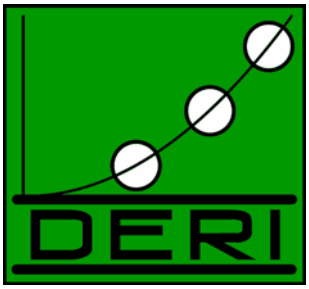
Digital Enterprise Research Institute



Survey on Goal-driven Architectures

Michael Stollberg and Florian Rhomberg
DERI Austria

Technical Report
DERI-TR-2006-06-04
June 4, 2006

DERI Galway
University Road
Galway
Ireland
www.deri.ie

DERI Innsbruck
Technikerstrasse 21a
A-6020 Innsbruck
Austria
www.deri.at

DERI Seoul
Yeonggun-Dong, Chongno-Gu
Seoul
Korea
www.deri.org

DERI Stanford
Serra Mall
Stanford, CA
USA
www.deri.org

DERI-TR-2006-06-04

Survey on Goal-driven Architectures

Michael Stollberg and Florian Rhomberg
DERI – Digital Enterprise Research Institute
University of Innsbruck, Austria

Abstract: The ultimate aim of advanced IT technology is to provide infrastructures for automated problem solving. Based on machine-readable descriptions, intelligent mechanisms shall enable dynamic usage and combination of available computational resources for solving problems. Therefore, different branches of Artificial Intelligence (AI) research develop frameworks for describing problems and resources along with technologies for automated problem resolution and resource usage. Most commonly, such frameworks center around the *service-side* that is concerned with how to describe computational resources in order to allow automated usage, and the *client-side* that is concerned with how to describe the problems to be solved and their resolution process. In order to provide IT technology that reflects real world problem solving in a sophisticated manner, the service- and the client-side should be decoupled to the highest possible extent. Appropriate models for the client-side should allow specifying objectives to be solved from the client's perspective without regard to their technical resolution, thereby providing sophisticated support for the client-side that is decoupled from technical service usage requests; on the other hand, client-side description elements should encompass all information required for automated problem resolution by intelligent mechanisms for automated resource detection, combination, and usage. Such sophisticated client-side models is what we refer to as goal-driven architectures. Therein a user only specifies the objective or problem to be solved while intelligent mechanisms handle the resolution process automatically. This paper surveys approaches for goal-driven architectures, deriving the state of the art on description models for the client-side that is intended to serve as a basis for developing a goal-driven architecture for Semantic Web Services.

Keywords: Goals, Automated Problem Solving, Cognitive Architectures, BDI Agents, AI Planning, UPML, Goal Types, Goal Resolution Techniques

TABLE OF CONTENTS

INDEX OF FIGURES

INDEX OF TABLES

1 Introduction

Research on Artificial Intelligence (AI) is concerned with the creation of intelligent computer systems that perform tasks automatically. Aiming at sophisticated support for automated problem solving, we observe two elementary methodologies that are collectively followed throughout the various sub-disciplines of AI research: first, logics and formal methods are applied in order to enable advanced processing of complex information, and secondly that system and technology design is founded on models for problem solving in the real world [McCarthy, 1989]. This has lead towards several foundational paradigms for computer technology design, whereof the following have significantly influenced IT technology development (in chronological order): the *relational algebra* providing the foundation for the broadly used relational database technology [Codd, 1972], the *object-oriented paradigm* as a programming technique that overcomes the deficiencies of procedural programming languages by means of data abstraction, polymorphism, encapsulation, and inheritance [Meyer, 1997], the *agent-oriented paradigm* that develops systems wherein agents reside as autonomous computational elements and satisfy their particular objectives in an interactive manner [Shoham, 1993], and *service-orientation* as the most recent paradigm that proclaims system architectures consisting of several service that encapsulate some computational facility and are dynamically used and combined for solving a specific problem [Erl, 2004].

