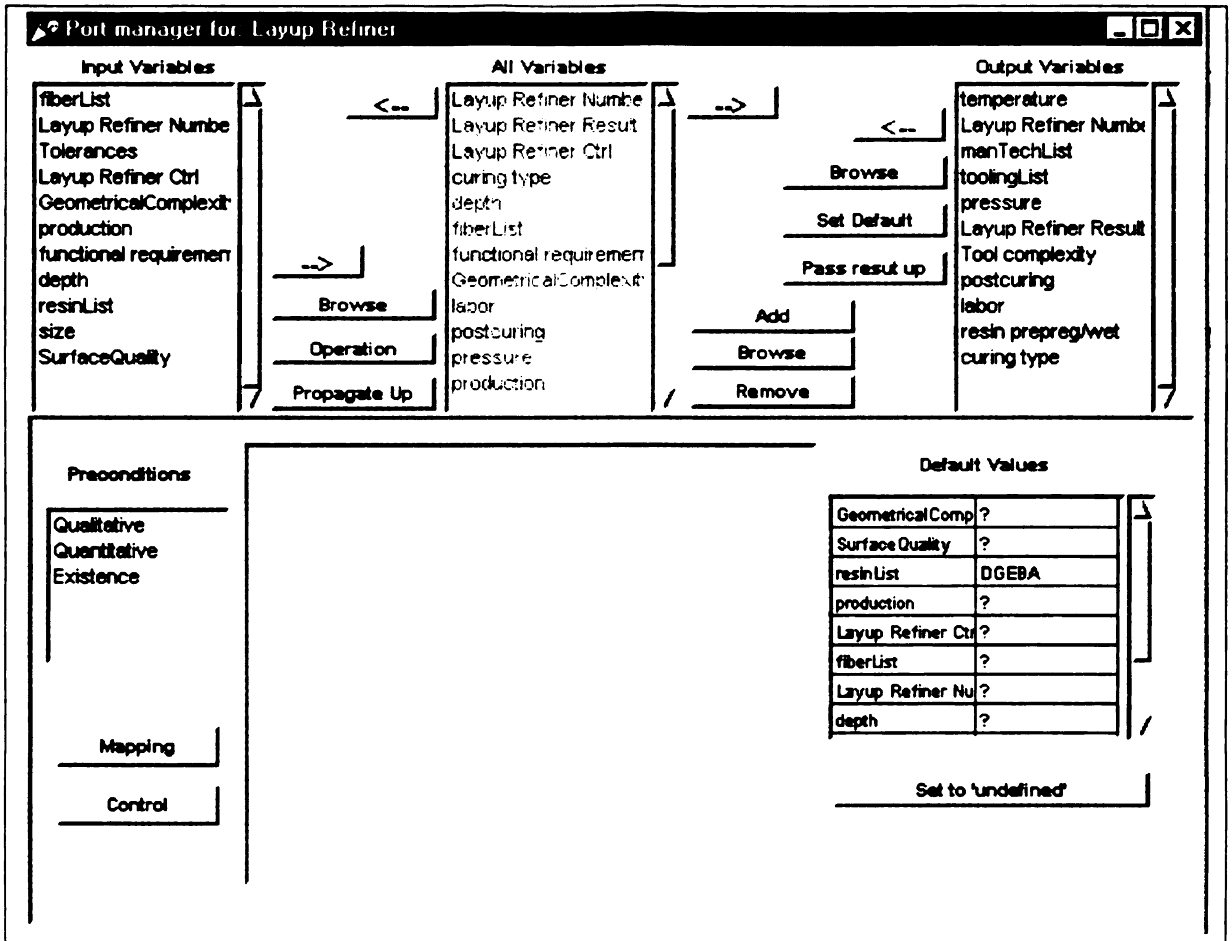
If the designer wants to set a default value for a variable he or she:

1. Clicks on the row in the `Default Values` table containing the desired variable
2. Selects the default value from the pop-up menu
3. Clicks Ok

To reverse default values to undefined state the designer clicks `Set to 'undefined'` button.

It is important to exercise caution in using this feature. If used improperly it could mask problems in the I-KBS architecture such as gaps in variable mapping. In fact, it is better to develop the system first without employing default values' tables and only then add them.

Figure 21 shows that the default value for `ResinList` variable of `Lay-up Refiner` node is set to `DGEBA`. This means that if the value of `ResinList` was not set by previous problem-solving activity it will be set to `DGEBA`.



Input Variables

fiberList
Layup Refiner Numbe
Tolerances
Layup Refiner Ctrl
GeometricalComplexit
production
functional requiremen
depth
resinList
size
SurfaceQuality

All Variables

Layup Refiner Numbe
Layup Refiner Result
Layup Refiner Ctrl
curing type
depth
fiberList
functional requiremen
GeometricalComplexit
labor
postcuring
pressure
production

Output Variables

temperature
Layup Refiner Numbe
manTechList
toolingList
pressure
Layup Refiner Result
Tool complexity
postcuring
labor
resin prepreg/wet
curing type

Preconditions

Qualitative
Quantitative
Existence

Default Values

GeometricalComp	?
Surface Quality	?
resinList	DGEBA
production	?
Layup Refiner Ctrl	?
fiberList	?
Layup Refiner Nu	?
depth	?

Figure 21. Interface for definition of default values of input variables

6.7. Information Processing Model: Mapping

The next task is to specify how the information flows from one node of PSH to another during the problem-solving process. That is, to determine variable mapping between different parts of PSH. S-Force gives the ability to do so by using Mapping feature of Port Managers.

Any node could receive information from many different sources during the problem solving. Moreover, depending on the problem-solving context a node could receive different kinds and type of information from the same node at different times. To accommodate this, S-Force enables the designer to create multiple variable mappings. At run time the mapping works as follows: once the node is reached during the problem solving, S-Force iterates over the list of mappings and fills the values of all the variables that are defined there.

There are two things that the I-KBS developer should keep in mind while mapping variables from one node to another:

1. Both variables (source and destination) should be defined using the same working ontology. Failure to do so will lead to a non-functional I-KBS since S-Force does not have enough information to translate the values. One of the solutions to this is to create translator problem-solver and map the variables through it.
2. If the mapping to be done from multiple output of a particular node then it is necessary to appoint a variable that would control mapping from this multiple output. The exact procedure of how to use multiple outputs to control a node is described in the Section 6.9.3 (RD Control).

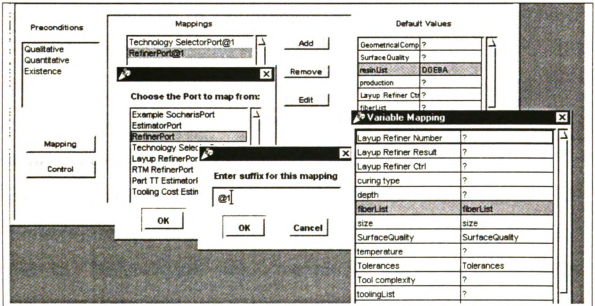
6.7.1. Socharis* Example

After clicking the `operations` button in the Port Manager's window the designer accesses the interface that allows manipulating with input variables of the node before the node's activation (Figure 21). To start working with mappings the designer clicks on the `Mapping` button, which calls mapping managing interface in the middle of the bottom pane. Then, to create mappings the designer repeats the following sequence of actions for each desired mapping:

1. Clicks on the `Add` button
2. From the appeared list of ports managers associated with every node of PSH, the designer selects the port manager to make mapping from (source port).

3. Enters the suffix (usually a number) to differentiate between multiple mappings from the same port and clicks `Ok`.
4. The interface that allows defining a mapping is similar in feel to the `Default Values` interface. That is, by clicking on the row containing the name of the destination variable in the left column the designer calls list of all available source variables from the source port.

The Figure 22 shows the cascaded set of interfaces that result in addition of mapping <u>`RefinerPort@1`</u> for `Lay-up Refiner` Port Manager (a destination) from Refiner Port Manager (a source).



Qualitative
Quantitative
Existence

Technology SelectorPort@1
RefinerPort@1

GeometricalComp	?
SurfaceQuality	?
resinList	DGEBA
production	?
Layup Refiner Ctr	?
fiberList	?

Choose the Port to map from:

Example SocharisPort
EstimatorPort
RefinerPort
Technology Selec
Layup RefinerPor
RTM RefinerPort
Part TT Estimator
Tooling Cost Estin

Enter suffix for this mapping

@1

Layup Refiner Number	?
Layup Refiner Result	?
Layup Refiner Ctrl	?
curing type	?
depth	?
fiberList	fiberList
size	size
SurfaceQuality	SurfaceQuality
temperature	?
Tolerances	Tolerances
Tool complexity	?
toolingList	?

Figure 22. Adding of mapping

Lay-up Refiner node has two mappings:

1. One from `Refiner` node: all input values necessary for running the Lay-up multiple routine designer.
2. Secondly from `Technology Selector` node. This mapping indicated that input control variable of Lay-up Refiner is to be taken from output variable TSResult. The detailed description of setting up this kind of control is given in the Section 6.9.

The designer can also delete and edit a selected mapping by clicking on buttons `Remove` and `Edit` correspondingly.

6.8. Information Processing Model: Preconditions

One of the roles the Port Managers play in an I-KBS is the setting and testing the preconditions. This role is inherited from the *precondition clause* of the traditional FR. The *precondition clause* usually contained a set of qualitative and quantitative predicates, all of which should be satisfied before the node is allowed to be executed. Meanwhile, the described framework augmented the *precondition clause* with a new kind of precondition: existential predicate. Existence predicate contains names of input variables whose presence is necessary for running the node. This ensures that the node with multiple functionality and/or behaviors each of which uses just a subset of the data runs as soon as it receives enough data to execute some of its paths.

6.8.1. Qualitative and Quantitative Predicates

Qualitative predicates are designed to test I-KBSs variables that take value from a predefined collection of legal values, whereas quantitative predicates test the values of I-KBSs numerical variables. A test expression for quantitative precondition may include complex formulas that could contain other I-KBSs variables. One important thing to remember while writing testing formulas for quantitative predicates is that precedence of arithmetic operations in these formulas is governed by the Smalltalk laws: all formulas are read left to right and no implicit precedence is stated (that is 2+3*5 = 25.)

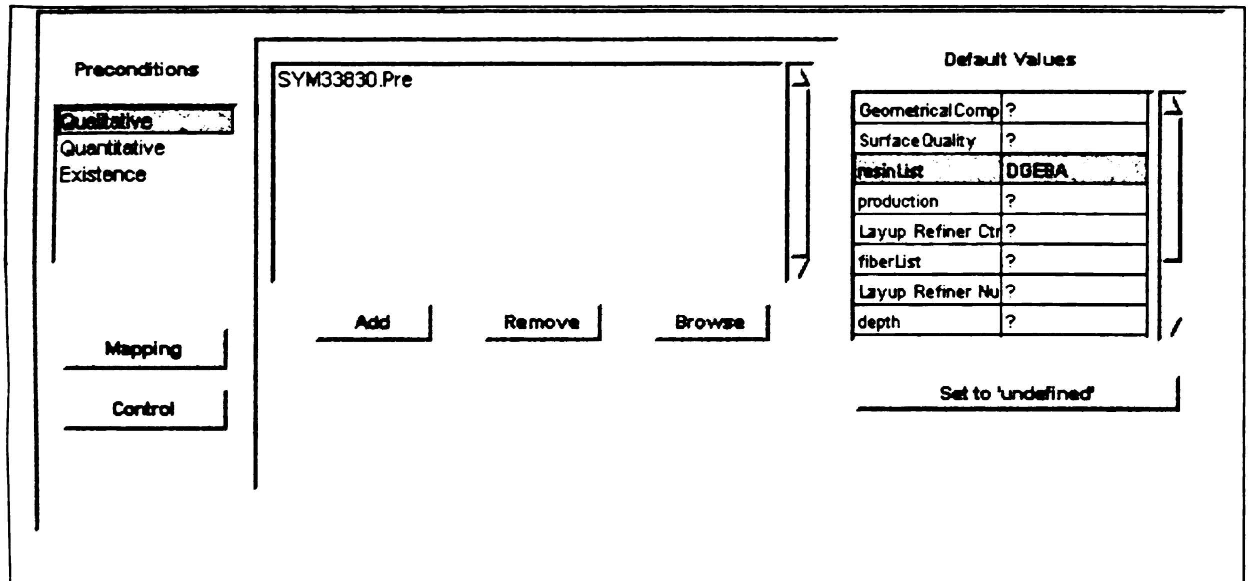
6.8.2. Existential Predicates

As it was mentioned earlier, existential predicates are used to specify sub-sets of input variables that should be *set* in order for the predicate to be satisfied. If there are more than one existential predicate defined for a particular port manager, then the collection of existential predicates is satisfied if either of members of this collection is satisfied.

In addition to the function of activating the node whenever it has enough defined input variables to activate some of the behaviors, existential predicate play another important role. In a case when the node is called but there is not, enough information to run the node S-Force uses existential predicates as a template to fill in values of input variables from the default values table.

6.8.3. Socharis* Example

To access interfaces for manipulating predicates the designer selects a `Qualitative`, `Quantitative`, or `Existential` from the predicate types list located in the upper left corner of lower pane of Port Manager Interface as shown in the Figure 23.



Preconditions

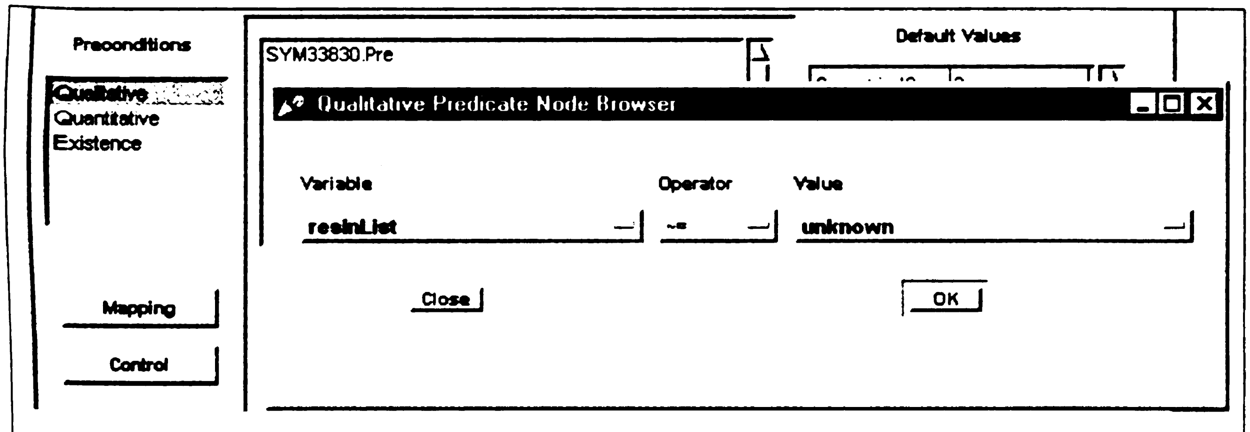
Qualitative
Quantitative
Existence

SYM33830.Pre

Default Values

Geometrical Comp	?
Surface Quality	?
resinList	DGEBA
production	?
Layup Refiner Ctr	?
fiberList	?
Layup Refiner Nu	?
depth	?

Figure 23. Predicate types list located at the upper left corner



Preconditions

Qualitative
Quantitative
Existence

SYM33830.Pre

Default Values

Qualitative Predicate Node Browser

Variable: resinList

Operator: ~=

Value: unknown

Figure 24. Qualitative predicate interface

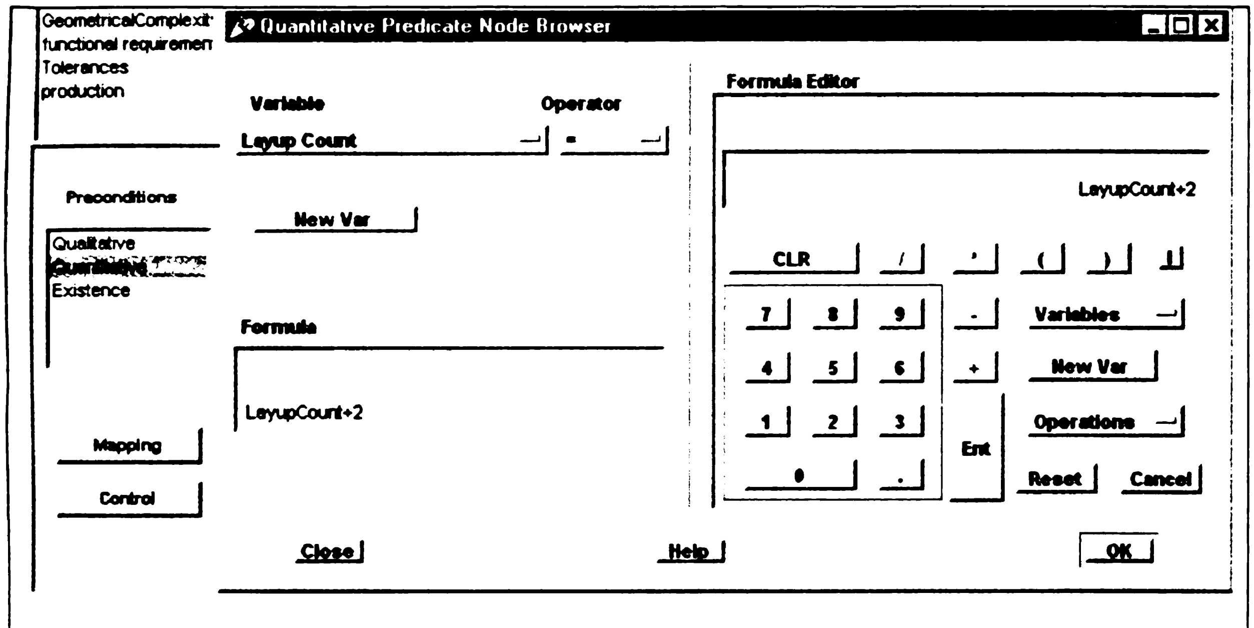
After the predicate type is selected, the central window shows the list of defined predicates of this type. The designer can add new predicate, remove, or browse selected predicate. To add a qualitative predicate the I-KBS designer:

1. Selects `Qualitative` in the predicate types list
2. Clicks `Add` in the central window

3. In the appeared interface (Figure 24) the designer selects a qualitative variable from the `Variable` drop list, equals on not equals sign from the `Operator` drop list and a desired value of this variable from the `Value` drop list.
4. Clicks `Ok`

To add a quantitative predicate the I-KBS designer:

1. Selects `Quantitative` in predicate types list
2. Clicks `Add` in the central window
3. In the appeared interface (Figure 5) the designer selects a `Quantitative` variable from the upper left `Variable` drop list, equal on not equals sign from the `Operator` drop list. In the right hand side calculator the designer enters the necessary formula using the `Variables` drop list to enter variables in the formula (operator precedence agrees with Smalltalk standards).
4. Clicks `Enters` and `Ok`



Variable: Layup Count

Operator: =

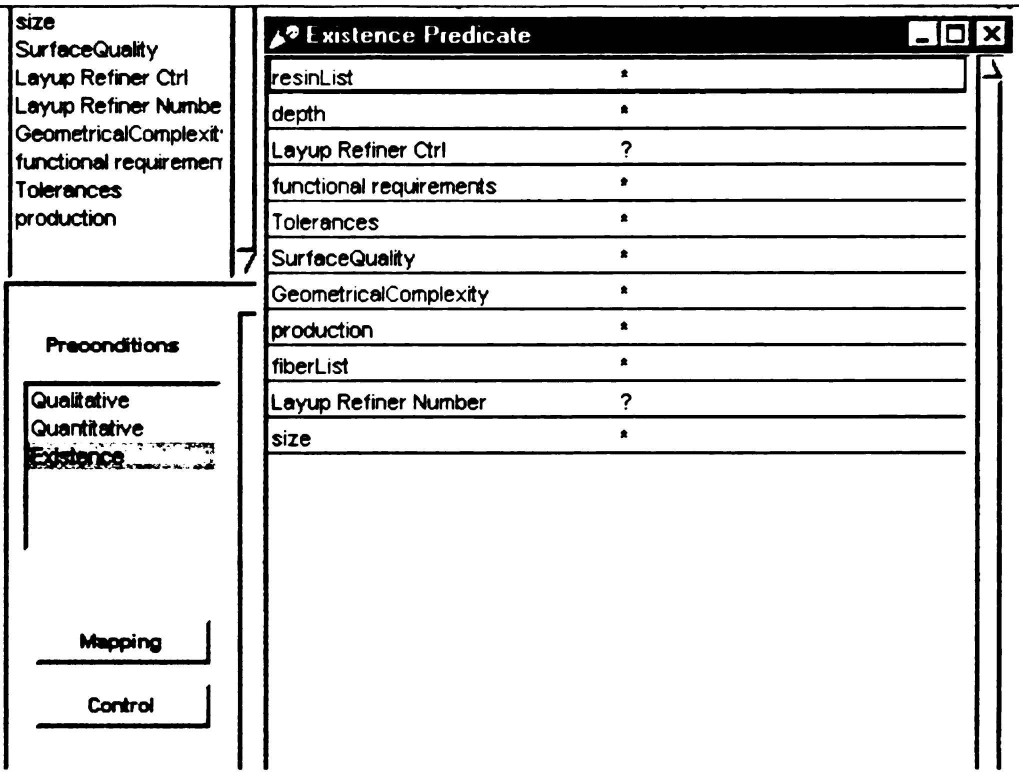
Formula: LayupCount+2

Formula Editor: LayupCount+2

Figure 25. Quantitative predicate interface

To add an existential predicate the I-KBS designer:

1. Selects `Existence` in the predicate types list
2. Clicks `Add` in the central window
3. In the Existential predicate interface (Figure26) the designer clicks on the rows that contains necessary variable therefore changing '?' (not needed) to '*' (should be set)
4. Closes table



resinList	*
depth	*
Layup Refiner Ctrl	?
functional requirements	*
Tolerances	*
SurfaceQuality	*
GeometricalComplexity	*
production	*
fiberList	*
Layup Refiner Number	?
size	*

Figure26. Existential predicate interface

6.9. Information Processing Model: Activation Control

Predicates' mechanism allows testing preconditions for single valued variables and does not account for multiple results and multiple valued variables that are produced by problem-solvers employed in an I-KBS. However, it is highly desirable to have this feature to enhance the control over the I-KBS's execution.