The ultimate aim of AI research – contemporaneously its motivation and initiation in the second half of the 20th century – is to create computer systems, or, more generally, machines that can solve problems in a similar way as human beings do [Turing, 1950]. Several works have been and still are concerned with defining appropriate frameworks for advanced, intelligent, and automated problem solving by following the above mentioned methodologies of studying problem solving in nature and applying logic-based techniques for simulating this. When analyzing the organization of such frameworks, we can identify the following four common top level elements: (1) the *service-side* that is concerned with how to describe computational resources in order to allow automated usage; (2) the *client-side* that is concerned with how to describe the problems to be solved and their resolution process. The elements of both the service- and the client-side are formally described in order to enable automated problem resolution by (3) *intelligent mechanisms*, which is supported by (4) *auxiliary elements*.[1] This paper concentrates on the client-side, investigating the requirements and the state of the art of so-called *goal-driven architectures*. Therein, a user shall

[1] This terminology has been chosen with respect to service-orientation as the most recent design paradigm for next generation IT systems. An example of such a framework is the Web Service Modeling Framework WSMF [Fensel and Bussler, 2002] with the Web Service Modeling Ontology WSMO as its successor [Lausen et al., 2005] that defines four top level elements: Web Service for the service-side, Goals for the client-side, along with Ontologies and Mediators as auxiliary elements.

only specify the objective or problem to be solved and intelligent mechanisms automatically handle the resolution process by appropriate computational facilities.

As the term 'Goal-driven Architecture' is rather vague and undefined in literature, we briefly explicate our understanding as well as the arising research questions on basis of the following example. A user *U* wants to book a one week holiday by using an advanced IT system *S* that provides computational facilities for automated travel and tourism related booking. *U* has some further constraints on the holiday package to be booked, for example: the destination should offer a beach and allow swimming in an ocean, but preferably not be in an Islamic country with respect to current political dangers; he also wants to book a scuba diving package, and the accommodation should not be located in the center of a city or village. The system *S* should allow *U* to specify his objective along with the constraints, while *S* should be able to automatically detect, arrange, and utilize available computational facilities for solving the objective – similar to the service offered by a real-world travel agency. The main merit of such systems is that they bridge the gap between the human and the machine level problem solving. While *U* only needs to specify an objective or goal to be reached, a goal-driven system *S* is capable of achieving this objective by automatically utilizing appropriate resources as if *U* was dealing with another human being. Obviously, sophisticated models for the client-side that allow specifying user objectives and carry all information needed for automated goal resolution are the central requirement for realizing goal-driven technology. In conjunction with appropriate definitions for the other three top level elements mentioned above, goal-driven architectures can realize the aim of "human level machine intelligence" [McCarthy, 1996].

Modeling of client-side elements as the core of goal-driven architectures raises several questions, including: how to appropriately specify user objectives? What elements are needed in order to provide all information that is needed for automated detection, combination, and usage of computational facilities by respective intelligent mechanisms? What is the difference and benefit of goals in comparison to technical usage requests? How can arbitrarily complex objectives be specified, and how can their resolution process be effectively supported? Although a commonly accepted framework for goal-driven architectures does not exist, related approaches have been developed in respective sub-disciplines of AI – namely Knowledge Engineering, Intelligent Agents, and AI Planning; also, the promise of Web Services seems to require goal-driven technology [Fensel and Bussler, 2002], [Preist, 2004]. The aim of this paper is to determine the requirements for goal-driven architectures and investigate existing approaches from respective AI research as the basis for developing a goal-driven architecture for Semantic Web Services.

The paper is structured as follows: Section 2 examines the motivation and aim of goal-driven architectures and rationalizes the working approach followed in the paper; Section 3 to 6 examine goal-driven approaches in respective AI disciplines (namely: the Soar technology in Section 3, Agent technology in Section 4, AI Planning in Section 5, and Problem Solving Methods in Section 6); finally, Section 7 summarizes the findings and concludes the paper.

2 What are Goal-driven Architectures?

As a foundation for the subsequent investigations, the following exposes the motivation for goal-driven architectures and determines the requirements for these. Besides, in order to rationalize the approach followed throughout this survey, we discuss the differences between goal-driven and not-goal-driven architectures, and finally outline the aspects of interest as well as the methodology for examining existing goal-driven approaches.