S-Force operates with two basic types of GT problem-solvers, each of which could produce multiple results: Hierarchical Classifier (HC) and Multiple Routine Designer (MRD). Hierarchical Classifier results in a set of parameters with values: 'matched', 'neutral', 'against', etc. These values correspond to the HC's confidence of how a particular parameter fits to the situation described through input variables. MRD's output is a collection of sets of instantiated parameters. Each set of this collection represents a valid solution to a given problem. It would be reasonable to allow the designer of an I-KBS to use these output to control the problem-solving process by either iterating over the MRD's output or selecting a next part of I-KBS to activate based on HC's output.

In addition to above S-Force enables the designer use boosting, i.e. combining the results of a number of problem-solvers into a single result by selecting the results intersection (in a set theoretic notation). This technique is widely used in the many areas of artificial intelligence including machine learning (Schapire, Singer et al. 1998; Schapire and Singer 1998). In S-Force's version, the designer can perform boosting of results of several hierarchical classifiers. This technique saves the time at the knowledge acquisition stage by allowing solving a given problem using different perspective in the multitude of problem-solvers in cases when the structure and knowledge architecture of a single problem-solver (that considers all aspects of the problem) is complex or unclear.

6.9.1. HC Control

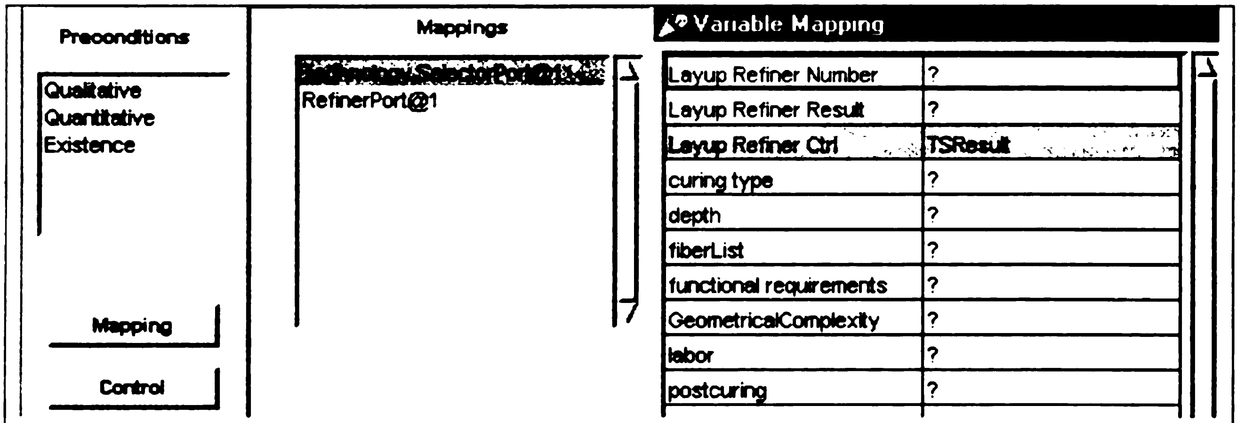
The result of Hierarchical Classifier (HC) is the list of its leaf specialists with their respected established values (e.g. matched, neutral, against). Hereafter, I will refer to all the specialists with values 'matched' as established specialists.

It is often the case that the further I-KBS execution depends on the established specialists (e.g. a particular HC selects one or several named parts of an I-KBS to execute next.) This behavior is very useful in implementing and modeling blackboard like mechanisms in an I-KBS, where the context defines activation of the next module at the run time. S-Force enables the designer to use this kind of control over the I-KBS behavior in the run time through the HC Control. The designer creates a special input variable (a control variable) that is being mapped to from the source node - the node that produces multiple result. Then this control variable is associated with a set of values. When at least one of these values are matched to one of the values that are mapped from the source node the current node will be activated. To finalize definition of the HC control the designer should designate a variable that will keep the value that caused the activation of the node. This helps to keep current problem-solving context up-to-date.

While the HC control is most suited to be used in adjunct with Hierarchical Classifier type of problem-solver it can also be used with any node that produces multiple results similar in nature with the result of HC. The use of the HC control with the booster is one of the examples of such operation (Section 6.9.5).

6.9.2. Socharis* Example

In the described example, the Lay-up Refiner node is being activated if the Technology Selector node is HC `operation_Selector_New`. `SimpleHCPS` has `Lay-up` as one of the established specialists. In previous section, I showed how to use the `Lay-up Refiner Port Manager` to execute mapping from a node to a node of PSH. The Figure 27 shows the mapping of the resulting variable of `Technology Selector` `TSResult` to `Lay-up Refiner Ctrl`.



Layup Refiner Number	?
Layup Refiner Result	?
Layup Refiner Ctrl	TSResult
curing type	?
depth	?
fiberList	?
functional requirements	?
GeometricalComplexity	?
labor	?
postcuring	?

Figure 27. Mapping of HC Input Control

The designer assigns the control and appoints the variable that will hold the value that activated the node as follows:

1. Adds new variable from I-KBS's database `manTechList` and transfers it to the output variables' list (this variable will hold the value that activated the node).
2. Adds new SmallTalk variable `Lay-up Refiner Ctrl` and transfers it to the input variables' list (this variable will hold values mapped from the other node).
3. Clicks on the `operations` button to access input operation interface.
4. Creates mapping from port `Technology SelectorPort – Technology SelectorPort@1` – as described in Section 1. 7. 1 and maps the variable `Lay-up Refiner Ctrl` from `TSResult.`
5. Clicks on the `Control` button. Then clicks the `Add` button and selects `Lay-up Refiner Ctrl` from the list of available control variables and click `Ok`
6. Selects `Lay-up Refiner Ctrl` and clicks the `HC control` button
7. Clicks `Yes` to confirm choice of established specialists from the list.

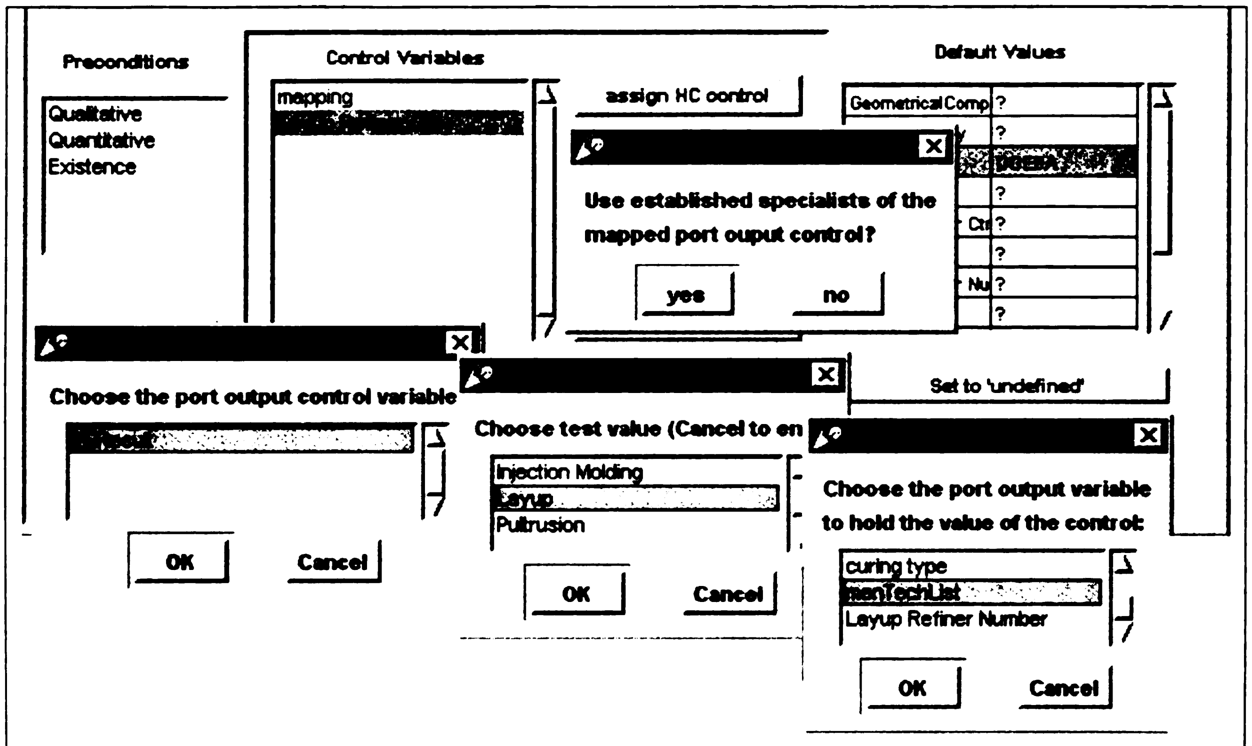
8. Selects TSResult from the list of result variables of Technology Selector Port.
9. Selects Lay-up from the list of available leaf specialists and clicks OK, then clicks Cancel to stop selection.
   9.1. If the designer needs to select more then he or she specialist repeats step 8 until he or she selects all necessary specialists, then clicks Cancel
   9.2. If the designer wishes to input names of specialists by hand, he or she can do it by selecting No at the Step 7 and then entering desired values by hand using '%' as delimiter.
10. From the list of output variables of current port selects variable where to send the value that activated the node. In our case manTechList.

The set of cascaded interfaces in the Figure results in the assigning HC control to the Lay-up Refiner Ctrl. This control will be satisfied if TSResult will contain Lay-up as one of the established specialists.

6.9.3. RD Control

RD Control is associated with the typical multiple routine designer output: a collection of sets of design parameters with established values. One can expect to have the system iterate over the collection of the parameters using each set in problem solving to explore the solution space.

In S-Force the designer is able to set this kind of the control of the node by mapping output variable of a node producing such kind of result (e.g. a node whose behavior is linked to a multiple routine designer) to an input control variable.



Use established specialists of the mapped port ouput control?

Choose the port output control variable

Choose test value (Cancel to en

Injection Molding
Layup
Pultrusion

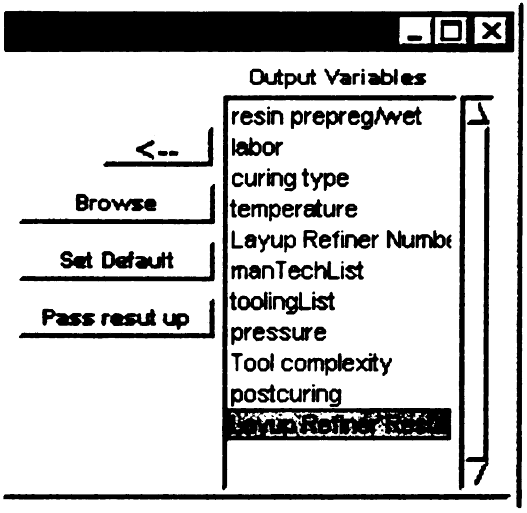
Choose the port output variable to hold the value of the control:

curing type
manTechList
Layup Refiner Number

Figure 28. Adding HC Control

Similar to the HC Control, the RD Control can be used not only with the nodes whose behavior is achieved through the MRD problem-solver. In fact, the result of any node can be used to control iterations if this result complies with the structure of the output of MRD node.

It is important however to take care of variable naming and used ontology. That is the names of the parameters in MRD output sets should correspond to the names of the input variables defined in mapping. The same is true for the used ontology: ontologies of the source node and the destination node should agree in the variables used for mappings. Otherwise, the problem-solving process would be compromised.



resin prepreg/wet
labor
curing type
temperature
Layup Refiner Numbe
manTechList
toolingList
pressure
Tool complexity
postcuring
Layup Refiner Resul

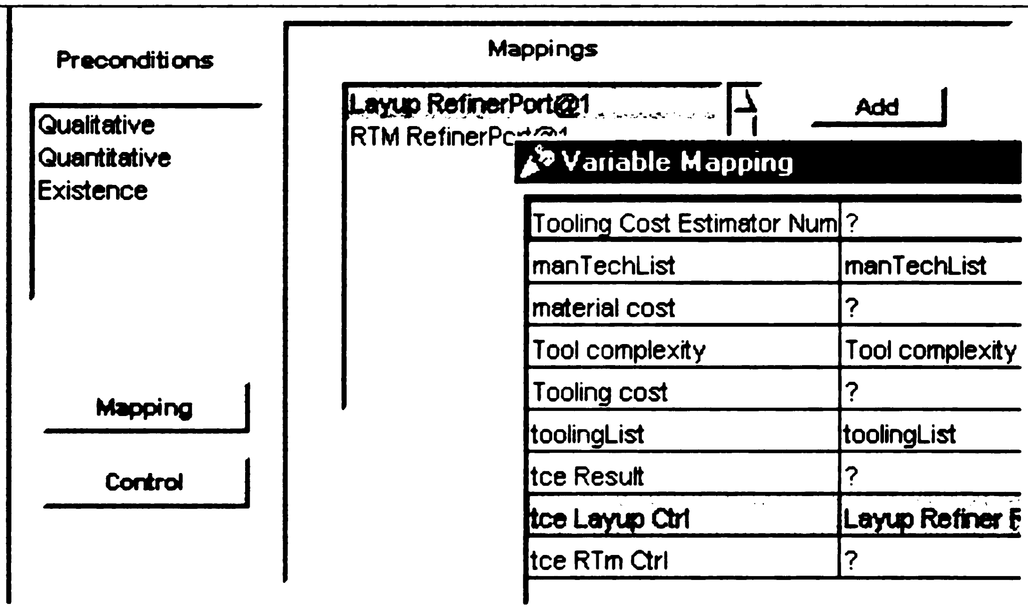
Figure 29. Output Variables of Lay-up Port Manager

6.9.4. Socharis* Example

The assignment of the RD Control to an input control variable is relatively straightforward task. In our example Tooling Cost Estimator should run for all sets of parameters set by either or both Lay-up Refiner and RTM Refiner nodes. The following sequence demonstrates how the I-KBS designer creates a RD Control from the Lay-up node to the Tooling Cost Estimator node.

1. Confirms that Lay-up Port Manager's list of output variables includes all necessary information: all Lay-up. BasicRDProblemSolver output variables and the node's designate output variable that holds the result of MRD run (Figure 29)
2. Creates new SmallTalk variable tceLay-upCtrl for Tooling Cost Estimator port manager and transfers it to the input variables' list.
3. In port manager for Tooling Cost Estimator creates new mapping Lay-up Refiner Port @1 as shown in the Figure 30. Notice that the designer maps not only the variables that are needed for the run of the MRD associated with

the node, but also the control variable. Again, it is important to have Lay-up Refiner resulting variables match Tooling Cost Estimator input variables compliant in names and ontology



Tooling Cost Estimator Num	?
manTechList	manTechList
material cost	?
Tool complexity	Tool complexity
Tooling cost	?
toolingList	toolingList
tce Result	?
tce Layup Ctrl	Layup Refiner F
tce RTm Ctrl	?

Figure 30. Mapping of Lay-up Result onto Tooling Cost Estimator input and control variables

1. The next step is the assignment of the RD Control to the tceLay-upCtrl variable. To do so the designer accesses operations Interface and clicks on the Control button to invoke control manipulating interface.
2. Clicks Add button and select tceLay-upCtrl from the list of available variables.
3. Selects tceLay-upCtrl in the central window of operations interface and clicks RD Control.
4. Select Lay-up Refiner Port Manager from the list of available ports to specify that control is indeed passed from the Lay-up Refiner Node.

6.9.5. Boosting

Combining the results of several different problem-solvers working on the same task from different perspectives proved an effective technique; this not only saves the time during the knowledge acquisition stage, but also tends to outperform a traditional approach. This method of combining the results is called boosting.

S-Force employs a form of boosting that allows taking the results (in form of lists of values) of two nodes and combining them by finding their intersection. By applying this procedure iteratively, one can perform boosting for multiple nodes.

6.9.6. Socharis* Example

Socharis* does not use this capability of S-Force, however in the original project this feature was used when the result of two Technology Selectors were combined to prune the list of applicable technological processes. Each Technology Selector chooses a list of appropriate manufacturing processes depending on: first – part geometrical and functional description; second – part material.

To assign a node to a booster function it is necessary to perform the following steps:

1. While defining behavior of a booster node, define state change as "By Definition" and enter 'of booster' in the input field.
2. Open port manager interface of this node and create three Smalltalk variables `list1`, `list2`, and `resulting` variable. Note, naming list1 and list 2 is obligatory

3. Transfer `list1` and `list2` to input list, `resulting` to output list
4. Make `resulting` variable a default output variable
5. Make two mappings from port managers corresponding to two nodes that produce results for boosting. Map output variables of these nodes to `list1` in one mapping and to `list2` in another

The result of booster node is similar in nature to the output of the node associated with HC: list of established specialists. Therefore, it can be used in another boosting operation.

6.10. Final Remarks

After the I-KBS developer built three models: problem-solving hierarchy, Functional Model, and information processing model, he or she can run the system by creating a new case (an instantiated set of I-KBS's input variables) and selecting PS → Run from the S-Force Launcher window. The result of problem-solving process could be browsed by selecting `Inspect Result` in action-click pop-up menu for every node in PSH. If designed correctly then the global result will be located in the root of the PSH.

6.10.1. Sequence of nodes' activation and mapping sequence

To build a fully functional I-KBS it is necessary to understand how S-Force works while running an I-KBS. The run of an I-KBS is a breadth-first traversal of the macroexpansion with performing precondition check, mapping, and defaults assignment. The sequence of actions that S-Force executes when it decides on an activation of a node are as follows:

1. Check if it is necessary to perform HC, RD, or Booster control and mapping.

2. In case when no control is specified:

    2.1. Assess all input variables

    2.2. Do mapping if exists

    2.3. Check Existential predicate and set defaults if necessary

    2.4. Test remaining preconditions

    2.5. Execute node (e.g. run associated problem-solver, perform state change)

3. In case when HC Control is specified:

    3.1. Assess mapped control values and compare them to the list of values that activate the node

    3.2. If the condition satisfied then perform steps 2. 1 through 2. 5

4. In case when RD Control is specified:

    4.1. For each sets in input MRD result collection

    4.2. Map input sets

    4.3. Execute steps 2. 1 through 2. 5

5. In case when boosting is specified:

    5.1. Map two input lists and find their intersection

6. Pass the result to the node up in the hierarchy if specified

7. Update I-KBS current case with the obtained as a result of the node run values

6.10.2. Color-coding

While building a complex I-KBS it is important to keep track of what parts of the system has been assigned functionality and if assigned what it is. S-Force provides a special color coding of the nodes in PSH to quickly overview the system in design. By selecting `Utility →Show PS Types in the PS Hierarchy window`, the designer accesses the interface where the nodes of the PSH are colored according to the specified function and/or behavior:

- RD or HC is associated with a behavior of the node
- the node's functionality is achieved using "by Knowledge" or "by Definition" behaviors
- the node exhibits multiple functionality or behavior
- node does not have a function associated with it yet.

Another color coding information available to the S-Force's user is the coloring of the nodes after I-KBS run. In this case, the color of each node corresponds to the status of node's activation during the problem-solving process:

- no precondition is specified and node has been activated
- precondition is satisfied
- precondition is satisfied and internal mapping was executed (e.g. mapping multiple routine designer's output to the input of this node)
- precondition(s) is satisfied after the default values were invoked
- preconditions are not satisfied
- the node was not called

This color-coding enables fast tracking of gaps in functionality and information passing in an I-KBS.

7. RE-IMPLEMENTATION OF SOCHARIS

The original Socharis consists of four major parts: the graphical user interface, the ontology editor, the conceptual composite assembly structure parser, and the knowledge-based kernel. This chapter describes how the framework for developing I-KBSs is used to redesign the knowledge-based core of Socharis.

The main knowledge intensive section of Socharis is the part which is responsible for generating of manufacturing alternatives, instantiating them with the sets of technological parameters and estimating the merits for each generated alternative as was discussed in Section 4.2. The problem-solving architecture of Socharis and its parts (depicted on the Figure 1) was hard coded in Smalltalk, therefore restricting the maintenance of the system to the people intimately familiar not only with Socharis's problem and Smalltalk, but also with the legacy software that Socharis is built on.

The use of S-Force, the Shell for Constructing Integrated Knowledge-Based Systems, allows re-implementation of Socharis's knowledge-based core (SocharisKB) in a framework of an advocated methodology that allows the designer and the user accessing the system, its knowledge-level architecture, and problem-solving architecture of its components.

7.1. Problem-Solving Hierarchy of Re-Designed SocharisKB

Following the domain dependent decomposition of the Socharis's problem (Section 2.2) the Problem-Solving Hierarchy (PSH) of Re-Designed SocharisKB (Re-SocharisKB) is divided into three major parts: Technology Selector, Technology Refiner, and Technology Estimator. Represented in the Figure 1 PSH of Re-SocharisKB follows the stationary task-subtask decomposition of the problem from the domain specialist's viewpoint. Where each of three major sub-problems is further decomposed into a number of smaller sub-problems that could be handled by a GT-based problem-solvers or being solved through unitary operations like *Boosting.*

Besides playing the role of task-subtask hierarchy, the PSH provides an access to every part of the I-KBS through the set of functional menus as well as enables system building and maintaining functions.

7.2. Functional Model of Re-Designed Socharis

The macroexpansion of Re-SocharisKB (Figure 32[12]) illustrates causal order of activation of node. Particularly it shows that first, the system selects technologies, then boosts the results of two selectors. Next, it instantiates the selected technologies by running correspondent problem-solver and finally, it estimates the parameterized technologies by activating nodes responsible for running estimator problem-solvers.

[12] Due to the large size of the macroexpansion, I substituted a system screenshot with its copy rearranged so it fits the space allotted.

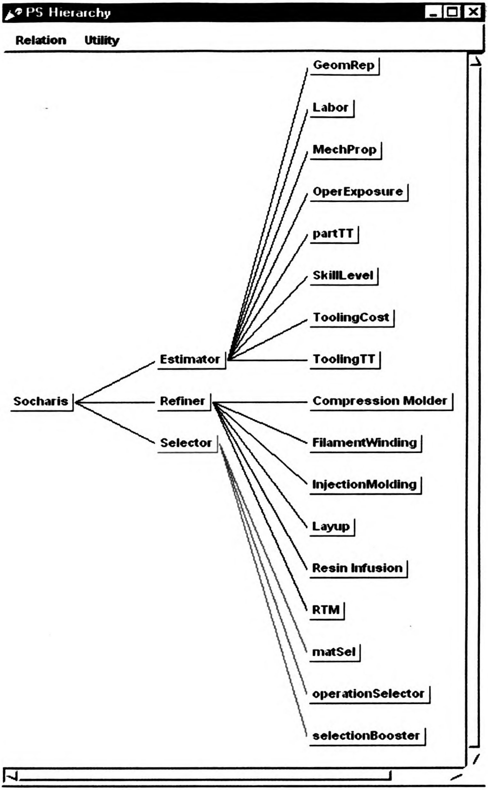


Figure 31. PSH of SocharisKB

S-Force treats a macroexpansion as a network with distributed control where every node is being invoked as soon as the system reaches the node's layer of the network. However, the node does testing the self-applicability through the mechanism of port managers. Macroexpansion gives an access to high-level information flow in an I-KBS, the level where the information is being treated as a monolithic *slug* of matter passing through the device's (I-KBSs) Functional Model.

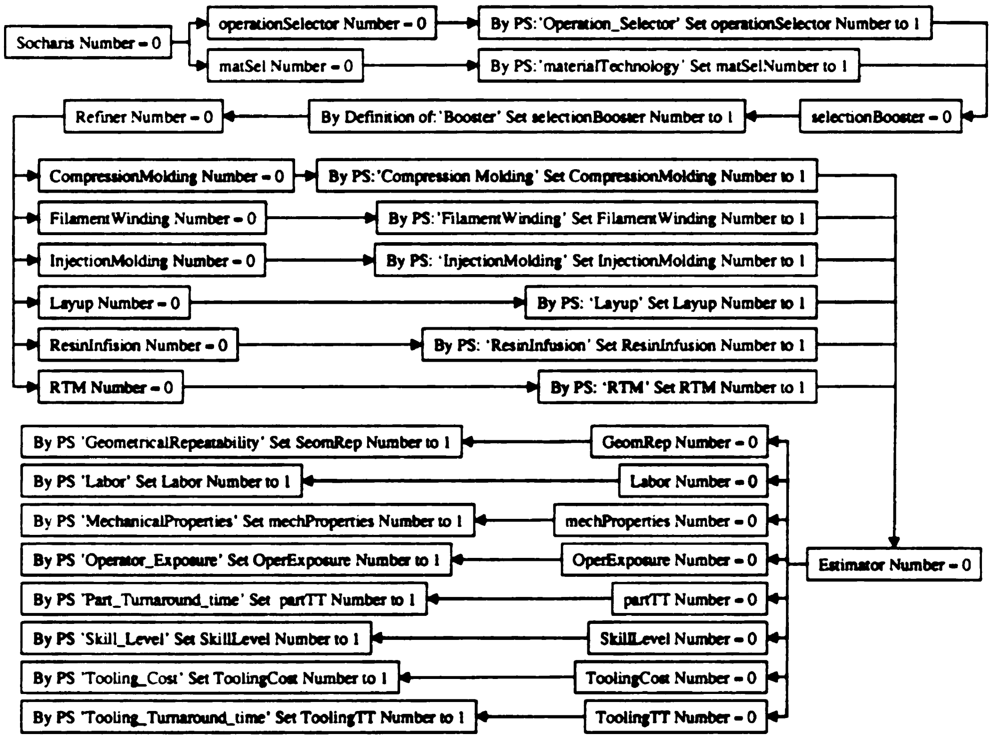

Socharis Number = 0
operationSelector Number = 0
By PS:'Operation_Selector' Set operationSelector Number to 1
matSel Number = 0
By PS:'materialTechnology' Set matSelNumber to 1
Refiner Number = 0
By Definition of:'Booster' Set selectionBooster Number to 1
selectionBooster = 0
CompressionMolding Number = 0
By PS:'Compression Molding' Set CompressionMolding Number to 1
FilamentWinding Number = 0
By PS:'FilamentWinding' Set FilamentWinding Number to 1
InjectionMolding Number = 0
By PS: 'InjectionMolding' Set InjectionMolding Number to 1
Layup Number = 0
By PS: 'Layup' Set Layup Number to 1
ResinInfision Number = 0
By PS: 'ResinInfusion' Set ResinInfusion Number to 1
RTM Number = 0
By PS: 'RTM' Set RTM Number to 1
By PS 'GeometricalRepeatability' Set SeomRep Number to 1
GeomRep Number = 0
By PS 'Labor' Set Labor Number to 1
Labor Number = 0
By PS 'MechanicalProperties' Set mechProperties Number to 1
mechProperties Number = 0
By PS 'Operator_Exposure' Set OperExposure Number to 1
OperExposure Number = 0
Estimator Number = 0
By PS 'Part_Turnaround_time' Set partTT Number to 1
partTT Number = 0
By PS 'Skill_Level' Set SkillLevel Number to 1
SkillLevel Number = 0
By PS 'Tooling_Cost' Set ToolingCost Number to 1
ToolingCost Number = 0
By PS 'Tooling_Turnaround_time' Set ToolingTT Number to 1
ToolingTT Number = 0

Figure 32. Macroexpansion of SocharisKB

7.3. *Information-Processing Model of Re-Designed Socharis*

In spite of the telltale nature of the macroexpansion, it does not show all the information necessary for understanding the problem-solving behavior of Re-SocharisKB. Information-Processing Layer is specifically designed to show how information is used and processed by every node of the functional model therefore exposing the problem-solving behavior of an I-KBS. Port managers are used to access the Information-Processing Model. Through the mechanism of port managers, the designer and the user can access individual nodes, their preconditions, mappings, and specific control.

Organization of port managers for operationSelector, RTM, and Labor (Appendix B) nodes of PSH could illustrate three distinctive examples of control structures that are used in Socharis.

Port manager for `operationSelector` indicates input and output variables of the node as well as existential conditions and default values. This port manager does not impose any mappings and dynamic control. That means that this node will activate the correspondent problem-solver if the existential predicate is satisfied. This node is directly and unconditionally mapped to Booster node that makes it an example of the Rigid Control architecture where the communication channels as well as the order of execution are being defined before the system has run.

If we look at the `RTM` node, then besides the information about the input/output variables and mapping we can see that the node is being controlled by that variable `rtmBoostCtrl` which is mapped from `boosterResult`. `rtmBoostCtrl` has a type of HC control, which tells us that the node will be activated when the value of `boosterResult` will contain value `'Resin Transfer Molding'` designated in `control` field of the `rtmBoostCtrl`. The similar control structure could be found at every node responsible for running individual technology refiners, which directly maps to the domain-derived strategy for identifying perspective technologies. On the other hand, the problem-solving architecture of the part of Re-SocharisKB responsible for technologies' instantiations is an example of the blackboard architecture. Blackboard is being modeled via broadcasting variables' values through the communication channels. Each refiner node (a knowledge source in blackboard terms) filters the necessary values through the mapping and activates the respective problem-solver if the value of the control variable *triggers* it.

Macroexpansion of Socharis does not show that the nodes responsible for the estimation of a particular merit have to be executed as many times as a number of alternatives generated by Re-SocharisKB. However, if we take a look at the Information-Processing Model of SocharisKB (Appendix B) at one of the estimator nodes (e.g. `Labor`) we notice six control variables, each responsible for manipulating the node's activation in case of a particular problem-solving context. Lets' take a closer look at the control variable `RTMCtrl`. This variable is being mapped from the `RTM Port` from the `RTMResult` output variable and contains the result (likely to be multiple result) of the `RTM` refiner. In the case when the control is being handed to this node by the `RTM` node, the `Labor` node effects the mapping of the variables using `RTMPort@1` mapping and iterates the Labor node for all values of the `RTMResult`. This is a clear example of Semi-Rigid Control Architecture, where one defines the communication channels but activation of the node and number of node's iteration are determined only during the run time.

The above instances demonstrated the ability of the S-Force to re-produce three different kinds of control architectures (Section 3.1.1) using mechanism of dynamic control provided by port managers within the Information-Processing Model.

7.4. Ontology, User Interface, Pre-, and Post-processing

S-Force does not cover problems associated with the development of domain ontology, user interface, and algorithmic pre- and post-processing.

The development of domain ontology in every particular case is a process where the KBS developer might use a number of available shells (e.g. Ontolingua server tool (Fikes 1997), ProtégéWin (Eriksson, Fergerson et al. 1999)) and then parse the result in to the knowledge structure convenient for building I-KBS. For Socharis, we developed a Manufacturing Ontology Editor (Martinez, Lukibanov et al. 1998; Martinez, Lukibanov et al. 1999) that allowed creating and modifying ontology of manufacturing with polymer composite materials. This manufacturing ontology is a four-level deep class–subclass hierarchy (Category → Class → Attribute → Value) that describes a part as well as manufacturing technology used to produce it.

Alternatively, I used ProtégéWin tool to create a similar ontology and output it into the balanced parenthesis list that could be easily parsed to re-create an existing ontology. However, ProtégéWin goes one step further by allowing creating interfaces that help produce instances using the developed ontology. These instances then can be sent to an external application for the processing. Unfortunately, as recognized by the team of creators of ProtégéWin, development of a good interface is not a problem that could be undertaken by a general-purpose tool. The interfaces, generated by ProtégéWin are fair for the development purposes; however, they are totally unacceptable for the end-user. Similar reasons motivated my research to forego features of S-Force that would allow developing the end-user interface for a front-end application and concentrate on the development of an interface suitable for the developmental purposes.

A particular domain- and application-dependent user interface can be developed in a Smalltalk environment using back-end of the S-Force that contains a list of pointers to all variables participating in the problem-solving process. It is also relatively straightforward to write CORBA or ActiveX interface definitions for the back-end of a developed I-KBS to use it as a server accessible by external applications.

Algorithmic pre- and post-processing in a KBS are usually operations that prepare input data for the I-KBS and transforming the output data for user's understanding. If some of these procedures are knowledge intensive they can be imported as parts of the I-KBS. However, for the most part of these procedures are highly domain, application, and user dependent hence, similar to end user interface and are left out of the scope of the described research.

7.5. Discussion

Chapters 6 and 7 demonstrated the possibility of implementation of I-KBS using the framework that supports the described approach. The approach is based on the Function-Based Reasoning theory, augmented with new functional ontology and addition of the Information-Processing layer that allows not only explicit assigning of input and output information for every node of Functional Model but also defining mapping, advanced preconditions, default values, and dynamic control.

Section 7.3 demonstrated that the problem-solving architecture of Re-SocharisKB leverages Rigid Control, Flexible Control, and Semi-Rigid Control architectures at different points of a problem-solving process depending on the available information, which allows close fitting of the I-KBS problem-solving architecture to the domain engineer problem-solving strategy and tactics.

8. CONCLUSION

This dissertation addresses two major problem areas consequently divided into the number of sub-goals:

1. In the knowledge-based systems area (KBS): development of methodology and implementing the supporting software framework for constructing integrated knowledge-based systems.

KBS-1. Augmentation of the Function-Based Reasoning methodology that allows ***functional decomposition*** of a real world problem using as building blocks a limited set of *types* of unitary problem-solvers.

KBS-2. Addition of an Information-Processing layer to the functional modeling methodology, which enables an explicit assignment of information and control flow through the Functional Model.

KBS-3. Addition of the capability of describing dynamic device control over the parts of the Functional Model of the I-KBS, therefore enabling implementation of arbitrary distributed control structures and problem-solving units within the I-KBS.

KBS-4. The development of a shell which supports building I-KBSs in the described framework.

2. In the area of composite materials (CMD): re-design of the knowledge-based core of the system Socharis whose goal was to generate a family of applicable conceptual manufacturing plans for assemblies made of polymer composite materials.

CMD-1. Possibility of on-the-fly change in the system's architecture, the architecture of its components, and the knowledge content. Re-designed in the developed shell Socharis will provide graphical interfaces to examine every part of the system and to modify them if necessary. Modification cannot only affect the knowledge encoded in the system, but also the control flow that governs the order of the problem-solvers' activation.

CMD-2. Explicit definition of the information and control flow between the components of the system that lets the user to understand the problem-solving strategy of Socharis. This, in turn, may be used as an educational instrument for teaching manufacturing in polymer composites.

This chapter reports on the accomplishments and the contribution of the dissertation research specifically covering the listed above sub-goals.

8.1. Contributions to the Knowledge-Based Systems Field

8.1.1. Preamble

The major KBS goal of this dissertation research was to develop a methodology for constructing integrated knowledge-based systems. The foundation of the proposed approach is the Function-Based Reasoning theory. A number of qualities of the FR methodology such as explicit causal network, hierarchical device decomposition, and the ability to arbitrary vary the level of granularity of the decomposition prompted this researcher to consider it as the methodology for description of the integrated problem-solving architecture. In addition to that, application of the FR techniques to component-based software engineering described in (Liver and Allemang 1995) justified the use of this methodology to describe integrated knowledge-based systems However, the function-based reasoning theory required certain modifications to accommodate the specifics of the modeled subject: an integrated knowledge based system.

The main obstacles that prohibit direct application of the FR methodology to decomposition and modeling I-KBSs could be listed as follows (Section 3.2):

- The lack of an adequate ontology capable of describing the functionality of problem-solving units.
- Inability to distinguish between different parts of the substance that is being passed through the functional model, i.e. treating it as inseparable *slug* of matter.
- Weakness of the precondition clause that made it very difficult to state complex conditions on the activation of parts of the functional model, which adds to the absence of complex control structures within the functional model.

The additions to the current FR framework which have been proposed in this dissertation augment traditional FR and particularly to its implementation in the Intelligent Systems Laboratory's legacy software package attacks these issues by:

- Introducing changes to the functional ontology, including functional description of bottom-level devices/problem-solvers (Section 3.2.1)
- Adding the notion of port managers that allow description of the information-processing layer that directs information and control flow in an I-KBS, and (Section 3.2.2)

- Using port managers as the distributed control units that expand the notion of pre-condition in the traditional FR (Section 3.2.2)

These augmentations allow using the FR techniques to describe I-KBS emphasizing task-subtask decomposition and the function of each separate or integrated component of the functional model.

Below I summarize the specific accomplishments in the areas listed above.

8.1.2. Functional Ontology

The modification to the traditional functional ontology (Section 3.2.1) includes description of functionality of GT-based problem-solving types. This ontology is best leveraged when the designer attempts to approach a new problem, which does not have a clear task – sub-task decomposition. In this case, the functional ontology guides the designer in identifying the role of subordinate units as one of the available problem-solving types. Moreover, it provides the designer with predetermined configuration of input and output data, specific for each type of problem-solver. This affects the process of building I-KBS since types of dynamic controls explicitly depend on the type and structure of this information. The result of multiple routine designer is a collection of sets of parameters that would satisfy the described in the inputs requirements the reasonable operation that could be performed on this result is consecutive look-up. On the other hand, the result of hierarchical classifier is the list of predetermined names with correspondent confidence value: “matched”, “neutral”, “against”, etc. The reasonable operation on this rather output structure is to compare it with specified *a priori* test expression. Therefore, the structure of the available information may hint the KBS designer to choose one problem-solving unit over the other.

In the case of explicit or known task breakdown, the functional ontology of problem solving does not play its guiding role in task-decomposition process. In turn the designer is faced with the task of finding an available off-the-shelf problem-solving unit capable of tackling each identified sub-task. This problem spurs a challenging branch in the research in the theoretical and practical FR: a problem of concise and sufficient definition of a device and its functionality. Such definition or a template should enable automatic or semiautomatic retrieval of a device from the library of devices. The research toward creating such a universal template for a device description is currently underway in Stanford University's Knowledge Systems Laboratory and Ohio State University's Laboratory for Artificial Intelligence Research.

8.1.3. Information-Processing Layer and Port Managers

The most important innovation in the research reported is the introduction of an information-processing layer to the functional model (Section 3.2.2). This layer allows treating information flowing through the functional model as a flow of divisible streams of data and control, rather than as an inseparable *slug* of substance: a representation common for the traditional FR approach. Information-processing layer is accessible to the I-KBS developer through the mechanism of port managers, introduced in Section 3.2.2. Port managers govern the inputs and outputs of every device/problem-solver in the functional model. They are able to perform testing of preconditions, checking assertions, assigning default values to the ports (input and output variables), map ports form one device to another, control device's activation, etc. Using port managers, information flow through the functional model could be separated, unified, mapped from one module to another, and re-directed.

Association of port managers with every device/problem-solver of the functional model allows associating of distributed control units with every part of the functional model. These control units enhance the functionality of their predecessors – precondition clauses – but also enable the development architectures with complex control structures including iteration and boosting.