2.1 Motivation for Goal-driven Architectures

As stated introductory, the ultimate aim of modern research on Artificial Intelligence is to develop technologies for human level machine intelligence in order to enable advanced automated problem solving. While the well-known Turing Test defines a benchmark for machine intelligence at a very early point in time – stating that a machine is considered to be intelligent if its user can not distinguish whether he interacts with a machine or a human [Turing, 1950] – approaches for achieving this aim have been developed in the following decades. Thereof, we find the theoretical basis and motivation for goal-driven architectures as the main research results of so-called *Cognitive Science*, an interdisciplinary field of research that aims at understanding the human mind and intelligence as the basis for creating intelligent systems [Nadel, 2003], [Wilson and Keil, 1999]. Cognitive Science follows the above mentioned general AI methodology of studying structures and processes in nature as the basis for simulating them by intelligent technology. Commencing in the 1970ies as basic AI research, it has produced an impressive compilation of results that serve as the philosophic-theoretical foundation of several AI technology developments.[2]

The relevant findings with respect to the motivation for goal-driven architectures are allocated in the field of *problem solving* – which itself is one of the core aspects of Cognitive Science as almost all cognitive activities can be regarded as problem solving. Problem solving is understood as a part of human thinking concerned with how to reach an objective from the current status of the world when the procedure therefore is not known a priori. In order to provide a basis for simulating intelligent behavior, the aim of research in Cognitive Science is to expose the generic mechanisms that humans apply for problem solving, wherefore the fundamental theory has been provided in [Newell and Simon, 1972]. Although rightly being criticized for reducing the conceptual model of mind to be presentable as an information processing system, human problem solving is defined as a goal-oriented activity for finding some

[2] Exhaustive synopses on the various branches and research results of Cognitive Science are provided in [Nadel, 2003], [Wilson and Keil, 1999], and on the Internet (e.g.: in the Stanford Encyclopedia of Philosophy http://plato.stanford.edu/entries/cognitive-science/). Most capacious work is subsumed in the Soar project (a world-wide initiative developing a cognitive system architecture for exhibiting intelligent behavior, based on [Newell, 1990], see homepage: http://sitemaker.umich.edu/soar), as well as in awarded books like [Hofstadter, 1979].

possible sequence of operators that allows proceeding from the initial state of the problem space to the goal state. A goal is understood as a desired state that is to be reached from the current state wherefore an applicable sequence of operators is not known; an operator is an activity or a process that performs the transition from one state to another in the problem space; the problem space is a potentially infinite number of states that can be reached by operators. Humans apply specific psychological techniques for problem solving [Anderson, 1999]: discovery, notification, or observation for becoming aware of operators that can be used in a specific state, and so-called *means-end analysis* as the key mechanism for choosing the most appropriate operator out of those available and applicable in a state. Roughly speaking, in means-end analysis the current state is compared to the goal state, a difference is determined between them, and the operator that can reduce this difference to the maximum extent in comparison to other available operators is chosen. This has been prototypically realized in the General Problem Solver as pioneer work in intelligent system development [Newell and Simon, 1963].

This model of problem solving – which, although being very basic, is still considered a valid theoretical basis of AI technology development – provides the following aspects with regard to the motivation and design of goal-driven architectures. First, the concept of goals as final states of the world that is to be reached in order to solve a problem along with the concept of goal-orientation, stating that all activities are performed rationally in order to solve a goal; secondly, the concept of operators that allow changing the current status of the world (which can be performed automatically or manually); and thirdly, that problem solving, or goal resolution, is realized by determining a possible or optimal execution sequence of operators wherefore generic strategies are applied. This complies with the understanding and aim of goal-driven technology that we explore in this paper. While the user of a system should only need to specify a goal as a desired final state to be achieved, the system should employ intelligent mechanisms for detecting and utilizing appropriate computational resources as the operators for the goal resolution process, thereby simulating human problem solving strategies. Hence, we retain that the idea of goal-driven architectures as pursued in this paper can be seen as candidate approach for realizing human level machine intelligence, following general AI methodologies.