8.1.4. S-Force: a Shell for Developing I-KBSs

Following the proposed changes to the FR framework, the shell for constructing I-KBSs – S-Force (Chapters 6 and 7) – was implemented on the basis of legacy software package (GT ITS) suitable for the development of stand-alone or hard-wired GT-based problem-solvers. The legacy of GT ITS was the main reason for using the GT-based problem-solvers as base-line problem-solving units. Another consideration in favor of exploiting GT-based problem-solvers was the fact that they have been successfully applied to the development of the number of KBSs in the Intelligent Systems Laboratory at Michigan State University over the last decade.

It is important to understand that in spite of use of GT-Based problem-solvers in S-Force, the methodology developed in this dissertation does not impose any restriction on the type of the problem-solving modules that serve as basic building blocks. The main criterion is to be able to look at the problem-solving unit as at *a black box* with clearly identifiable input, output, and functionality.

S-Force provides the developer with the graphical environment that allows on the fly designing and building I-KBSs by defining and systematically describing three layers of an I-KBS: the Problem-Solving Hierarchy, the Functional Model, and the Information-Processing Model (Chapter 6). Development of an I-KBS is finished when each leaf of the Problem-Solving Hierarchy is associated with a specific problem-solving unit. S-Force allows re-use previously built in the GT ITS framework units as parts of an I-KBS as well as supporting the design of the problem-solving units from scratch.

8.1.5. Practical Application of S-Force

The practicality of S-Force was studied on several examples including re-designing knowledge-based core of *Socharis* system (Chapter 7 Appendix B) and building of an exemplary tutorial system (Chapter 6, Appendix A). S-Force's features are proved adequate for designing and modeling integrated knowledge-based systems. In fact, it took approximately four hours to re-design KBS core of Socharis consisted of thirty cooperating entities, if all problem-solving units were available in advance.

S-Force gives access to the knowledge level architecture of I-KBSs designed in it by the means of graphical user interfaces that disclose every part of the problem-solving architecture, functionality and internal structure of the constituent parts of an I-KBS, and an I-KBSs information-processing layer.

8.1.6. Summary and Venues of Future Research

There are some implementation-level shortcomings of S-Force that should be addressed in future research. Current implementation of S-Force does not take advantage of possibility to distribute the problem-solving modules across the network. However, it is possible to locate modules on the different machines and call them asynchronously using one of available protocols and broker architectures. One thing that has to be kept in mind is the synchronization of results of multiple problem-solvers working on the same problem.

One of the features of port managers unexplored in the current implementation is the possibility to post-process results of the run of the node. The addition of the post-processing capability can considerably increase the power of the shell to develop more complex problem-solving architectures.

To summarize the features and the capabilities of the reported in this dissertation research towards developing methodology and the tool for constricting I-KBSs Table 8 below compares it with the existing methodologies and shells for building I-KBSs. The comparison is done along the lines presented in the Section 2.2.5 that underscores the explicitness of the knowledge level architecture, implemented control strategies, and the existence of the shell supporting the methodology.

	Explicity of KLA (1 - 5)	Global control strategy	Local control strategy	Ability to handle multiple control structures	Shell
BB1, BBK	1	FC	SRC	No	Yes
PROTÉGÉ-II	3	FC	All	Somewhat (programming required)	Somewhat
Generic Task Toolset	5	RC	RC	No (programming is needed)	Yes
TIPS	4	SRC	RC	No	No
GT SOAR	3	RC	SRC	No	No
CommonKADS	5	SRC	RC	Some	Several
VITAL	5	RC, some SRC	RC	No	Yes
CoMoMAS	2	FC	SRC	No	No
Conventional FR	5	SRC	RC	No	Yes
Augmented FR **S-Force**	**5**	**RC, SRC, FC**	**RC, SRC, FC**	**Yes**	**Yes**

Table 8. Comparison of the S-Force with other methodologies and shells.

The above discussion leads to the conclusion that the goals KBS-1 through KBS-4 were achieved in this dissertation. However, the research opened new research paths that could be investigated in the future:

- Development of the device/problem-solver definition template
- Augmentation S-Force to allow development and modeling distributed problem-solving architectures
- Improvement of port-manager's functionality with the addition of post-processing.

8.2. Contributions to the Application Domain

Redesigning the knowledge-based core of Socharis using S-Force accomplished the stated in the Introduction to the dissertation application domain goals CMD-1 – CMD-2 by enabling the direct access to the control structures governing problem-solvers' activation process. These control structures were previously hidden in the Smalltalk code that disallowed effortless alteration and augmentation of Socharis. Use of S-Force enabled the designer to directly access the decomposition of the problem of the conceptual manufacturing planning as it is perceived be a domain specialist. The major benefit of a re-designed system is that the control and information flows through a problem-solving process and are directly exposed to the designer through the graphical user interface. Such open architecture enables the designer making changes in the problem-solving architecture in the case of necessity to adapt future versions of Socharis to new deployment sites.

Another important contribution of work on Socharis and re-designed Socharis in the application domain area is the hypothesis introduced in (Lukibanov, Martinez et. al.. 2000), which asserts that a high-level problem-solving architecture developed for Socharis is a high-level problem-solving architecture of the conceptual process planning task in general. Given this, the problem-solving skeleton of the redesigned Socharis can serve as a backbone for the process-planning intelligent systems in domains others than polymer composite materials. It would be challenging and engaging to take on the problem of manufacturing process planning in the area of metal- or woodworking using the Socharis's problem-solving architecture implemented in S-Force as a starting point.

8.3. Concluding Remarks

By taking the hypothesis introduced in (Lukibanov, Martinez et. al.. 2000) – that supposes that problem-solving architecture of Socharis is, in fact, the problem-solving architecture of the conceptual manufacturing planning – one step further it is reasonable to suggest that there exists a number of problem-solving architectures commonly used in the problem-solving activity. The PSMs of Second-Generation Knowledge-Based Systems (Section 3.2) serve as templates for building solutions for homogeneous tasks, tasks that require a single method. In contrast, integrated problem-solving architectures can be used to approach heterogeneous problems that are impossible to solve by application of a single problem-solving method but rather by their combinations.

The challenge is in identifying such architectures. Extracting and categorizing problem-solving architectures could be done only after analyses of a great number of different tasks and approaches or by generalizing on a successful implementation in a particular area. One of the sources where such an analysis is possible is the Domain Specific Software Architectures (DSSA) research (Mettala 1992) (Section 3.4.4). Five-year long DARPA sponsored research was aimed to identify common software architecture in six different areas:

1. Avionics Navigation
2. Guidance and Flight Director

3. Command and Control
4. Distributed Intelligent Control and Management (DICAM) for Vehicle Management
5. Intelligent Guidance, Navigation and Control
6. Hybrid Control, and Prototyping Technology

Research was distributed among several contractors: IBM Federal Sector Division, GTE Federal Systems, Tecknowledge Federal Systems, Honeywell Systems and Research Center, ORA Corporation, and TPW. This research resulted in the set of methodic and tools that defined specification, design, and validation of software systems specific for each domain. One the accomplishments of the DSSA project was the conclusion that mature software architecture should possess three basic elements:

- well-defined notation for capturing architectures
- well-defined methods for producing and analyzing formal models from specification, captured in notation
- a well-defined method for producing implementation from a specification captured in the notation.

However, no study was performed to identify similarity(s) between the developed software architectures and processes. The reason for this is that at that time, the researchers lacked the necessary methodology to describe the software architecture and processes in the unified way. Nowadays, Rational Unified Process (RUP) developed as a result of research in Capability Maturity Model, Personal Software Process, and Object Modeling Techniques can help in capturing organizational flow and architectural details. Analyzing RUP models, it might be possible to recognize similarities in the overall organizational process as well as software architecture. However, knowledge-intensive processing is beyond the RUP covered area since it does not support knowledge modeling techniques.

The information presented in this dissertation methodology provides means to model such knowledge-intensive tasks by describing problem-solving architectures that could be compared along three axis: problem decomposition, functional decomposition, and information processing. This feature enables comparison between a difference modeled in S-Force systems. Such an analysis followed by generalization could lead to definition of a number of useful problem-solving architectures that would serve as backbone for the next generation of knowledge-based systems consisting of multiple cooperating entities and featuring accessible knowledge level.

APPENDICES

APPENDIX A

KNOWLEDGE STRUCTURE OF SOCHARIS*

Device: Example Socharis
Function: Example Socharis Fn
Beahvior: Example Socharis Beh
Port Manager: Example SocharisPort
All variables:

Example Socharis Number	input output
Example Socharis Result	output
Technology Selector Number	input
Fiber_Architecture	input
size	input
Wall_Thickness	input
Aspect_Ratio	input
Shape	input
partMaterial	input
Layup Refiner Number	input
Tolerances	input
SurfaceQuality	input
GeometricalComplexity	input
functional requirements	input
Refiner Number	input
Layup Refiner Number	input
RTM Refiner Number	input
Production	input
Tooling Cost Estimator Number	input
Part TT Estimator Number	input
fiberList	input

Qualitative predicates:
Quantitative predicates:
Existential predicates:
Default values:
Mappings:
Controls:

Device: Estimator
Function: EstimatorF
Beahvior: EstimatorB

Port Manager: EstimatorPort
All variables:

Estimator Number	input output
EstimatorResult	output
depth	input
Tooling Cost Estimator Number	input
Part TT Estimator Number	input

Qualitative predicates:
Quantitative predicates:
Existential predicates:
Default values:
Mappings:
Example SocharisPort@1

Estimator Number	?
EstimatorResult	?
depth	Wall_Thickness

Controls:

Device: Refiner

Function: RefinerFn

Beahvior: RefinerBh

Port Manager: RefinerPort

All variables:

Refiner Number	input output
RefinerResult	output
Tolerances	input
SurfaceQuality	input
size	input
resinList	input
Production	input
GeometricalComplexity	input
Fiber_Architecture	input
RTM Refiner Number	input
depth	input
functional requirements	input
Layup Refiner Number	input
fiberList	input

Qualitative predicates:

Quantitative predicates:

Existential predicates:

Production	?
resinList	?
Fiber_Architecture	*
depth	*
Refiner Number	?
GeometricalComplexity	*
functional requirements	*
Layup Refiner Number	?
Tolerances	*
SurfaceQuality	*
RTM Refiner Number	?
size	*

Default values:

Mappings:

Example SocharisPort@1

Refiner Number	?
RefinerResult	?
Tolerances	Tolerances
SurfaceQuality	SurfaceQuality
size	size
resinList	partMaterial
Production	Production

GeometricalComplexity	GeometricalComplexity
Fiber_Architecture	Fiber_Architecture
RTM Refiner Number	?
depth	Wall_Thickness
functional requirements	functional requirements
Layup Refiner Number	?

Controls:

Device: Technology Selector
Function: SelectTechnology
Beahvior: stBeh
Port Manager: Technology SelectorPort
All variables:

Technology Selector Number	input output
Aspect_Ratio	input
Fiber_Architecture	input
partMaterial	input
Shape	input
size	input
Wall_Thickness	input
TSResult	output

Qualitative predicates:
partMaterial ~= unknown
Shape ~= unknown
Quantitative predicates:
Existential predicates:

partMaterial	*
Wall_Thickness	*
Aspect_Ratio	*
Shape	*
Technology Selector Number	?
Fiber_Architecture	*
size	*

Default values:
Mappings:
Example SocharisPort@1

Technology Selector Number	?
Aspect_Ratio	Aspect_Ratio
Fiber_Architecture	Fiber_Architecture
partMaterial	partMaterial
Shape	Shape
size	size
Wall_Thickness	Wall_Thickness
TSResult	?

Controls:

Device: Part TT Estimator
Function: Estimate PTT
Beahvior: epttBeh
Port Manager: Part TT EstimatorPort
All variables:

Part TT Estimator Number	input output
CMtemperature	input
curing type	input
FWtemperature	input
IMtemperature	input
manTechList	input
Part turnaround time	output
resin prepreg/wet	input
resinList	input
RTMHeating method	input
size	input
Wall_Thickness	input
ptte Layup Ctrl	input
ptte RTM Ctrl	input
pteResult	output

Qualitative predicates:
Quantitative predicates:
Existential predicates:
Default values:
Mappings:
Layup RefinerPort@1\

Part TT Estimator Number	?
CMtemperature	?
curing type	curing type
FWtemperature	?
IMtemperature	?
manTechList	manTechList
Part turnaround time	?
resin prepreg/wet	resin prepreg/wet
resinList	resinList
RTMHeating method	?
size	size
Wall_Thickness	depth
ptte Result	?
ptte Layup Ctrl	Layup Refiner Result
ptte RTM Ctrl	?

EstimatorPort@1

Part TT Estimator Number	?
CMtemperature	?
curing type	?

FWtemperature	?
IMtemperature	?
manTechList	?
Part turnaround time	?
resin prepreg/wet	?
resinList	?
RTMHeating method	?
size	?
Wall_Thickness	depth
ptte Result	?
ptte Layup Ctrl	?
ptte RTM Ctrl	?

RTM RefinerPort@1

Part TT Estimator Number	?
CMtemperature	?
curing type	Curing type
FWtemperature	?
IMtemperature	?
manTechList	manTechList
Part turnaround time	?
resin prepreg/wet	?
resinList	resinList
RTMHeating method	Heating method
size	size
Wall_Thickness	?
ptte Result	?
ptte Layup Ctrl	?
ptte RTM Ctrl	RTM RefinerResult

Controls:

ptte Layup Ctrl

type	rdInputControl
control name	ptte Layup Ctrl
control	Layup Refiner Result

ptte RTM Ctrl

type	rdInputControl
control name	ptte RTM Ctrl
control	RTM RefinerResult

Device: Tooling Cost Estimator
Function: Estimate Toolig Cost
Beahvior: etcBeh
Port Tooling Cost EstimatorPort
All variables:

Tooling Cost Estimator Number	input output
manTechList	input
material cost	
Tool complexity	
Tooling cost	
toolingList	input
tce Layup Ctrl	input
tce RTm Ctrl	input
tce Result	output

Qualitative predicates:
toolingList ~= unknown
Quantitative predicates:
Existential predicates:

manTechList	*
tce RTm Ctrl	?
Tooling Cost Estimator Number	?
tce Layup Ctrl	?
toolingList	*

Default values:
Mappings:
Layup RefinerPort@1

Tooling Cost Estimator Number	?
manTechList	manTechList
material cost	?
Tool complexity	Tool complexity
Tooling cost	?
toolingList	toolingList
tce Result	?
tce Layup Ctrl	Layup Refiner Result
tce RTm Ctrl	?

RTM RefinerPort@1

Tooling Cost Estimator Number	?
manTechList	manTechList
material cost	?
Tool complexity	Tool complexity
Tooling cost	?
toolingList	ToolingMaterial
tce Result	?
tce Layup Ctrl	?
tce RTm Ctrl	RTM RefinerResult

Controls:

tce RTm Ctrl

type	rdInputControl
control name	tce RTm Ctrl
control	RTM RefinerResult

tce Layup Ctrl

type	rdInputControl
control name	tce Layup Ctrl
control	Layup Refiner Result

Device: Layup Refiner
Function: Refine Layup
Beahvior: rlBeh
Port Manager: Layup RefinerPort
All variables:

Layup Refiner Number	input output
Layup Refiner Result	output
Layup Refiner Ctrl	input
curing type	output
depth	input
fiberList	input
functional requirements	input
GeometricalComplexity	input
labor	output
postcuring	output
pressure	output
production	input
resin prepreg/wet	output
resinList	input
size	input
SurfaceQuality	input
temperature	output
Tolerances	input
Tool complexity	output
toolingList	output
manTechList	output

Qualitative predicates:
Quantitative predicates:
Existential predicates:

resinList	*
depth	*
Layup Refiner Ctrl	?
functional requirements	*
Tolerances	*
SurfaceQuality	*
GeometricalComplexity	*
production	*
fiberList	*
Layup Refiner Number	?
size	*

Default values:

resinList	DGEBA

Mappings:
Technology SelectorPort@1

Layup Refiner Number	?
Layup Refiner Result	?
Layup Refiner Ctrl	TSResult
curing type	?
depth	?
fiberList	?
functional requirements	?
GeometricalComplexity	?
labor	?
postcuring	?
pressure	?
production	?
resin prepreg/wet	?
resinList	?
runSystem	?
size	?
SurfaceQuality	?
temperature	?
Tolerances	?
Tool complexity	?
toolingList	?
manTechList	?

RefinerPort@1

Layup Refiner Number	?
Layup Refiner Result	?
Layup Refiner Ctrl	?
curing type	?
depth	?
fiberList	fiberList
functional requirements	functional requirements
GeometricalComplexity	GeometricalComplexity
labor	?
postcuring	?
pressure	?
production	Production
resin prepreg/wet	?
resinList	resinList
runSystem	?
size	size
SurfaceQuality	SurfaceQuality
temperature	?
Tolerances	Tolerances
Tool complexity	?
toolingList	?

manTechList	?

Controls:

Layup Refiner Ctrl

type	hcInputControl
control name	Layup Refiner Ctrl
output	manTechList
control	%Layup

Device: RTM Refiner
Function: Refine RTM
Beahvior: rrtmBeh
Port Manager: RTM RefinerPort
All variables:

RTM Refiner Number	input output
RTM RefinerResult	output
RTM Refiner Ctrl	input
Curing temperature	output
Curing time	output
Curing type	output
FiberFormingMethod	output
Fiber_Architecture	input
GeometricalComplexity	input
Heating method	output
labor	output
Postcuring Required	output
Production	input
resinList	input
runSystem	
size	input
SurfaceQuality	input
Tolerances	input
Tool complexity	output
ToolingMaterial	output
manTechList	output

Qualitative predicates:
Quantitative predicates:
Existential predicates:

RTM Refiner Ctrl	?
RTM Refiner Number	?
size	*
Tolerances	*
Production	*
resinList	*
SurfaceQuality	*
GeometricalComplexity	*
Fiber_Architecture	*

Default values:
Mappings:
Technology SelectorPort@1

RTM Refiner Number	?
RTM RefinerResult	?
RTM Refiner Ctrl	TSResult

Curing temperature	?
Curing time	?
Curing type	?
FiberFormingMethod	?
Fiber_Architecture	?
GeometricalComplexity	?
Heating method	?
labor	?
Postcuring Required	?
Production	?
resinList	?
runSystem	?
size	?
Tool complexity	?
ToolingMaterial	?

RefinerPort@1

RTM Refiner Number	?
RTM RefinerResult	?
RTM Refiner Ctrl	?
Curing temperature	?
Curing time	?
Curing type	?
FiberFormingMethod	?
Fiber_Architecture	Fiber_Architecture
GeometricalComplexity	GeometricalComplexity
Heating method	?
labor	?
Postcuring Required	?
Production	Production
resinList	resinList
runSystem	?
size	size
SurfaceQuality	SurfaceQuality
Tolerances	Tolerances
Tool complexity	?
ToolingMaterial	?

Controls:

RTM Refiner Ctrl

type	hcInputControl
control name	RTM Refiner Ctrl
output	manTechList
control	%RTM

APPENDIX B

KNOWLEDGE STRUCTURE OF RE-ENGINEERED SOCHARIS

Device: Socharis
Function: SocharisFn
Behavior: SocharisBh
Port Manager: SocharisPort
All variables:

Socharis Number	input output
socharisResult	output
Selector Number	input
operationSelector Number	input
Aspect_Ratio	input
Fiber_Architecture	input
partMaterial	input
Shape	input
size	input
Wall_Thickness	input
matSel Number	input
GeometricalComplexity	input
production	input
Compression Molder Number	input
SurfaceQuality	input
Tolerances	input
Layup Number	input
fiberList	input
functional requirements	input
RTM Number	input
Resin Infusion Number	input
FilamentWinding Number	input
inserts	input

Qualitative predicates:
Quantitative predicates:
Existential predicates:
Default values:
Mappings:
Controls:

Device: Estimator
Function: Estimation
Behavior: EstimatorBeh
Port Manager: EstimatorPort
All variables:

Estimator Number	input output
estimatorResult	output

Qualitative predicates:
Quantitative predicates:
Existential predicates:
Default values:
Mappings:
Controls:

Device: Refiner

Function: refinerFn

Behavior: refinerBh

Port Manager: RefinerPort

All variables:

Refiner Number	input output
refinerResult	output
GeometricalComplexity	input
production	input
Compression Molder Number	input
fiberArchList	input
resinList	input
SurfaceQuality	input
Tolerances	input
Layup Number	input
fiberList	input
depth	
size	input
functional requirements	input
RTM Number	input
Resin Infusion Number	input
FilamentWinding Number	input
inserts	input

Qualitative predicates:

Quantitative predicates:

Existential predicates:

Default values:

Mappings:

SocharisPort@1

Refiner Number	?
refinerResult	?
GeometricalComplexity	GeometricalComplexity
production	production
Compression Molder Number	?
fiberArchList	Fiber_Architecture
resinList	partMaterial
SurfaceQuality	SurfaceQuality
Tolerances	Tolerances
Layup Number	?
fiberList	fiberList
depth	Wall_Thickness
size	size
functional requirements	functional requirements

Controls:

Device: Selector
Function: SelectorFn
Behavior: SelectorBh
Port Manager: SelectorPort
All variables:

Selector Number	input output
selectorResult	output
operationSelector Number	input
Aspect_Ratio	input
Fiber_Architecture	input
partMaterial	input
Shape	input
size	input
Wall_Thickness	input
matSel Number	input

Qualitative predicates:
Quantitative predicates:
Existential predicates:
Default values:
Mappings:
SocharisPort@1

Selector Number	?
selectorResult	?
operationSelector Number	?
Aspect_Ratio	?
Fiber_Architecture	?
partMaterial	?
Shape	?
size	?
Wall_Thickness	?
matSel Number	?

Controls:

Device: GeomRep
Function: grFun
Behavior: grBeh
Port Manager: GeomRepPort
All variables:

GeomRep Number	input output
GeometricalComplexity	input
Geom_Rep	input
manTechList	input
resin prepreg/wet	input
GeomRepeatRslt	output
layupCtrl	input
RTMCtrl	input
cmCtrl	input
imCtrl	input
fwCtrl	input
riCtrl	input

Qualitative predicates:
Quantitative predicates:
Existential predicates:
Default values:
Mappings:
LayupPort@1

GeomRep Number	?
GeometricalComplexity	GeometricalComplexity
Geom_Rep	?
manTechList	manTechList
resin prepreg/wet	resin prepreg/wet
GeomRepeatRslt	?
layupCtrl	lyupRslt
RTMCtrl	?
cmCtrl	?
imCtrl	?
fwCtrl	?
riCtrl	?

FilamentWindingPort@1

GeomRep Number	?
GeometricalComplexity	GeometricalComplexity
Geom_Rep	?
manTechList	manTechList
resin prepreg/wet	?
GeomRepeatRslt	?
layupCtrl	?
RTMCtrl	?
cmCtrl	?

imCtrl	?
fwCtrl	fwResult
riCtrl	?

Compression MolderPort@1

GeomRep Number	?
GeometricalComplexity	GeometricalComplexity
Geom_Rep	?
manTechList	manTechList
resin prepreg/wet	?
GeomRepeatRslt	?
layupCtrl	?
RTMCtrl	?
cmCtrl	cmResult
imCtrl	?
fwCtrl	?
riCtrl	?

RTMPort@1

GeomRep Number	?
GeometricalComplexity	GeometricalComplexity
Geom_Rep	?
manTechList	manTechList
resin prepreg/wet	?
GeomRepeatRslt	?
layupCtrl	?
RTMCtrl	RTMResult
cmCtrl	?
imCtrl	?
fwCtrl	?
riCtrl	?

Resin InfusionPort@1

GeomRep Number	?
GeometricalComplexity	GeometricalComplexity
Geom_Rep	?
manTechList	manTechList
resin prepreg/wet	?
GeomRepeatRslt	?
layupCtrl	?
RTMCtrl	?
cmCtrl	?
imCtrl	?
fwCtrl	?
riCtrl	riResult

InjectionMoldingPort@1

GeomRep Number	?
GeometricalComplexity	GeometricalComplexity

Geom_Rep	?
manTechList	manTechList
resin prepreg/wet	?
GeomRepeatRslt	?
layupCtrl	?
RTMCtrl	?
cmCtrl	?
imCtrl	imResult
fwCtrl	?
riCtrl	?

Controls:

imCtrl

type	rdInputControl
control name	imCtrl
control	imResult

cmCtrl

type	rdInputControl
control name	cmCtrl
control	cmResult

riCtrl

type	rdInputControl
control name	riCtrl
control	manTechList

RTMCtrl

type	rdInputControl
control name	RTMCtrl
control	RTMResult

layupCtrl

type	rdInputControl
control name	layupCtrl
control	lyupRslt

fwCtrl

type	rdInputControl
control name	fwCtrl
control	fwResult

Device: Labor

Function: laborFn

Behavior: laborBeh

Port Manager: LaborPort

All variables:

Labor Number	input output
labor	input
Labor estimation	output
manTechList	input
layupCtrl	input
laborResult	output
RTMCtrl	input
cmCtrl	input
imCtrl	input
fwCtrl	input
riCtrl	

Qualitative predicates:

Quantitative predicates:

Existential predicates:

Default values:

Mappings:

LayupPort@1

Labor Number	?
labor	labor
Labor estimation	?
manTechList	manTechList
layupCtrl	lyupRslt
laborResult	?
RTMCtrl	?
cmCtrl	?
imCtrl	?
fwCtrl	?
riCtrl	?

FilamentWindingPort@1

Labor Number	?
labor	?
Labor estimation	?
manTechList	manTechList
layupCtrl	?
laborResult	?
RTMCtrl	?
cmCtrl	?
imCtrl	?
fwCtrl	fwResult
riCtrl	?

Compression MolderPort@1	
Labor Number	?
labor	labor
Labor estimation	?
manTechList	manTechList
layupCtrl	?
laborResult	?
RTMCtrl	?
cmCtrl	cmResult
imCtrl	?
fwCtrl	?
riCtrl	?

RTMPort@1

Labor Number	?
labor	labor
Labor estimation	?
manTechList	RTMResult
layupCtrl	?
laborResult	?
RTMCtrl	RTMResult
cmCtrl	?
imCtrl	?
fwCtrl	?
riCtrl	?

Resin InfusionPort@1

Labor Number	?
labor	labor
Labor estimation	?
manTechList	manTechList
layupCtrl	?
laborResult	?
RTMCtrl	?
cmCtrl	?
imCtrl	?
fwCtrl	?
riCtrl	riResult

InjectionMoldingPort@1

Labor Number	?
labor	?
Labor estimation	?
manTechList	manTechList
layupCtrl	?
laborResult	?
RTMCtrl	?
cmCtrl	?

imCtrl	imResult
fwCtrl	?
riCtrl	?

Controls:

cmCtrl

type	rdInputControl
control name	cmCtrl
control	cmResult

fwCtrl

type	rdInputControl
control name	fwCtrl
control	fwResult

RTMCtrl

type	rdInputControl
control name	RTMCtrl
control	RTMResult

riCtrl

type	rdInputControl
control name	riCtrl
control	riResult

imCtrl

type	rdInputControl
control name	imCtrl
control	imResult

layupCtrl

type	rdInputControl
control name	layupCtrl
control	lyupRslt

Device: MechProp
Function: mpFun
Behavior: mpBeh
Port Manager: MechPropPort
All variables:

MechProp Number	input output
MachanicalProperties	output
manTechList	input
resin prepreg/wet	input
mechPropResult	output
layupCtrl	input
RTMCtrl	input
imCtrl	input
cmCtrl	input
fwCtrl	input
riCtrl	input

Qualitative predicates:
Quantitative predicates:
Existential predicates:
Default values:
Mappings:
InjectionMoldingPort@1

MechProp Number	?
MachanicalProperties	?
manTechList	manTechList
resin prepreg/wet	?
mechPropResult	?
layupCtrl	?
RTMCtrl	?
imCtrl	imResult
cmCtrl	?
fwCtrl	?
riCtrl	?

FilamentWindingPort@1

MechProp Number	?
MachanicalProperties	?
manTechList	manTechList
resin prepreg/wet	?
mechPropResult	?
layupCtrl	?
RTMCtrl	?
imCtrl	?
cmCtrl	?
fwCtrl	fwResult
riCtrl	?

Compression MolderPort@1

MechProp Number	?
MachanicalProperties	?
manTechList	manTechList
resin prepreg/wet	?
mechPropResult	?
layupCtrl	?
RTMCtrl	?
imCtrl	?
cmCtrl	cmResult
fwCtrl	?
riCtrl	?

LayupPort@1

MechProp Number	?
MachanicalProperties	?
manTechList	manTechList
resin prepreg/wet	resin prepreg/wet
mechPropResult	?
layupCtrl	lyupRslt
RTMCtrl	?
imCtrl	?
cmCtrl	?
fwCtrl	?
riCtrl	?

RTMPort@1

MechProp Number	?
MachanicalProperties	?
manTechList	manTechList
resin prepreg/wet	?
mechPropResult	?
layupCtrl	?
RTMCtrl	RTMResult
imCtrl	?
cmCtrl	?
fwCtrl	?
riCtrl	?

Resin InfusionPort@1

MechProp Number	?
MachanicalProperties	?
manTechList	manTechList
resin prepreg/wet	?
mechPropResult	?
layupCtrl	?
RTMCtrl	?
imCtrl	?

cmCtrl	?
fwCtrl	?
riCtrl	riResult

Controls:

imCtrl

type	rdInputControl
control name	imCtrl
control	imResult

cmCtrl

type	rdInputControl
control name	cmCtrl
control	cmResult

RTMCtrl

type	rdInputControl
control name	RTMCtrl
control	RTMResult

layupCtrl

type	rdInputControl
control name	layupCtrl
control	lyupRslt

riCtrl

type	rdInputControl
control name	riCtrl
control	riResult

fwCtrl

type	rdInputControl
control name	fwCtrl
control	fwResult

Device: OperExposure
Function: oeFun
Behavior: oeBeh
Port Manager: OperExposurePort
All variables:

OperExposure Number	input output
Fiber	input
fiberList	input
manTechList	input
OperatorExposeValue	output
resin prepreg/wet	input
resinList	input
Solvents&Corrosives	
Toxins	output
oeResult	output
layupCtrl	input
RTMCtrl	input
cmCtrl	input
imCtrl	input
fwCtrl	input
riCtrl	input

Qualitative predicates:
Quantitative predicates:
Existential predicates:
Default values:
Mappings:
LayupPort@1

OperExposure Number	?
Fiber	?
fiberList	fiberList
manTechList	manTechList
OperatorExposeValue	?
resin prepreg/wet	resin prepreg/wet
resinList	resinList
Solvents&Corrosives	?
Toxins	?
oeResult	?
layupCtrl	lyupRslt
RTMCtrl	?
cmCtrl	?
imCtrl	?
fwCtrl	?
riCtrl	?

FilamentWindingPort@1

OperExposure Number	?

Fiber	?
fiberList	?
manTechList	manTechList
OperatorExposeValue	?
resin prepreg/wet	?
resinList	?
Solvents&Corrosives	?
Toxins	?
oeResult	?
layupCtrl	?
RTMCtrl	?
cmCtrl	?
imCtrl	?
fwCtrl	fwResult
riCtrl	?

Compression MolderPort@1

OperExposure Number	?
Fiber	?
fiberList	?
manTechList	manTechList
OperatorExposeValue	?
resin prepreg/wet	?
resinList	?
Solvents&Corrosives	?
Toxins	?
oeResult	?
layupCtrl	?
RTMCtrl	?
cmCtrl	cmResult
imCtrl	?
fwCtrl	?
riCtrl	?

RTMPort@1

OperExposure Number	?
Fiber	?
fiberList	?
manTechList	manTechList
OperatorExposeValue	?
resin prepreg/wet	?
resinList	?
Solvents&Corrosives	?
Toxins	?
oeResult	?
layupCtrl	?
RTMCtrl	RTMResult

cmCtrl	?
imCtrl	?
fwCtrl	?
riCtrl	?

Resin InfusionPort@1

OperExposure Number	?
Fiber	?
fiberList	?
manTechList	manTechList
OperatorExposeValue	?
resin prepreg/wet	?
resinList	?
Solvents&Corrosives	?
Toxins	?
oeResult	?
layupCtrl	?
RTMCtrl	?
cmCtrl	?
imCtrl	?
fwCtrl	?
riCtrl	riResult

InjectionMoldingPort@1

OperExposure Number	?
Fiber	?
fiberList	?
manTechList	manTechList
OperatorExposeValue	?
resin prepreg/wet	?
resinList	?
Solvents&Corrosives	?
Toxins	?
oeResult	?
layupCtrl	?
RTMCtrl	?
cmCtrl	?
imCtrl	imResult
fwCtrl	?
riCtrl	?

RefinerPort@1

OperExposure Number	?
Fiber	?
fiberList	fiberList
manTechList	?
OperatorExposeValue	?
resin prepreg/wet	?

resinList	resinList
Solvents&Corrosives	?
Toxins	?
oeResult	?
layupCtrl	?
RTMCtrl	?
cmCtrl	?
imCtrl	?
fwCtrl	?
riCtrl	?

Controls:

layupCtrl

type	rdInputControl
control name	layupCtrl
control	lyupRslt

riCtrl

type	rdInputControl
control name	riCtrl
control	riResult

fwCtrl

type	rdInputControl
control name	fwCtrl
control	fwResult
RTMCtrl	
type	rdInputControl
control name	RTMCtrl
control	RTMResult

cmCtrl

type	rdInputControl
control name	cmCtrl
control	cmResult

imCtrl

type	rdInputControl
control name	imCtrl
control	imResult

Device: partTT

Function: partTTfn

Behavior: partTTBeh

Port Manager: partTTPort

All variables:

partTT Number	input output
CMtemperature	input
curing type	input
FWtemperature	input
IMtemperature	input
Part turnaround time	
resin prepreg/wet	input
resinList	input
RTMHeating method	input
size	input
Wall_Thickness	input
manTechList	input
pttResult	output
lyupControl	input
rtmCtrl	input
imCtrl	input
cmCtrl	
fwCtrl	input
riCtrl	input

Qualitative predicates:

Quantitative predicates:

Existential predicates:

Default values:

Mappings:

LayupPort@1

partTT Number	?
CMtemperature	?
curing type	curing type
FWtemperature	?
IMtemperature	?
Part turnaround time	?
resin prepreg/wet	resin prepreg/wet
resinList	resinList
RTMHeating method	?
size	size
Wall_Thickness	depth
manTechList	manTechList
pttResult	?
lyupControl	lyupRslt
RTM Number	?

rtmCtrl	?
imCtrl	?
cmCtrl	?
fwCtrl	?
riCtrl	?

FilamentWindingPort@1

partTT Number	?
CMtemperature	?
curing type	?
FWtemperature	temperature
IMtemperature	?
Part turnaround time	?
resin prepreg/wet	delivery system
resinList	resinList
RTMHeating method	?
size	size
Wall_Thickness	?
manTechList	manTechList
pttResult	?
lyupControl	?
rtmCtrl	?
imCtrl	?
cmCtrl	?
fwCtrl	fwResult
riCtrl	?

Compression MolderPort@1

partTT Number	?
CMtemperature	temperature
curing type	?
FWtemperature	?
IMtemperature	?
Part turnaround time	?
resin prepreg/wet	?
resinList	?
RTMHeating method	?
size	?
Wall_Thickness	?
manTechList	manTechList
pttResult	?
lyupControl	?
rtmCtrl	?
imCtrl	?
cmCtrl	cmResult
fwCtrl	?
riCtrl	?

RTMPort@1

partTT Number	?
CMtemperature	?
curing type	?
FWtemperature	?
IMtemperature	?
Part turnaround time	?
resin prepreg/wet	?
resinList	?
RTMHeating method	Heating method
size	?
Wall_Thickness	?
manTechList	manTechList
pttResult	?
lyupControl	?
rtmCtrl	RTMResult
imCtrl	?
cmCtrl	?
fwCtrl	?
riCtrl	?

Resin InfusionPort@1

partTT Number	?
CMtemperature	?
curing type	curing type
FWtemperature	?
IMtemperature	?
Part turnaround time	?
resin prepreg/wet	?
resinList	?
RTMHeating method	?
size	?
Wall_Thickness	Wall_Thickness
manTechList	manTechList
pttResult	?
lyupControl	?
rtmCtrl	?
imCtrl	?
cmCtrl	?
fwCtrl	?
riCtrl	riResult

InjectionMoldingPort@1

partTT Number	?
CMtemperature	?
curing type	?
FWtemperature	?

IMtemperature	temperature
Part turnaround time	?
resin prepreg/wet	?
resinList	resinList
RTMHeating method	?
size	size
Wall_Thickness	?
manTechList	manTechList
pttResult	?
lyupControl	?
rtmCtrl	?
imCtrl	imResult
cmCtrl	?
fwCtrl	?
riCtrl	?

RefinerPort@1

partTT Number	?
CMtemperature	?
curing type	?
FWtemperature	?
IMtemperature	?
Part turnaround time	?
resin prepreg/wet	?
resinList	?
RTMHeating method	?
size	?
Wall_Thickness	depth
manTechList	?
pttResult	?
lyupControl	?
rtmCtrl	?
imCtrl	?
cmCtrl	?
fwCtrl	?
riCtrl	?

Controls:

riCtrl

type	rdInputControl
control name	riCtrl
control	riResult

rtmCtrl

type	rdInputControl
control name	rtmCtrl
control	RTMResult

imCtrl

type	rdInputControl
control name	imCtrl
control	imResult

fwCtrl

type	rdInputControl
control name	fwCtrl
control	fwResult

lyupControl

type	rdInputControl
control name	lyupControl
control	lyupRslt

cmCtrl

type	rdInputControl
control name	cmCtrl
control	cmResult

Device: SkillLevel

Function: slFun

Behavior: slBeh

Port Manager: SkillLevelPort

All variables:

SkillLevel Number	input output
manTechList	input
runSystem	
Skill level	output
slResult	output
layupCtrl	input
RTMCtrl	input
cmCtrl	input
imCtrl	input
fwCtrl	input
riCtrl	input

Qualitative predicates:

Quantitative predicates:

Existential predicates:

Default values:

Mappings:

InjectionMoldingPort@1

SkillLevel Number	?
manTechList	manTechList
runSystem	?
Skill level	?
slResult	?
layupCtrl	?
RTMCtrl	?
cmCtrl	?
imCtrl	imResult
fwCtrl	?
riCtrl	?

FilamentWindingPort@1

SkillLevel Number	?
manTechList	manTechList
runSystem	?
Skill level	?
slResult	?
layupCtrl	?
RTMCtrl	?
cmCtrl	?
imCtrl	?
fwCtrl	fwResult
riCtrl	?

Compression MolderPort@1

SkillLevel Number	?
manTechList	manTechList
runSystem	?
Skill level	?
slResult	?
layupCtrl	?
RTMCtrl	?
cmCtrl	cmResult
imCtrl	?
fwCtrl	?
riCtrl	?

LayupPort@1

SkillLevel Number	?
manTechList	manTechList
runSystem	?
Skill level	?
slResult	?
layupCtrl	lyupRslt
RTMCtrl	?
cmCtrl	?
imCtrl	?
fwCtrl	?
riCtrl	?

RTMPort@1

SkillLevel Number	?
manTechList	manTechList
runSystem	?
Skill level	?
slResult	?
layupCtrl	?
RTMCtrl	RTMResult
cmCtrl	?
imCtrl	?
fwCtrl	?
riCtrl	?

Resin InfusionPort@1

SkillLevel Number	?
manTechList	manTechList
runSystem	?
Skill level	?
slResult	?
layupCtrl	?
RTMCtrl	?
cmCtrl	?

imCtrl	?
fwCtrl	?
riCtrl	riResult

Controls:

RTMCtrl

type	rdInputControl
control name	RTMCtrl
control	RTMResult

cmCtrl

type	rdInputControl
control name	cmCtrl
control	cmResult

layupCtrl

type	rdInputControl
control name	layupCtrl
control	lyupRslt

imCtrl

type	rdInputControl
control name	imCtrl
control	imResult

riCtrl

type	rdInputControl
control name	riCtrl
control	riResult

fwCtrl

type	rdInputControl
control name	fwCtrl
control	fwResult

Device: ToolingCost

Function: tooligCost

Behavior: toolingCostBeh

Port Manager: ToolingCostPort

All variables:

ToolingCost Number	input output
manTechList	input
material cost	
Tool complexity	input
Tooling cost	output
toolingList	input
toolingCostResult	output
layupCtrl	input
RTMCtrl	input
cmCtrl	input
imCtrl	input
fwCtrl	input
riCtrl	input

Qualitative predicates:

Quantitative predicates:

Existential predicates:

Default values:

Mappings:

InjectionMoldingPort@1

ToolingCost Number	?
manTechList	manTechList
material cost	?
Tool complexity	tool complexity
Tooling cost	?
toolingList	toolingList
toolingCostResult	?
layupCtrl	?
RTMCtrl	?
cmCtrl	?
imCtrl	imResult
fwCtrl	?
riCtrl	?