Two other aspects are relevant for goal-driven architectures. First, they reside on the so-called *knowledge level* as illustrated in [Newell, 1982].[3] The knowledge level is concerned with actions, effects, and behavior in the world (i.e. the environment that operations take place in); beneath this, the 'symbol level' is concerned with mechanisms and operations for actually executing actions. Goal-driven architectures are mainly concerned with the Why and How of goal resolution behavior, neglecting technical implementations on the symbol level – certainly, both levels are interconnected and both need to be addressed in order to realize goal-driven technology.

[3] We primarily refer to works of Allen Newell here. This does not result from inadequate research but from the fact that he has been a leading pioneer of AI research and especially an originator of Cognitive Science, see: http://stills.nap.edu/readingroom/books/biomems/anewell.html.

However, the main aspects of interest reside on the knowledge level. The second aspect is that goal-driven architectures shall allow bridging the gap between human-level intelligence and automated information processing by machines. Goals allow specifying objectives to be achieved on a higher level of abstraction; they are dynamically connected to appropriate operators for automated resolution by intelligent mechanisms that realize psychological methods for problem solving. This establishes the connection between the knowledge and the symbol level as a crucial task of IT system design and usage. Because of this, goal-orientation is proposed to become a new software engineering paradigm [Lamsweerde and Letier, 2002].

2.2 Requirements on Goal-driven Architectures

On basis of the preceding examinations, the following explicates the idea of goal-driven architectures that underlies this work and depicts requirements on these.

Figure 1 shows a course model of a goal-driven architecture that serves as a working hypothesis in this paper. As outlined introductory, we distinguish four top level elements for enabling automated goal-driven problem solving: (1) the *client-side* contains elements for supporting automated goal resolution from the user perspective, (2) the *service-side* contains the descriptions and implementations of operators for problem solving that typically are computational facilities,[4] (3) *intelligent mechanisms* as the facilities for enabling automated goal resolution by working on the formal descriptions of the client- and the service-side, and (4) *auxiliary elements* needed for automated goal resolution (e.g. machine-processable knowledge definitions).

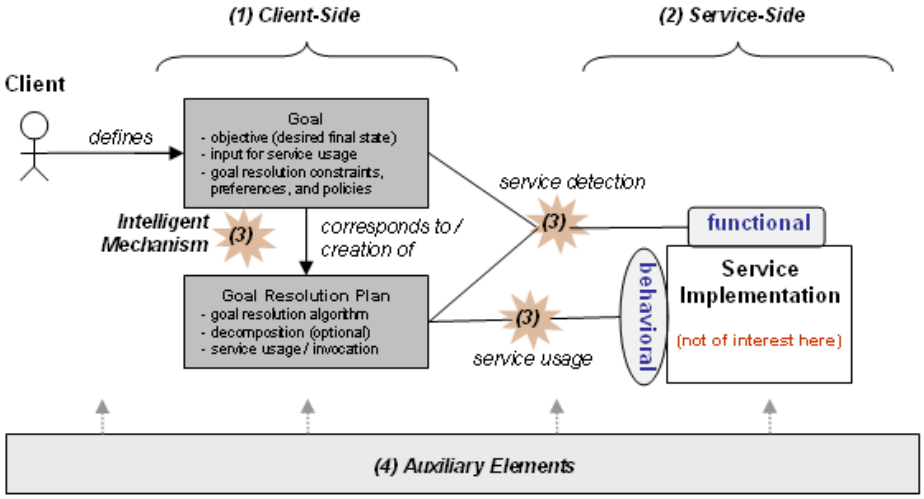


Figure 1: Abstract Model of Goal Driven Architectures

[4] As stated above, we use the term 'service-side' with respect to service-oriented architectures as the most recent design paradigm for IT systems (see Introduction). We could also call it the 'operator-side' with regard to the element classification in classical problem solving as discussed in Section 2.1; however, we consider this course model to be generally applicable for goal-driven architectures independent of particular technical environments.

The primary aspect of interest in goal-driven architectures is the client-side. This needs to provide appropriate elements for supporting problem solving from the user perspective on the knowledge level that shall be automated to the highest possible extent. The service-side needs to provide the formal descriptions of the operators available for problem solving, which commonly consist of a functional description (what the service does) and a behavioral description (how the service works, especially how to communicate with the service in order to consume its functionality). For the client-side, the figure distinguishes two elements: *Goals* shall encompass the objective or problem specification by the user, input for using services as automated operators for problem solving, and constraints and preferences the user defines for goal resolution; the *Goal Resolution Plan* shall encompass the procedure for goal resolution (called the goal resolution algorithm), potentially problem decomposition with regard to available services, and facilities for automated service invocation and usage (which is needed for automated goal resolution by execution of services by operator usage on the symbol level). This distinction is made with regard to two purposes: first, the user should only have to define those aspects summarized in *Goals* – see the introductory example of booking a holiday – and secondly the elements contained in the *Goal Resolution Plan* should be determined automatically by respective intelligent mechanisms with respect to available services and the problem solving process during system runtime.

This abstract model of goal-oriented architectures is a working hypothesis that is to be verified by the preceding examinations. Nevertheless, we can determine six requirements for technologies as aspired here:

1) the concept of a **Goal** should allow specifying (client) objectives or problem as the *desired final state to be reached*
2) goal definitions can be **accompanied by additional constraints** that are *defined on the knowledge level*
3) goal resolution should be **automated to the highest possible** extent by enforcing *clients to only provide the least possible amount of information* required
4) the **Goal Resolution Plan** should be determined *automatically*; therefore, client-side elements (as all other elements) need to have an **unambiguous formal description**
5) goal resolution techniques should be able to **determine the optimal Goal Resolution Plan** for a given goal or problem
6) specification and handling of **arbitrary complex goals** should be supported by means of *problem decomposition*.

2.3 Goals versus Service Usage Requests

The main element of goal-driven architectures is the concept of goals that shall allow specification of client objectives on the knowledge level without respect to their technical resolution. In contrast, several technologies provide support for the client-side by only mirroring or copying the structure of service-side elements. This is what we call *service usage requests* that contradict the fundamental idea of goal-driven architectures as we discuss in the following.

For clarifying what we mean by service usage requests in comparison to goals let's consider a simple example of retrieving address information of all customers that live in Innsbruck. If the data repository is a conventional RDB, we would define a SQL query like "`SELECT * FROM customers WHERE address.city = 'Innsbruck'`". Here, if we understand the database as the service-side, SQL provides a query mechanism for the client-side that mirrors the structure of the database for retrieving the desired information. In contrast, a goal-based technology would allow specifying the client objective of retrieving the desired information in a more abstract, intuitive way that is decoupled from the underlying technology. In fact, appropriate support for the client-side that allows definition of goals and their automated resolution is missing here.

We observe that several broadly used software engineering technologies as well lack of client-side support as aspired in goal-driven architectures. For example, the Common Object Request Broker Architecture CORBA (see homepage: http://www.corba.org/) as platform-independent infrastructure for distributed, object-based computing allows access to implementations via so-called *Object Request Brokers* (ORB) that handle the physical invocation of object implementations on basis of the standardized Interface Description Language IDL. A client provides a proxy that needs to contain all information required for invoking and using an implementation (the IDL stub); the implementation is accessed via an IDL skeleton as a complementary description which is grounded to the respective implementation technology. Here, as in the simply example above, the client-side in form of an IDL stub mirrors the structure of the IDL skeleton as service-side but does not offer support for goals. Similar architectures are applied within current Web Service technologies. Therein, in order to use a WSDL described Web Service, we have to create a mirror of the service-side WSDL description in order to be able to invoke and communicate with the Web Service. This is supported by Web Service development environments like Apache Axis (see homepage: http://ws.apache.org/axis/). These technologies provide the technical basis for enabling component-based, distributed computing, also over the Internet – but they do not encompass appropriate support for the client-side in order to support system developers or (human) end users.