FilamentWindingPort@1

ToolingCost Number	?
manTechList	manTechList
material cost	?
Tool complexity	?
Tooling cost	?
toolingList	toolingList
toolingCostResult	?

layupCtrl	?
RTMCtrl	?
cmCtrl	?
imCtrl	?
fwCtrl	fwResult
riCtrl	?

Compression MolderPort@1

ToolingCost Number	?
manTechList	manTechList
material cost	?
Tool complexity	Tool complexity
Tooling cost	?
toolingList	toolingList
toolingCostResult	?
layupCtrl	?
RTMCtrl	?
cmCtrl	cmResult
imCtrl	?
fwCtrl	?
riCtrl	?

LayupPort@1

ToolingCost Number	?
manTechList	manTechList
material cost	?
Tool complexity	Tool complexity
Tooling cost	?
toolingList	toolingList
toolingCostResult	?
layupCtrl	lyupRslt
RTMCtrl	?
cmCtrl	?
imCtrl	?
fwCtrl	?
riCtrl	?

RTMPort@1

ToolingCost Number	?
manTechList	manTechList
material cost	?
Tool complexity	Tool complexity
Tooling cost	?
toolingList	ToolingMaterial
toolingCostResult	?
layupCtrl	?
RTMCtrl	RTMResult
cmCtrl	?

imCtrl	?
fwCtrl	?
riCtrl	?

Resin InfusionPort@1

ToolingCost Number	?
manTechList	manTechList
material cost	?
Tool complexity	Tool complexity
Tooling cost	?
toolingList	toolingMaterial
toolingCostResult	?
layupCtrl	?
RTMCtrl	?
cmCtrl	?
imCtrl	?
fwCtrl	?
riCtrl	riResult

Controls:

fwCtrl

type	rdInputControl
control name	fwCtrl
control	fwResult

riCtrl

type	rdInputControl
control name	riCtrl
control	riResult

RTMCtrl

type	rdInputControl
control name	RTMCtrl
control	RTMResult

layupCtrl

type	rdInputControl
control name	layupCtrl
control	lyupRslt

imCtrl

type	rdInputControl
control name	imCtrl
control	imResult

cmCtrl

type	rdInputControl
control name	cmCtrl
control	cmResult

Device: ToolingTT

Function: tttFn

Behavior: tttBeh

Port Manager: ToolingTTPort

All variables:

ToolingTT Number	input output
GeometricalComplexity	input
size	input
Tooling Turn Around Time	output
toolingList	input
toolinTTResult	output
lyupCtrl	input
RTMCtrl	input
cmCtrl	input
riCtrl	input
fwCtrl	input
imCtrl	input

Qualitative predicates:

Quantitative predicates:

Existential predicates:

Default values:

Mappings:

InjectionMoldingPort@1

ToolingTT Number	?
GeometricalComplexity	GeometricalComplexity
size	size
Tooling Turn Around Time	?
toolingList	toolingList
toolinTTResult	?
lyupCtrl	?
RTMCtrl	?
cmCtrl	?
riCtrl	?
fwCtrl	?
imCtrl	imResult

RefinerPort@1

ToolingTT Number	?
GeometricalComplexity	?
size	size
Tooling Turn Around Time	?
toolingList	?
toolinTTResult	?
lyupCtrl	?
RTMCtrl	?
cmCtrl	?

riCtrl	?
fwCtrl	?
imCtrl	?

FilamentWindingPort@1

ToolingTT Number	?
GeometricalComplexity	GeometricalComplexity
size	size
Tooling Turn Around Time	?
toolingList	toolingList
toolinTTResult	?
lyupCtrl	?
RTMCtrl	?
cmCtrl	?
riCtrl	?
fwCtrl	fwResult
imCtrl	?

Compression MolderPort@1

ToolingTT Number	?
GeometricalComplexity	GeometricalComplexity
size	?
Tooling Turn Around Time	?
toolingList	toolingList
toolinTTResult	?
lyupCtrl	?
RTMCtrl	?
cmCtrl	cmResult
riCtrl	?
fwCtrl	?
imCtrl	?

LayupPort@1

ToolingTT Number	?
GeometricalComplexity	GeometricalComplexity
size	size
Tooling Turn Around Time	?
toolingList	toolingList
toolinTTResult	?
lyupCtrl	lyupRslt
RTMCtrl	?
cmCtrl	?
riCtrl	?
fwCtrl	?
imCtrl	?

RTMPort@1

ToolingTT Number	?
GeometricalComplexity	GeometricalComplexity

size	size
Tooling Turn Around Time	?
toolingList	ToolingMaterial
toolinTTResult	?
lyupCtrl	?
RTMCtrl	RTMResult
cmCtrl	?
riCtrl	?
fwCtrl	?
imCtrl	?

Resin InfusionPort@1

ToolingTT Number	?
GeometricalComplexity	GeometricalComplexity
size	size
Tooling Turn Around Time	?
toolingList	toolingMaterial
toolinTTResult	?
lyupCtrl	?
RTMCtrl	?
cmCtrl	?
riCtrl	riResult
fwCtrl	?
imCtrl	?

Controls:

cmCtrl

type	rdInputControl
control name	cmCtrl
control	cmResult

riCtrl

type	rdInputControl
control name	riCtrl
control	riResult

fwCtrl

type	rdInputControl
control name	fwCtrl
control	fwResult

RTMCtrl

type	rdInputControl
control name	RTMCtrl
control	RTMResult

lyupCtrl

type	rdInputControl
control name	lyupCtrl
control	lyupRslt

imCtrl

type	rdInputControl
control name	imCtrl
control	imResult

Device: Compression Molder
Function: cmFunction
Behavior: cmBehavior
Port Manager: Compression MolderPort
All variables:

Compression Molder Number	input output
fiberArchList	
GeometricalComplexity	input
labor	output
pressure-ksi	output
production	input
resinList	
temperature	output
Tool complexity	output
toolingList	output
boosterInputControl	input
cmResult	output
manTechList	output

Qualitative predicates:
Quantitative predicates:
Existential predicates:
Default values:
Mappings:
selectionBoosterPort@1

Compression Molder Number	?
fiberArchList	?
GeometricalComplexity	?
labor	?
pressure-ksi	?
production	?
resinList	?
temperature	?
Tool complexity	?
toolingList	?
boosterInputControl	boosterResult

RefinerPort@1

Compression Molder Number	?
fiberArchList	fiberArchList
GeometricalComplexity	GeometricalComplexity
labor	?
pressure-ksi	?
production	production
resinList	resinList
temperature	?
Tool complexity	?

toolingList	?

Controls:

boosterInputControl

type	hcInputControl
control name	boosterInputControl
output	manTechList
control	Compression Molding

Device: FilamentWinding
Function: fwFn
Behavior: fnBeh
Port Manager: FilamentWindingPort
All variables:

FilamentWinding Number	input output
cure type	output
delivery system	output
functional requirements	input
GeometricalComplexity	input
pressure	output
production	input
resinList	input
size	input
temperature	output
toolingList	output
fwResult	output
fwBoosterCtrl	input
manTechList	output

Qualitative predicates:
Quantitative predicates:
Existential predicates:
Default values:
Mappings:
selectionBoosterPort@1

FilamentWinding Number	?
cure type	?
delivery system	?
functional requirements	?
GeometricalComplexity	?
pressure	?
production	?
resinList	?
size	?
temperature	?
toolingList	?
fwResult	?
fwBoosterCtrl	boosterResult

RefinerPort@1

FilamentWinding Number	?
cure type	?
delivery system	?
functional requirements	functional requirements
GeometricalComplexity	GeometricalComplexity
pressure	?

production	production
resinList	resinList
size	size
temperature	?
toolingList	?
fwResult	?
fwBoosterCtrl	?

Controls:
fwBoosterCtrl

type	hcInputControl
control name	fwBoosterCtrl
output	manTechList
control	Filament Winding

Device: InjectionMolding
Function: imFn
Behavior: imBeh
Port Manager: InjectionMoldingPort
All variables:

InjectionMolding Number	input output
GeometricalComplexity	input
heated tool	output
inserts	input
pressure-ksi	output
production	input
resinList	input
size	input
temperature	output
tool complexity	output
toolingList	output
imResult	output
imBoostingCtrl	input
manTechList	output
Qualitative predicates:	
Quantitative predicates:	
Existential predicates:	
GeometricalComplexity	*
size	*
inserts	*
InjectionMolding Number	?
resinList	*
production	*

Default values:

inserts	none

Mappings:
selectionBoosterPort@1

InjectionMolding Number	?
GeometricalComplexity	?
heated tool	?
inserts	?
pressure-ksi	?
production	?
resinList	?
size	?
temperature	?
tool complexity	?
toolingList	?
imResult	?
imBoostingCtrl	boosterResult

RefinerPort@1

InjectionMolding Number	?
GeometricalComplexity	GeometricalComplexity
heated tool	?
inserts	inserts
pressure-ksi	?
production	production
resinList	resinList
size	size
temperature	?
tool complexity	?
toolingList	?

Controls:

imBoostingCtrl

type	hcInputControl
control	Injection Molding
output	manTechList

Device: Layup

Function: LayupFn

Behavior: LayupBeh

Port Manager: LayupPort

All variables:

Layup Number	input output
curing type	output
depth	input
fiberList	
functional requirements	input
GeometricalComplexity	input
labor	output
postcuring	output
pressure	output
production	input
resin prepreg/wet	output
resinList	input
runSystem	
size	input
SurfaceQuality	input
temperature	output
Tolerances	input
Tool complexity	output
toolingList	output
lyupBoostCtrl	input
lyupRslt	output
manTechList	output

Qualitative predicates:

Quantitative predicates:

Existential predicates:

fiberList	*
size	*
depth	*
Layup Number	*
resinList	*
GeometricalComplexity	*
Tolerances	*
functional requirements	*
SurfaceQuality	*
lyupBoostCtrl	?
production	*

Default values:

Mappings:

selectionBoosterPort@1

Layup Number	?

curing type	?
depth	?
fiberList	?
functional requirements	?
GeometricalComplexity	?
labor	?
postcuring	?
pressure	?
production	?
resin prepreg/wet	?
resinList	?
runSystem	?
size	?
SurfaceQuality	?
temperature	?
Tolerances	?
Tool complexity	?
toolingList	?
lyupBoostCtrl	boosterResult
lyupRslt	?
manTechList	?

RefinerPort@1

Layup Number	?
curing type	?
depth	depth
fiberList	fiberList
functional requirements	functional requirements
GeometricalComplexity	GeometricalComplexity
labor	?
postcuring	?
pressure	?
production	production
resin prepreg/wet	?
resinList	resinList
runSystem	?
size	size
SurfaceQuality	SurfaceQuality
temperature	?
Tolerances	Tolerances
Tool complexity	?
toolingList	?
lyupBoostCtrl	?
lyupRslt	?
manTechList	?

Controls:

lyupBoostCtrl

type	hcInputControl
control name	lyupBoostCtrl
output	manTechList
control	Layup

Device: Resin Infusion
Function: riFn
Behavior: riBeh
Port Manager: Resin InfusionPort
All variables:

Resin Infusion Number	input output
curing type	output
fiberArchList	input
FiberFormingMethod	output
fiberList	input
functional requirements	input
GeometricalComplexity	input
labor	output
postcuring	output
pressure	output
production	input
resinList	input
runSystem	
size	input
SurfaceQuality	input
temperature	output
Tolerances	input
Tool complexity	output
toolingMaterial	output
Wall_Thickness	input
riBoosterCtrl	input
riResult	output
manTechList	output

Qualitative predicates:
Quantitative predicates:
Existential predicates:

resinList	*
Wall_Thickness	*
functional requirements	*
Resin Infusion Number	?
riBoosterCtrl	?
fiberList	*
SurfaceQuality	*
production	*
fiberArchList	*
size	*
Tolerances	*
GeometricalComplexity	*

Default values:
Mappings:

selectionBoosterPort@1

Resin Infusion Number	?
curing type	?
fiberArchList	?
FiberFormingMethod	?
fiberList	?
functional requirements	?
GeometricalComplexity	?
labor	?
postcuring	?
pressure	?
production	?
resinList	?
runSystem	?
size	?
SurfaceQuality	?
temperature	?
Tolerances	?
Tool complexity	?
toolingMaterial	?
Wall_Thickness	?
riBoosterCtrl	boosterResult
riResult	?
manTechList	?

RefinerPort@1

Resin Infusion Number	?
curing type	?
fiberArchList	fiberArchList
FiberFormingMethod	?
fiberList	fiberList
functional requirements	functional requirements
GeometricalComplexity	GeometricalComplexity
labor	?
postcuring	?
pressure	?
production	production
resinList	resinList
runSystem	?
size	size
SurfaceQuality	SurfaceQuality
temperature	?
Tolerances	Tolerances
Tool complexity	?
toolingMaterial	?
Wall_Thickness	depth

riBoosterCtrl	?
riResult	?
manTechList	?

Controls:

riBoosterCtrl

type	hcInputControl
control name	riBoosterCtrl
output	manTechList
control	Resin Infusion

Device: RTM

Function: RTMFn

Behavior: RTMBeh

Port Manager: RTMPort

All variables:

RTM Number	input output
Curing temperature	output
Curing time	output
Curing type	output
FiberFormingMethod	output
Fiber_Architecture	input
GeometricalComplexity	input
Heating method	output
labor	output
Postcuring Required	output
Production	input
resinList	input
runSystem	
size	input
SurfaceQuality	input
Tolerances	input
Tool complexity	output
ToolingMaterial	output
rtmBoostCtrl	input
RTMResult	output
manTechList	output

Qualitative predicates:

Quantitative predicates:

Existential predicates:

Default values:

Mappings:

selectionBoosterPort@1

RTM Number	?
Curing temperature	?
Curing time	?
Curing type	?
FiberFormingMethod	?
Fiber_Architecture	?
GeometricalComplexity	?
Heating method	?
labor	?
Postcuring Required	?
Production	?
resinList	?
runSystem	?

size	?
SurfaceQuality	?
Tolerances	?
Tool complexity	?
ToolingMaterial	?
rtmBoostCtrl	boosterResult
RTMResult	?
manTechList	?

RefinerPort@1

RTM Number	?
Curing temperature	?
Curing time	?
Curing type	?
FiberFormingMethod	?
Fiber_Architecture	fiberArchList
GeometricalComplexity	GeometricalComplexity
Heating method	?
labor	?
Postcuring Required	?
Production	production
resinList	resinList
runSystem	?
size	size
SurfaceQuality	SurfaceQuality
Tolerances	Tolerances
Tool complexity	?
ToolingMaterial	?
rtmBoostCtrl	?
RTMResult	?
manTechList	?

Controls:

rtmBoostCtrl

type	hcInputControl
control name	rtmBoostCtrl
output	manTechList
control	Resin Transfer Molding

Device: matSel

Function: matSelFn

Behavior: matSelBh

Port Manager: matSelPort

All variables:

matSel Number	input output
resinList	input
matSelResult	output

Qualitative predicates:

resinList ~= unknown

Quantitative predicates:

Existential predicates:

resinList	*
matSel Number	?

Default values:

Mappings:

SelectorPort@1

matSel Number	?
resinList	partMaterial
matSelResult	?

Controls:

Device: operationSelector

Function: opSelFn

Behavior: opSelBh

Port Manager: operationSelectorPort

All variables:

operationSelector Number	input output
Aspect_Ratio	input
Fiber_Architecture	input
partMaterial	input
Shape	input
size	input
Wall_Thickness	input
opSelResult	output

Qualitative predicates:

size ~= unknown

Aspect_Ratio ~= unknown

partMaterial ~= unknown

Fiber_Architecture ~= unknown

Shape ~= unknown

Wall_Thickness ~= unknown

Quantitative predicates:

Existential predicates:

size	*
Fiber_Architecture	*
Shape	*
operationSelector Number	?
partMaterial	*
Wall_Thickness	*
Aspect_Ratio	*

Default values:

Fiber_Architecture	quasiIsotropic
Shape	casting
Aspect_Ratio	medium
size	medium
partMaterial	DGEBA
Wall_Thickness	medium

Mappings:

Controls:

Device: selectionBooster

Function: boosterFn

Behavior: boosterBh

Port Manager: selectionBoosterPort

All variables:

selectionBooster Number	input output
list1	input
list2	input
boosterResult	output

Qualitative predicates:

Quantitative predicates:

Existential predicates:

selectionBooster Number	?
list1	*
list2	*

Default values:

Mappings:

matSelPort@1

selectionBooster Number	?
list1	matSelResult
list2	?
boosterResult	?

operationSelectorPort@1

selectionBooster Number	?
list1	?
list2	opSelResult
boosterResult	?

Controls:

APPENDIX C

MANUFACTURING ONTOLOGY REDESIGNED IN PROTEGE FRAMEWORK

```
(defclass PartTechnology
        (is-a Root)
        (role concrete)
        (slot techname
                (type STRING)
                (create-accessor read-write))
        (slot tech-description
;+              (allowed-classes Technology)
                (type INSTANCE)
                (create-accessor read-write)))

(defclass Technology
        (is-a Root)
        (role abstract)
        (multislot tool-complexity
;+              (allowed-classes ToolComplexity)
                (type INSTANCE)
                (create-accessor read-write))
        (slot name
                (type STRING)
                (create-accessor read-write)))

(defclass ResinTransferMolding
        (is-a Technology)
        (role concrete)
        (multislot postcuring-required
;+              (allowed-classes Postcuring)
                (type INSTANCE)
                (create-accessor read-write))
        (multislot curing-temperature
;+              (allowed-classes Temperature)
                (type INSTANCE)
                (create-accessor read-write))
        (multislot heating-method
;+              (allowed-classes HeatingMethod)
                (type INSTANCE)
                (create-accessor read-write))
        (multislot fiber-forming-method
;+              (allowed-classes FiberFormingMethod)
                (type INSTANCE)
                (create-accessor read-write))
        (multislot curing-time
;+              (allowed-classes CuringTime)
                (type INSTANCE)
                (create-accessor read-write)))

(defclass Sprayup
        (is-a Technology)
        (role concrete)
        (multislot labor
;+              (allowed-classes Labor)
                (type INSTANCE)
                (create-accessor read-write))
        (multislot cure-type
;+              (allowed-classes CureType)
                (type INSTANCE)
                (create-accessor read-write)))

(defclass Extrusion
        (is-a Technology)
        (role concrete))

(defclass Pultrusion
        (is-a Technology)
        (role concrete))

(defclass CompressionMolding
        (is-a Technology)
        (role concrete)
        (multislot pressure-ksi
;+              (allowed-classes PressureKSI)
                (type INSTANCE)
                (create-accessor read-write))
        (multislot labor
;+              (allowed-classes Labor)
                (type INSTANCE)
                (create-accessor read-write))
        (multislot temperature
;+              (allowed-classes Temperature)
                (type INSTANCE)
                (create-accessor read-write)))

(defclass Layup
        (is-a Technology)
        (role concrete)
        (multislot pressure
;+              (allowed-classes Pressure)
                (type INSTANCE)
                (create-accessor read-write))
        (multislot labor
;+              (allowed-classes Labor)
                (type INSTANCE)
                (create-accessor read-write))
        (multislot postcuring
;+              (allowed-classes Postcuring)
                (type INSTANCE)
                (create-accessor read-write))
        (multislot cure-type
;+              (allowed-classes CureType)
                (type INSTANCE)
                (create-accessor read-write))
        (multislot resin-prepreg-wet
;+              (allowed-classes ResinPrepregWet)
                (type INSTANCE)
                (create-accessor read-write))
        (multislot temperature
;+              (allowed-classes Temperature)
                (type INSTANCE)
                (create-accessor read-write)))

(defclass TechnologyMisc
        (is-a Root)
        (role abstract))

(defclass CureType
        (is-a TechnologyMisc)
        (role concrete)
        (slot cure-type
                (allowed-values autoclave microwave oven room press)
                (type SYMBOL)
                (create-accessor read-write)))

(defclass CuringTime
        (is-a TechnologyMisc)
        (role concrete)
        (slot curing-time
                (allowed-values days hours minutes)
                (type SYMBOL)
                (create-accessor read-write)))

(defclass DeliverySystem
        (is-a TechnologyMisc)
        (role concrete)
        (slot delivery-system
                (allowed-values prepreg wet winding wet rolled)
                (type SYMBOL)
                (create-accessor read-write)))

(defclass FiberFormingMethod
        (is-a TechnologyMisc)
        (role concrete)
        (slot fiber-forming-method
                (allowed-values braiding cut and place directed fiber stamping
textile preforming)
                (type SYMBOL)
                (create-accessor read-write)))

(defclass HeatedTool
        (is-a TechnologyMisc)
        (role concrete)
        (slot heated-tool
                (allowed-values yes no)
                (type SYMBOL)
                (create-accessor read-write)))

(defclass HeatingMethod
        (is-a TechnologyMisc)
        (role concrete)
        (slot heating-method
                (allowed-values electric heat blanket heated platens microwave oil
oven steam no heating)
                (type SYMBOL)
                (create-accessor read-write)))

(defclass Labor
        (is-a TechnologyMisc)
        (role concrete)
        (slot labor
                (allowed-values high medium low)
                (type SYMBOL)
                (create-accessor read-write)))

(defclass Postcuring
        (is-a TechnologyMisc)
        (role concrete)
        (slot postcuring
                (allowed-values yes no possible)
                (type SYMBOL)
                (create-accessor read-write)))

(defclass Pressure
        (is-a TechnologyMisc)
        (role concrete)
        (slot pressure
                (allowed-values low moderate high)
                (type SYMBOL)
                (create-accessor read-write)))

(defclass PressureKSI
        (is-a TechnologyMisc)
        (role concrete)
        (slot pressure-ksi
                (allowed-values 0. 5-2 0. 5-1. 5 1. 5-3. 5 2-5 5-10 10-20)
                (type SYMBOL)
                (create-accessor read-write)))

(defclass ResinPrepregWet
        (is-a TechnologyMisc)
        (role concrete)
        (slot prepreg-wet
                (allowed-values prepreg wet)
                (type SYMBOL)
                (create-accessor read-write)))

(defclass Temperature
        (is-a TechnologyMisc)
        (role concrete)
        (slot temperature
                (allowed-values 25-50 25-100 50-150 80-150 100-200 150-200
150-300 200-250 200-300 250-350 300-450)
                (type SYMBOL)
                (create-accessor read-write)))

(defclass ToolComplexity
        (is-a TechnologyMisc)
        (role concrete)
        (slot tool-complexity
                (allowed-values veryhigh high medium low)
                (type SYMBOL)
                (create-accessor read-write)))
```