Several techniques have been developed in order to ease the development of software systems. Formal methodologies like the Z specification language [Diller, 1994] or Abstract State Machines [Börger and Staerk, 2003] provide supportive means for large scale software development by interrelated formal descriptions from very abstract levels down to executable code generation. Although used as the basis for goal-

oriented software engineering [Lamsweerde, 2003], these techniques do not support goal-driven architectures for automated problem solving as pursued here. So-called *Software Reuse* [Krueger, 1992] aims at supporting software creation out of existing implementations rather then development from the scratch. Commencing in the 1970ies, several techniques have been developed for classifying and retrieving reusable software from component libraries and support for integrating them into other systems. Formal methods are applied for describing and handling software component libraries [Jeng and Cheng, 1993] along with formal retrieval techniques that include early work on semantically enable specification matchmaking (e.g. [Chen and Cheng, 2000]). Although these technologies provide a basis for ongoing research for discovery in service-oriented architectures, they are mainly considered with formal descriptions of the service-side but do not encompass appropriate notions or support for the client-side as aspired in goal-driven architectures.

What we are interested in is how goals and related client-side elements are formally described, and which mechanisms are applied for automated goal resolution. The motivation for this is to provide a basis for developing a goal-driven technology for Semantic Web Services in order to allow knowledge level problem solving over the Web. When analyzing current architectural models for Web Services, we notice that such technology is demanded for realizing the vision of the Semantic Web. The W3C Web Service Architecture remarks the concept of goals to be relevant without any further explanation [Booth at al., 2004]; OWL-S only provides a description ontology for the service-side, whereby client-requests are described as service profile definitions from the client perspective [Paolucci et al., 2002]; although the Web Service Modeling Ontology WSMO defines the notion of Goals as a top level notion, these are described by *requested capabilities* and *requested interfaces* and thus as well are service-side descriptions from the client perspective [Lausen et al., 2005]. Hence, as the support for the client-side in both OWL-S and WSMO as the most significant approaches for Semantic Web Services are service usage requests as they simply mirror and copy the structure of their respective service-side descriptions, they can not be considered to be sophisticated goal-driven architectures.

2.4 Surveying Goal-driven Approaches

The preceding examinations reveal that we can omit works related to the above mentioned technologies for investigating the state of the art in goal-driven architectures. Instead, we concentrate on specific approaches from respective AI research fields. The main aspects that we are interested in are:

1) How are goals and related client-side elements defined and described, and what is there interrelation?
2) What techniques are applied for automated, goal-driven problem solving?
3) What kind of goals and problems can be handled by the technology?
4) Which aspects and solutions appear to be useful with respect to goal-driven architectures as pursued in this paper?

Therefore, those sub-disciplines out of the numerous in AI research appear to be of interest that are concerned with applying basic AI techniques for creating intelligent system architectures. Investigation has revealed that works from the following working fields are relevant for this: *Cognitive Architectures* that aim at realizing intelligent systems on basis of cognitive models of the human mind and intelligence [Anderson, 1991], *Intelligent Agents* that develop at intelligent systems wherein autonomous agents reside that satisfy their particular objectives by interacting in a collaborative manner [Russell and Norvig, 2003], and *Knowledge Engineering* that is concerned with developed of techniques and systems for advanced, knowledge-based information processing [Studer et al., 1998].

Out of these, we have chosen particular approaches and technologies that are concerned with client-side element specification and techniques for automated, goal-driven problem solving. At first, we investigate the Soar technology that develops a cognitive architecture on basis of the results from Cognitive Science we have discussed as the motivation for goal-driven architectures. Then, we inspect goal-driven techniques from Intelligent Agent research, focusing on so-called belief-desire-intention (BDI) architectures and multi-agent collaboration theories as the core techniques for goal-based agent behavior. As the third approach, we examine how advanced notions of goals are defined and used in AI Planning in order to allow automated construction of goal resolution plans for more complex problems, and finally we inspect the Unified Problem Solving Method Development Language UMPL, a framework for describing the reasoning behavior of knowledge-based systems for automated problem solving.