APPENDIX D

MANUFACTURING ONTOLOGY REDESIGNED IN XML

```
<<header id="Manufacturing Ontology">
  <class id="Tooling with Parameters">
    <category id="Aluminium"/>
    <category id="Nickel Electroforms"/>
    <category id="CRP"/>
    <category id="Ceramics"/>
    <category id="Polymers"/>
    <category id="GRP"/>
    <category id="Cast Iron"/>
    <category id="Tooling Foam"/>
    <category id="Steel"/>
  </class>
  <class id="Technologies with Parameters">
    <category id="Sprayup">
      <parameter type="oneOfVar" id="cure type">
        <value id="autoclave"/>
        <value id="microwave"/>
        <value id="oven"/>
        <value id="heated tool"/>
        <value id="room cure"/>
      </parameter>
      <parameter type="oneOfVar" id="pressure-psi"/>
      <parameter type="oneOfVar" id="labor">
        <value id="intense"/>
        <value id="medium"/>
        <value id="low"/>
      </parameter>
      <parameter type="oneOfVar" id="temperauture"/>
      <parameter type="oneOfVar" id="time"/>
    </category>
    <category id="Extrusion">
      <parameter type="oneOfVar" id="pressure-psi"/>
      <parameter type="oneOfVar" id="time"/>
      <parameter type="oneOfVar" id="temperature"/>
    </category>
    <category id="Filament winding">
      <parameter type="oneOfVar" id="temperature">
        <value id="25-50"/>
        <value id="50-150"/>
    <value id="150-200"/>
    <value id="200-300"/>
  </parameter>
  <parameter type="oneOfVar" id="delivery system">
    <value id="prepreg"/>
    <value id="wet winding"/>
    <value id="wet rerolled"/>
  </parameter>
  <parameter type="oneOfVar" id="pressure">
    <value id="low"/>
    <value id="moderate"/>
    <value id="high"/>
  </parameter>
  <parameter type="oneOfVar" id="cure type">
    <value id="autoclave"/>
    <value id="microwave"/>
    <value id="oven"/>
    <value id="room"/>
  </parameter>
</category>
<category id="Resin Transfer Molding">
  <parameter type="oneOfVar" id="Postcuring Required">
    <value id="yes"/>
    <value id="no"/>
    <value id="possible"/>
  </parameter>
  <parameter type="oneOfVar" id="Curing temperature">
    <value id="100-200"/>
    <value id="200-300"/>
    <value id="25-100"/>
  </parameter>
  <parameter type="oneOfVar" id="Heating method">
    <value id="electric"/>
    <value id="heat blanket"/>
    <value id="heated platens"/>
    <value id="microwave"/>
    <value id="oil"/>
    <value id="oven"/>
    <value id="steam"/>
    <value id="no heating"/>
  </parameter>
  <parameter type="oneOfVar" id="Tool complexity">
    <value id="high"/>
    <value id="low"/>
    <value id="medium"/>
    <value id="very high"/>
  </parameter>
  <parameter type="oneOfVar" id="FiberFormingMethod">
    <value id="Braiding"/>
    <value id="Cut-and-Place"/>
    <value id="Directed Fiber"/>
    <value id="Stamping"/>
    <value id="Textile Preforming"/>
  </parameter>
  <parameter type="oneOfVar" id="Curing time">
    <value id="days"/>
    <value id="hours"/>
    <value id="minutes"/>
  </parameter>
</category>
<category id="Compression Molding">
  <parameter type="oneOfVar" id="pressure-ksi">
    <value id="0. 5-1. 5"/>
    <value id="1. 5-3. 5"/>
  </parameter>
  <parameter type="oneOfVar" id="labor">
    <value id="high"/>
    <value id="medium"/>
    <value id="low"/>
  </parameter>
  <parameter type="oneOfVar" id="temperature">
    <value id="150-200"/>
    <value id="200-250"/>
    <value id="250-350"/>
  </parameter>
  <parameter type="oneOfVar" id="Tool complexity">
    <value id="high"/>
    <value id="low"/>
    <value id="medium"/>
    <value id="very high"/>
  </parameter>
</category>
<category id="Layup">
  <parameter type="oneOfVar" id="pressure">
    <value id="low"/>
    <value id="moderate"/>
    <value id="high"/>
  </parameter>
  <parameter type="oneOfVar" id="labor">
    <value id="high"/>
    <value id="medium"/>
    <value id="low"/>
  </parameter>
  <parameter type="oneOfVar" id="postcuring">
    <value id="yes"/>
    <value id="no"/>
    <value id="possible"/>
  </parameter>
  <parameter type="oneOfVar" id="curing type">
    <value id="oven"/>
    <value id="autoclave"/>
    <value id="microwave"/>
    <value id="press"/>
    <value id="room"/>
  </parameter>
  <parameter type="oneOfVar" id="Tool complexity">
    <value id="high"/>
    <value id="low"/>
    <value id="medium"/>
    <value id="very high"/>
  </parameter>
  <parameter type="oneOfVar" id="resin prepreg/wet">
    <value id="prepreg"/>
    <value id="wet"/>
  </parameter>
  <parameter type="oneOfVar" id="temperature">
    <value id="25-50"/>
    <value id="50-150"/>
    <value id="150-200"/>
    <value id="200-300"/>
  </parameter>
</category>
<category id="Pultrusion">
  <parameter type="oneOfVar" id="pressure-psi"/>
  <parameter type="oneOfVar" id="time"/>
  <parameter type="oneOfVar" id="temperature"/>
</category>
<category id="Resin Infusion">
  <parameter type="oneOfVar" id="pressure">
    <value id="low"/>
    <value id="moderate"/>
    <value id="high"/>
  </parameter>
  <parameter type="oneOfVar" id="labor">
    <value id="high"/>
    <value id="medium"/>
    <value id="low"/>
  </parameter>
  <parameter type="oneOfVar" id="postcuring">
    <value id="yes"/>
    <value id="no"/>
    <value id="possible"/>
  </parameter>
  <parameter type="oneOfVar" id="curing type">
    <value id="oven"/>
    <value id="microwave"/>
    <value id="press"/>
    <value id="room"/>
  </parameter>
  <parameter type="oneOfVar" id="Tool complexity">
    <value id="high"/>
    <value id="low"/>
    <value id="medium"/>
    <value id="very high"/>
  </parameter>
  <parameter type="oneOfVar" id="FiberFormingMethod">
    <value id="Braiding"/>
    <value id="Textile_Preforming"/>
    <value id="Directed_Fiber"/>
    <value id="Stamping"/>
    <value id="Cut-and-Place"/>
  </parameter>
  <parameter type="oneOfVar" id="temperature">
    <value id="25-50"/>
    <value id="50-150"/>
    <value id="150-200"/>
  </parameter>
</category>
<category id="Injection Molding">
  <parameter type="oneOfVar" id="pressure-ksi">
    <value id="0. 5-2"/>
    <value id="2-5"/>
    <value id="5-10"/>
    <value id="10-20"/>
  </parameter>
  <parameter type="oneOfVar" id="heated tool">
    <value id="yes"/>
    <value id="no"/>
  </parameter>
  <parameter type="oneOfVar" id="temperature">
    <value id="80-150"/>
    <value id="150-300"/>
    <value id="300-450"/>
  </parameter>
```

```
    <parameter type="oneOfVar" id="tool complexity">
      <value id="very high"/>
      <value id="high"/>
      <value id="medium"/>
      <value id="low"/>
    </parameter>
  </category>
</class>
<class id="Joining Technologies with Parameters"/>
<class id="Joinings with Parameters">
  <category id="Washer"/>
  <category id="Bushing"/>
  <category id="Rivet"/>
  <category id="Strap"/>
  <category id="Lap"/>
  <category id="Fastener"/>
  <category id="Bolt"/>
  <category id="Bearing"/>
  <category id="Weld"/>
  <category id="Pin"/>
  <category id="Snap Fit"/>
  <category id="Screw"/>
  <category id="Adhesive"/>
</class>
<class id="Features with Parameters">
  <category id="blind hole">
    <parameter type="oneOfVar" id="tolerance">
      <value id="loose"/>
      <value id="tight"/>
    </parameter>
  </category>
  <parameter type="singleValued">radius</parameter>
  <parameter type="singleValued">depth</parameter>
  <category id="flange">
    <parameter type="oneOfVar" id="tolerance">
      <value id="loose"/>
      <value id="tight"/>
    </parameter>
  </category>
  <parameter type="singleValued">height</parameter>
  <parameter type="singleValued">length</parameter>
  <parameter type="singleValued">width</parameter>
  <category id="wall/web">
    <parameter type="oneOfVar" id="tolerance">
      <value id="loose"/>
      <value id="tight"/>
    </parameter>
  </category>
  <parameter type="singleValued">thickness</parameter>
  <category id="insert">
    <parameter type="oneOfVar" id="numberOf">
      <value id="none"/>
      <value id="1"/>
      <value id="&gt;2"/>
    </parameter>
    <parameter type="oneOfVar" id="tolerance">
      <value id="loose"/>
      <value id="tight"/>
    </parameter>
  </category>
  <category id="bridge">
    <parameter type="oneOfVar" id="tolerance">
      <value id="loose"/>
      <value id="tight"/>
    </parameter>
  </category>
  <parameter type="singleValued">height</parameter>
  <parameter type="singleValued">length</parameter>
  <parameter type="singleValued">width</parameter>
  <category id="bend">
    <parameter type="oneOfVar" id="tolerance">
      <value id="loose"/>
      <value id="tight"/>
    </parameter>
  </category>
  <parameter type="singleValued">radius</parameter>
  <parameter type="singleValued">angle</parameter>
  <category id="blind slot">
    <parameter type="oneOfVar" id="tolerance">
      <value id="loose"/>
      <value id="tight"/>
    </parameter>
  </category>
  <parameter type="singleValued">depth</parameter>
  <parameter type="singleValued">length</parameter>
  <parameter type="singleValued">width</parameter>
  <category id="marking">
    <parameter type="oneOfVar" id="tolerance">
      <value id="loose"/>
      <value id="tight"/>
    </parameter>
  </category>
```

```
<category id="groove">
  <parameter type="oneOfVar" id="machining">
    <value id="yes"/>
    <value id="no"/>
  </parameter>
  <parameter type="oneOfVar" id="molding">
    <value id="yes"/>
    <value id="no"/>
  </parameter>
  <parameter type="oneOfVar" id="tolerance">
    <value id="loose"/>
    <value id="tight"/>
  </parameter>
</category>
<parameter type="singleValued">depth</parameter>
<parameter type="singleValued">length</parameter>
<parameter type="singleValued">width</parameter>
<category id="hole">
  <parameter type="oneOfVar" id="machining">
    <value id="yes"/>
    <value id="no"/>
  </parameter>
  <parameter type="oneOfVar" id="molding">
    <value id="yes"/>
    <value id="no"/>
  </parameter>
  <parameter type="oneOfVar" id="tolerance">
    <value id="loose"/>
    <value id="tight"/>
  </parameter>
</category>
<parameter type="singleValued">radius</parameter>
<parameter type="singleValued">numberOf</parameter>
<category id="step">
  <parameter type="oneOfVar" id="tolerance">
    <value id="loose"/>
    <value id="tight"/>
  </parameter>
</category>
<parameter type="singleValued">width</parameter>
<category id="depression">
  <parameter type="oneOfVar" id="tolerance">
    <value id="loose"/>
    <value id="tight"/>
  </parameter>
</category>
```

```
<parameter type="singleValued">depth</parameter>
<parameter type="singleValued">length</parameter>
<parameter type="singleValued">width</parameter>
<category id="blind pocket">
  <parameter type="oneOfVar" id="tolerance">
    <value id="loose"/>
    <value id="tight"/>
  </parameter>
</category>
<parameter type="singleValued">depth</parameter>
<parameter type="singleValued">length</parameter>
<parameter type="singleValued">width</parameter>
<category id="internal thread">
  <parameter type="oneOfVar" id="tolerance">
    <value id="loose"/>
    <value id="tight"/>
  </parameter>
</category>
<category id="tolerance">
  <parameter type="oneOfVar" id="range">
    <value id="High"/>
    <value id="Medium"/>
    <value id="Low"/>
  </parameter>
  <parameter type="oneOfVar" id="tolerance">
    <value id="loose"/>
    <value id="tight"/>
  </parameter>
</category>
<category id="fin">
  <parameter type="oneOfVar" id="tolerance">
    <value id="loose"/>
    <value id="tight"/>
  </parameter>
</category>
<parameter type="singleValued">height</parameter>
<parameter type="singleValued">length</parameter>
<category id="finish">
  <parameter type="oneOfVar" id="quality">
    <value id="High"/>
    <value id="Medium"/>
    <value id="Low"/>
  </parameter>
  <parameter type="oneOfVar" id="tolerance">
    <value id="loose"/>
    <value id="tight"/>
  </parameter>
</category>
<category id="cutout">
  <parameter type="oneOfVar" id="complexity">
    <value id="high"/>
    <value id="medium"/>
    <value id="low"/>
  </parameter>
  <parameter type="oneOfVar" id="size">
    <value id="medium"/>
    <value id="small"/>
    <value id="large"/>
  </parameter>
  <parameter type="oneOfVar" id="tolerance">
    <value id="loose"/>
    <value id="tight"/>
  </parameter>
</category>
<category id="chamfer">
  <parameter type="oneOfVar" id="tolerance">
    <value id="loose"/>
    <value id="tight"/>
  </parameter>
</category>
<parameter type="singleValued">radius</parameter>
<category id="slot">
  <parameter type="oneOfVar" id="tolerance">
    <value id="loose"/>
    <value id="tight"/>
  </parameter>
</category>
<parameter type="singleValued">depth</parameter>
<parameter type="singleValued">length</parameter>
<parameter type="singleValued">width</parameter>
<category id="extruded hole">
  <parameter type="oneOfVar" id="tolerance">
    <value id="loose"/>
    <value id="tight"/>
  </parameter>
</category>
<parameter type="singleValued">radius</parameter>
<parameter type="singleValued">height</parameter>
<category id="external thread">
  <parameter type="oneOfVar" id="tolerance">
    <value id="loose"/>
    <value id="tight"/>
    </parameter>
  </category>
  <category id="boss">
    <parameter type="oneOfVar" id="tolerance">
      <value id="loose"/>
      <value id="tight"/>
    </parameter>
  </category>
  <parameter type="singleValued">height</parameter>
  <parameter type="singleValued">lenght</parameter>
  <parameter type="singleValued">width</parameter>
  <category id="pocket">
    <parameter type="oneOfVar" id="tolerance">
      <value id="loose"/>
      <value id="tight"/>
    </parameter>
  </category>
  <parameter type="singleValued">depth</parameter>
  <parameter type="singleValued">length</parameter>
  <parameter type="singleValued">width</parameter>
  <category id="rib">
    <parameter type="oneOfVar" id="tolerance">
      <value id="loose"/>
      <value id="tight"/>
    </parameter>
  </category>
  <parameter type="singleValued">height</parameter>
  <parameter type="singleValued">length</parameter>
  <parameter type="singleValued">width</parameter>
</class>
<class id="Matrix Material with Parameters">
  <category id="Aluminum"/>
  <category id="PA66"/>
  <category id="PAI"/>
  <category id="PEI"/>
  <category id="PC/PBT"/>
  <category id="Metal"/>
  <category id="ABS"/>
  <category id="ABS/PBT"/>
  <category id="PSU"/>
  <category id="PMR 15 Monomers"/>
  <category id="Epoxidized Phenolic Novolac"/>
  <category id="Chlorendic Resin"/>
  <category id="PC/ABS"/>
  <category id="PBT"/>
  <category id="PPS"/>
  <category id="PET"/>
  <category id="TGETPE"/>
  <category id="PC"/>
  <category id="Vinyl Ester Resin"/>
  <category id="DGEBA"/>
  <category id="TGMDA"/>
  <category id="BPA Fumarate Resin"/>
  <category id="Phenolic Novolac Resin"/>
  <category id="Phenolic Resole Resin"/>
  <category id="4,4'-MDA-BMI"/>
  <category id="Isophthalic Resin"/>
  <category id="PPO"/>
  <category id="Orthophthalic Resin"/>
  <category id="Thermid 600 Oligomers"/>
  <category id="PAS"/>
  <category id="PEEK"/>
</class>
<class id="Fiber Type with Parameters">
  <category id="AS-4 Carbon"/>
  <category id="P-100 Graphite"/>
  <category id="Kevlar-29"/>
  <category id="C-Glass"/>
  <category id="P-55 Graphite"/>
  <category id="Kevlar-149"/>
  <category id="SiC"/>
  <category id="Boron"/>
  <category id="None"/>
  <category id="E-Glass"/>
  <category id="Kevlar-49"/>
  <category id="S-Glass"/>
  <category id="Quartz"/>
</class>
<class id="Fiber Architecture with Parameters">
  <category id="continous SM"/>
  <category id="woven"/>
  <category id="braided"/>
  <category id="chopped"/>
  <category id="unidirectional"/>
  <category id="quasiIsotropic"/>
  <category id="chopped SM"/>
  <category id="special"/>
</class>
<class id="Shape with Parameters">
  <category id="casting">
    <parameter type="oneOfVar" id="Aspect_Ratio">
      <value id="high"/>
    <value id="medium"/>
    <value id="low"/>
  </parameter>
  <parameter type="oneOfVar" id="size">
    <value id="big"/>
    <value id="medium"/>
    <value id="small"/>
  </parameter>
  <parameter type="oneOfVar" id="GeometricalComplexity">
    <value id="High"/>
    <value id="Medium"/>
    <value id="Low"/>
  </parameter>
  <parameter type="oneOfVar" id="Wall_Thickness">
    <value id="thick"/>
    <value id="medium"/>
    <value id="thin"/>
  </parameter>
</category>
<category id="beam">
  <parameter type="oneOfVar" id="Aspect_Ratio">
    <value id="high"/>
    <value id="medium"/>
    <value id="low"/>
  </parameter>
  <parameter type="oneOfVar" id="size">
    <value id="big"/>
    <value id="medium"/>
    <value id="small"/>
  </parameter>
  <parameter type="oneOfVar" id="GeometricalComplexity">
    <value id="High"/>
    <value id="Medium"/>
    <value id="Low"/>
  </parameter>
  <parameter type="oneOfVar" id="Wall_Thickness">
    <value id="thick"/>
    <value id="medium"/>
    <value id="thin"/>
  </parameter>
</category>
<category id="rotation_figure">
  <parameter type="oneOfVar" id="Aspect_Ratio">
    <value id="high"/>
    <value id="medium"/>
    <value id="low"/>
  </parameter>
  <parameter type="oneOfVar" id="size">
    <value id="big"/>
    <value id="medium"/>
    <value id="small"/>
  </parameter>
  <parameter type="oneOfVar" id="GeometricalComplexity">
    <value id="High"/>
    <value id="Medium"/>
    <value id="Low"/>
  </parameter>
  <parameter type="oneOfVar" id="Wall_Thickness">
    <value id="thick"/>
    <value id="medium"/>
    <value id="thin"/>
  </parameter>
</category>
<category id="closed_shell">
  <parameter type="oneOfVar" id="Aspect_Ratio">
    <value id="high"/>
    <value id="medium"/>
    <value id="low"/>
  </parameter>
  <parameter type="oneOfVar" id="size">
    <value id="big"/>
    <value id="medium"/>
    <value id="small"/>
  </parameter>
  <parameter type="oneOfVar" id="GeometricalComplexity">
    <value id="High"/>
    <value id="Medium"/>
    <value id="Low"/>
  </parameter>
  <parameter type="oneOfVar" id="Wall_Thickness">
    <value id="thick"/>
    <value id="medium"/>
    <value id="thin"/>
  </parameter>
</category>
<category id="shell">
  <parameter type="oneOfVar" id="Aspect_Ratio">
    <value id="high"/>
    <value id="medium"/>
    <value id="low"/>
  </parameter>
  <parameter type="oneOfVar" id="size">
```

```
        <value id="big"/>
        <value id="medium"/>
        <value id="small"/>
      </parameter>
      <parameter type="oneOfVar" id="GeometricalComplexity">
        <value id="High"/>
        <value id="Medium"/>
        <value id="Low"/>
      </parameter>
      <parameter type="oneOfVar" id="Wall_Thickness">
        <value id="thick"/>
        <value id="medium"/>
        <value id="thin"/>
      </parameter>
    </category>
  </class>
</header>
```

BIBLIOGRAPHY

9. BIBILOGRAPHY

Allemang, D. and B. Chandrasekaran (1991). Functional Representation and Program Debugging. 6-th Annual Knowledge Based Software Engineering Conference.

Balder, J. and H. Akkermans (1992). "Formal Methods for Knowledge Modeling in the CommonKADS Methodology, A Compilation. " Netherlands Energy Research Foundation ECN(December).

Ben-Natan, R. (1995). CORBA: a Guide to Common Object Request Broker Archiecture, McGraw-Hill.

Bickerton, S. , H. C. Stadtfeld, et al. (1998). Active Control of Resin Injection for the Resin Transfer Molding Process. American Society for Composites Thirteenth Technical Conference, University of Maryland Press.

Bond, W. E. and M. Pegah (1993). "Automated Model Selection for Simulation Based on Relevance Reasoning. " IEEE Expert.

Bond, W. E. and M. Pegan (1993). "Representation and Reasoning about the Fuel System in the McDonnel Douglas FA-18 from a Functional Viewpoint. " EEEI Expert **April**.

Brazier, F. M. T. , Wijngaards, N. J. E (1997). A Purpose-Driven Method for the Comparison of Modeling Frameworks. 7th Workshop on Knowledge Engineering: Methods and Languages (KEML '97), Open University, Milton Keynes, UK.

Brooks, R. A. (1986). "A Robust Layered Control System for a Mobile Robot. " IEEE Journal of Robotics and Automation **RA-2**(April): 14-23.

Brown, D. C. (1987). Routine Design Problem Solving. Knowledge Based Systems in Engineering and Architecture. J. Gero, Addison-Wesley.

Buchanan, B. G. and E. H. Shortliffe (1984). Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Cambridge, MA, Addison-Wesley.

Carver, N. and V. Lesser (1992). The Evolution of Blackboard Control Architectures. Amherst, University of Massachusetts.

Chandrasekaran, B. (1983). "Towards Taxonomy of Problem-Solving Types. " AI Magazine **4**(1): 9-17.

Chandrasekaran, B. (1993). The Functional Representation Language: A Framework for Reasoning, Ohio State University.

Chandrasekaran, B. and T. R. Johnson (1993). Generic Task And Task Structures: History, Critique and New Directions. Second Generation Expert Systems. J. P. K. J. M. David, and R. Simmons, Springer Verlag.

Chandrasekaran, B. and H. Kaindl (1996). Representing Functional Requirements and User-System Interactions. AAAI-96 workshop on Modeling and Reasoning about Function, Portland, OR.

Chandrasekaran, B. , S. Mittal, et al. (1979). An Approach to Medical Diagnosis Based on Conceptual Structures. IJCAI-6.

Chen, Y. and B. H. C. Cheng (1997). Formalizing and Automating Component Reuse. IEEE international conference on Tools with AI, Newport Beach, CA, IEEE Press.

Clancey, W. J. (1985). "Heuristic Classification. " Artificial Intelligence **XXVII**: 289-350.

Clancey, W. J. (1989). "Commentary on Sticklen's 'Problem Solving Architecture at Knowledge Level'. " Journal of Experimental and Theoretical Artificial Intelligence **2**.

Committee, A. H. , Ed. (1987). Engineered Material Handbook. Composites. Metal Park, OH, ASM International.

DARPA, P. H. (1998). HPKB Introduction, Teknowledge. **1999**.

de Kleer, J. (1977). Multiple Representations of Knowledge in a Mechanics Problem-Solver. IJCAI-77, Cambridge, MA.

El-Sheikh, E. , C. Penney, et al. (1997). Intelligent Tutoring for Polymer Composite Molding. 1997 Symposium on Low-Cost, High-Speed Polymer Composites Processing, Michigan State University, East Lansing, MI.

El-Sheikh, E. , J. Sticklen, et al. (1996). Neper Wheat: Integrating Expert Systems and Crop Modeling Technology. 6th International Conference on Computers in Agriculture, Cancun, Mexico, American Society of Agricultural Engineers.

Eriksson, H. , R. W. Fergerson, et al. (1999). Automatic Generation of Ontology Editors. Twelfth Banff Knowledge Acquisition for Knowledge-based systems Workshop, Banff, Alberta, Canada.

Erman, L. , F. Hayes-Roth, et al. (1988). The Hearsay-II Speech-understanding system: Integrating Knowledge to Resolve Uncertainty. Blackboard Systems. R. E. a. T. Morgan. Reading, MA, Addison-Wesley: 31-86.

Ferguson, P. , W. S. Humphrey, et al. (1997). "Results of Applying the Personal Software Process. " Computer(May): 24-31.

Fikes, R. , A. Farquhar, & J. Rice. (1997). Tools for Assembling Modular Ontologies in Ontolingua, Knowledge Systems Laboratory, Stanford University.

Finn, T. , Y. Labrou, et al. (1997). KQML as an Agent Communication Language. Software Agents. J. Bradshaw. Menlo Park, CA, AAAI Press.

Forbus, K. (1988). Qualitative Physics: Past, Present and Future. Exploring Artificial Intelligence. H. Shrobe, Morgan Kauffman: 239-296.

Fowler, M. and K. Scott (2000). UML Distilled. Reading, MA, Addison-Wesley.

Friedman-Hill, E. J. (1997). Jess, The Java Expert System Shell. Livermore, CA, Distributed Computing Systems at Sandia National Laboratories.

Genesereth, M. R. (1998). Knowledge Interchange Format draft proposed American National Standard (dpANS). San-Francisco, CA, Stanford University.

Genesereth, M. R. , R. E. Fikes, et al. (1992). Knowledge Interchange Format Version 3. 0 Reference Manual. Stanford, Ca, Stanford University.

Gennari, J. H. , D. E. Oliver, et al. (1995). A Web-Based Architecture for a Medical Vocabulary ServerServer. Nineteenth Annual Symposium on Computer Applications in Medical Care.

Ginsberg, M. L. (1991). "Knowledge Interchange Format: The KIF of Death. " AI Magazine(Fall 1997): 57-63.

Glaser, N. (1996). Contribution to Knowledge Acquisition and Modeling in a Multi-Agent Framework (The CoMoMAS Approach). Doctorat de l'Universite Henri Poincare, l'Université Henri Poincaré.

Gruber, T. , R. (1993). "A Translation Approach to Portable Ontologies. " Knowledge Acquisition 5(2): 199-220.

Gruber, T. R. (1992). Ontolingua: A mechanism to Support Portable Ontologies. , Knowledge Systems Laboratory, Stanford University,.

Hartman, J. and B. Chandrasekaran (1995). Functional Representation and Understanding of Software: Technology and Application. 5th Annual Dual-Use Technologies & Applications Conference, IEEE and Rome Lab, Utica, NY, IEEE Press.

adaptaion of FR to representation of software

Hawkins, R. , J. McDowell, K. ,, et al. (1993). Function-Based Modeling and Troubleshooting. AAAI Workshop on Reasoning about Function.

Hayes, P. (1979). The Naive Physics Manifesto. Expert System in the Microelectronic Age. D. Michie, Edinburgh University Press.

Hayes, W. and D. Zubrow (1995). Moving On Up: Data and Experience Doing CMM-Based Process Improvement. Pittsburg, Software Engineering Institute, Carnegie Mellon University.

Hayes-Roth, F. , L. D. Erman, et al. (1992). Domain-Specific Software Archittectures: Distributed Intelligent Control and Communication. SEI at Carnegie Mellon University, SEI at Carnegie Mellon University.

Hidlum, D. W. , N. M. Sadeh, et al. (1996). Mixed-Initiative Management of Integrated Process-Planning and Production-Scheduling Solutions. Artificial Intelligence in Manufacturing Research Planning workshop, Albuquerque, NM, AAAI Press.

Honeywell (1999). What are the Benefits of Using a DSSA?, Honeywell. **1999**.

Johnson, M. V. and B. Hayes-Roth (1987). Integrating diverse reasoning methods in BB1. Annual Conference of the American Association for Artificial Intelligence, Seattle, Wa.

Johnson, T. R. (1991). Generic Tasks in the Problem-Solving Space Paradigm: Building Flexible Knowledge Systems while Using Task Level Constraints. Columbus, Ohio State Univarsity: 166.

Kamel, A. , O. Lukibanov, et al. (1997). A Task Specific Architecture for Conceptual Fabrication Sequence Planning for Structural Assemblies made from Composite Materials. Interfaces 1997, Montpellier, France.

Kamel, A. , J. Sticklen, et al. (1989). A Model-Based Approach for Organizing Quantitative Computations. Model-Based Diagnosis workshop, Paris, France, AAAI Press.

Keuneke, A. , M. (1989). Machine Understanding of Devices Causal Explanation of Diagnostic Conclusions. Computer Science Department. Columbus, Ohio State University: 103.

proposed to make, to maintain, to prevent function

Kuipers (1986). "Qualitative Simulation. " Artificial Intelligence(29): 289-338.

Laird, J. E. , A. Newell, et al. (1987). "SOAR: An architecture for general intelligence.. " Artificial Intelligence 33(3).

Lambert, M. , B. Riera, et al. (1997). Application of Some Functional Analysis Techniques on a Nuclear Reprocessing System. 5th International Workshop on Functional Modeling of Complex technical Systems, Paris-Troyes, France, The Center of Technology Risk Studies at University of Marilnd College Park.

Lenat, D. and Guha (1990). Building Large Knowledge-Based Systems, Addison Wesley.

Lenz, T. , M. Hawley, et al. (1998). "Virtual Prototyping in Polymer Composites. " Journal of Thermopplastic Composite Materials 11: 394-416.

Lenz, T. , J. K. McDowell, et al. (1996). "The Evolution of a Design Support Architecture for Polymer Composite Design. " IEEE Expert Intelligent Systems and Their Application 11(5): 77-83.

Liver, B. and D. Allemang (1995). "A Functional Representation for Software Design. " International Journal of Software Engineering and Knowledge Engineering 5(2): 227-269.

Lossak, R. , S. ,, M. Yoshioka, et al. (1998). A Comparative Analysis of function modeling in the design systems DIICAD-Entwurf and FBS/KIEF-System. Functional Modeling and Teleological reasoning Work shop at AAAI-98, Madison, WS, AAAI Press.

Lukibanov, O. and I. Martinez (2000). " Socharis: The Instantiation of a Strategy for Conceptual Manufacturing Planning. " Journal of Aritficial Intelligence in Engineering Design and Manufacturing(Fall).

Lukibanov, O. , I. Martinez, et al. (1998). Metal to Composites Structural Assemblies: Developing Appropriate Functional Modeling Frameworks for Static Analysis. FM/TR Workshop at AAAI-98, WI.

MacGregor, R. (1991). The Evolving Technology of Classification-Based Knowledge Representation Systems. Principles of Semantic Networks: Exploration in the Representation of Knowledge. J. Sowa, Morgan Kaufmann.

Martinez, I. , O. Lukibanov, et al. (1999). Augmenting Conceptual Design with Manufacturing: an Integrated Generic Task Approach. DETC-99/DFM99, Las Vegas, NV, ASME.

Martinez, I. , O. Lukibanov, et al. (1998). Function-Based Modeling of Fabrication Plans for Structural Assemblies. Special Interest Group in Manufacturing Workshop: State of the Art and State of Practice (SIGMAN-98), Albuquerque, New Mexico, AAAI, Press.

McDermott, J. (1988). "Preliminary Steps Towards a Taxonomy of Probglem Solving Methods. " Automated Knowledge Ackuisition for Expert Systems.

McDowell, J. K. , J. Sticklen, et al. (1997). Conceptual Design of Manufacturing Sequences for Composite Assemblies. ACCE, Dearborn, MI.

Mettala, E. , M. Graham (1992). The Domain-Specific Software Architecture Program. Pittsburg, PA, SEI at Carnegie Mellon University.

Mili, H. , F. Mili, et al. (1995). "Reusing Software: Issues and Research Directions. " IEEE Transactions on Software Engineering **21**(6): 528-561.

Modarres, M. (1998). Functional Modeling of Physical Systems Using the Goal Tree-Success Tree Framework. Functional Modeling and Teleological reasoning Work shop at AAAI-98, Madison, WS.

Motta, E. , K. O'Hara, et al. (1994). A VITAL Solution to the Sisyphus II Elevator Design Problem. 8th Knowledge Acquisition for Knowledge-Based Systems Conference, Banff, Canada.

Moy, B. , J. K. McDowell, et al. (1995). Integrated Design and Agile Manufacturing in Polymer Matrix Composites: The Role of Intelligent Decision Support Systems. SAMPE-1995, CA.

Murdoc, J. and A. Goel (1998). A Functional Modeling Architecture for Reflecting Agents. Functional Modeling and Teleological reasoning Work shop at AAAI-98, Madison, WS, AAAI Press.

Newell, A. (1980). The Knowledge Level (Presidential Address). AAAI-1980, Stanford, CA, AAAI.

Pegah, M. , W. E. Bond, et al. (1992). "Representing and Reasoning about the Fuel System of the McDonnell Douglas F/A-18 from a Functional Perspective. " IEEE Expert: in press.

Pegah, M. , R. Hawkins, et al. (1994). Functional Modeling using Standard Parts: Supporting Conceptual Design. AAAI Workshop on Functional Reasoning, Seattle, Washington.

Price, C. J. (1996). "Identifying Sneak Paths though Function. ".

Punch, W. F. (1989). A Diagnostic System Using A Task Integrated Problem Solver Architecture (TIPS), Including Causal Reasoning. Computer Science Department. Columbus, Ohio State University.

Riley, G. (1998). CLIPS a Tool for Building Expert Systems, Gary Riley. **1999**.

Schach, S. R. (1997). Software Engineering with JAVA. Chicago, Richard D. Irwin, a Times Mirror Higher Education Group Inc.

Schapire, R. , Y. Singer, et al. (1998). Boosting and Rocchio Applied to Text Filtering. SIGIR'98, Melbourne, Australia.

Schapire, R. E. and Y. Singer (1998). Improved Boosting Algorithms Using Confidence-rated Predictions. COLT-98, MAdison, WS, AAAI press.

Sembugamoorthy, V. and B. Chandrasekaran (1986). Functional Representation of the Devices and Compilation of Diagnostic Problem Solving Systems. Hillsdail, NJ, Lawrence Erlbaum Associates.

Slator, B. M. (1989). "Decomposing Meat: a Commentary on Sticklen's 'Problem Solving Architecture at Knowledge Level'. " Journal of Experimental and Theoretical Artificial Intelligence **2**.

Steels, L. (1990). "Components of Expertise. " AI Magazine(Summer): 28-49.
compares generic task approach to KAD

Stevens, W. P. , G. J. Myers, et al. (1974). "Structured Design. " IBM Systems Journal **13**(2): 115-139.

Sticklen, J. (1987). MDX2: An Integrated Medical Diagnostic System, Ohio State University.
University Microfilm Number 87-17732

Sticklen, J. (1989). "Problem Solving Architectures at the Knowledge Level. " Journal of Experimental and Theoretical Artificial Intelligence **1**: 1-52.

Sticklen, J. and B. Chandrasekaran (1985). Use of Deep Level Reasoning in Medical Diagnosis. Government Symposium in Expert Systems.

Sticklen, J. , A. Kamel, et al. (1991). "Integrating Quantitative and Qualitative Computation in a Functional Framework. " Engineering Applications of Artificial Intelligence **4**.

Sticklen, J. , A. Kamel, et al. (1992). "Fabricating Composite Materials: A Comprehensive Problem Solving Architecture Based on a Generic Task Viewpoint. " IEEE Expert **7**(2): 43-53.

Sticklen, J. , A. Kamel, et al. (1992). An Artificial Intelligence-Based Design Tool for Thin Film Composite Materials. East Lansing, Michigan State University.

Sticklen, J. and R. Tufankji (1992). "Utilizing a Functional Approach for Modeling Biological Systems. " Mathematical and Computer Modeling **16**: 145-160.

Stroulia, E. and P. Sorenson (1998). Supporting Software Redesign: Functional Reasoning meets Meta-Case tools. FM/TR Workshop at AAAI-98, Madison, Wisconsin, AAAI Press.

Tsumoto, S. and H. Tanake (1995). "Interpretation of Medical Laboratory Data Based on Functional Model. " .

Tu, S. W. , H. Eriksson, et al. (1995). "Ontology-based configuration of problem-solving methods and generation of knowledge -acquisition tools: Application of PROTEGE-II to protocol-based decision support. " Artificial Intelligence in Medicine **7**: 257-289.

Tu, S. W. , M. G. Kahn, et al. (1989). "Episodic skeletal -plan refinement based on temporal data. " Communications of the ACM **32**(12): 1439-1455.

Uschold, M. and M. King (1995). Towards a Methodology for Building Ontologies. IJCAI-95.

van Heijst, G. , A. T. Schreiber, et al. (1997). "Using Explicit Ontologies in KBS Development. " Int. Journal Human-Computer Studies **45**: 183-292.

Velde, W. V. d. (1994). A Constuctivisst View on Knowledge Engineering ECAI 94. A. Cohn, Wiley & Sons, Ltd.

Velde, W. V. d. (1991). "Tractable Rationality at the Knowledge Level. " .

Velde, W. V. d. (1993). Issues in Knowledge Level Modelling. Second Generation Expert Systems. J. -M. David, J. -P. Krivine and R. Simmons. Berlin, Springer Verlag.

Vranes, S. and M. Stanojevic (1995). "Integrating Multiple Paradigms within the Blackboard Framework. " IEEE Transactions on Software Engineering **21**(3): 244-262.

Wang, E. Y. and B. H. C. Cheng (1998). Formalizing and Integrating the FunctionalModel into Object Oriented Design. 10th International Conference on Software Engineering and Knowledge Engineering, San Francisco.

Wang, E. Y. , H. A. Ritcher, et al. (1997). Formalizing and Integrating the Dynamic Model within OMT. IEEE International Conference on Software Engineering.

Wielinga, B. J. and A. T. Schreiber (1994). Conceptual modeling of large reusable knowledge bases. Berlin, Germany, Springer Verlag.

Zhou, K. , T. J. Lenz, et al. (1999). A Problem Solving Architecture for Virtual Prototyping in Metal to Polymer Composite Redesign. ASME-99, Las Vegas, NV.