For surveying each approach, we first outline the aim and origin, then explain the main elements and techniques for automated goal-driven problem solving, and finally conclude the usability and contributions for goal-driven architectures. Thereby, we aim at gaining a general synopsis and understanding of goal-driven technologies; hence, we concentrate on the principal approaches while referring to continuative resources for details on specific technologies.

3 The Soar Technology

The Soar system provides a cognitive architecture by implementing the conceptual model of human cognition presented in [Newell, 1990] –the final book and life's work of AI-pioneer Allen Newell towards a technical architecture for intelligent machine behavior based on an integrated theory of human cognition. When initiated in the 1970ies, Soar was the acronym for **S**tate, **O**perator, **A**nd **R**esult as the core elements of a cognitive architecture based on theoretical models for human mind and intelligence, but it became a designation of its own in still ongoing research and development work with numerous participants from all over the world (see Soar project homepage for further information and software: http://sitemaker.umich.edu/soar.).

In essence, the Soar architecture is a *production system* that represents the structure of human cognition and problem solving that has been determined by Cognitive Science research. The core elements of the architecture are *problem spaces* that represent tasks by *states*, *operators*, and *goals*; problem solving is performed by a *decision cycle* that processes different *memory types* for reaching a goal that is defined as a final state in some problem space. Two integrated components allow enhanced intelligent problem solving: so-called *impasse handling* that realizes automated subgoaling for not-resolvable problems, and *chunking* that allows learning in order to improve the system's problem solving capabilities. The basic Soar architecture has been extended by several associative technologies and has been applied as a basis for several other AI working fields in order to test and demonstrate its applicability as a generic architecture for intelligent behavior. While referring to [Rosenberg et al., 1993] as a 2-Volume collection of collected research publications on Soar, the following explains the core architecture in more detail.

3.1 The Basic Architecture

As outlined above, the core of the Soar architecture is a production system wherein the decision cycle realizes automated problem solving by working on problem spaces with different memory types. The following explains the central components, their specification and interplay in more detail.

3.1.1 Knowledge Items and Representation

A *problem space* contains the domain knowledge, states, available operators, and goals of a task or problem that can be solved automatically. Domain knowledge describes the entities involved in the problem by object-attribute-value definitions, while all other elements are described by production rules. A product rule has then general form: `IF (condition) THEN (action).` In order to ensure uniform knowledge representation, all knowledge in the Soar system is represented by production rules whose structure differs for specifying the distinct conceptual elements. A state denotes a status in the problem space that can be reached by an operator; state descriptions have an empty condition-part, while the action-part describes the status

of the world in the state by concrete values of attributes. Operators are applied to progress from one state to another, whereby the condition specifies conditions on the current state that need to hold for applying the operator and the action denotes the changes to the problem space that result from applying the operator. Goals are defined as final states in the problem space, meaning that the condition part describes the state of the problem space that is considered to be solving the goal while having an empty action-part (i.e. no further progress can be made in the problem space). Typically, a problem space consist of one goal, one or more initial states, and all operators that can be used. For illustration purpose, Listing 1 gives a brief example of a problem space for the well-known blocks-world problem.[5] It is defined in the Soar syntax, which itself is based on OPS5, an early production system language using forward chaining as its main inference mechanism [Forgy, 1981].

Listing 1: Soar Problem Space Definition Example[6]

```
Initial State
# blocks A, B, C on table; no block on top of another one
# incl. knowledge definitions (blocks, table, on-top-relation)
sp {blocks-world*elaborate*initial-state
(state <s> ^superstate nil)
-->
(<s> ^problem-space blocks
^thing <block-A> <block-B> <block-C> <table>
^ontop <ontop-A> <ontop-B> <ontop-C>)
(<block-A> ^type block ^name A)
(<block-B> ^type block ^name B)
(<block-C> ^type block ^name C)
(<table> ^type table ^name TABLE)
(<ontop-A> ^top-block <block-A> ^bottom-block <table>)
(<ontop-B> ^top-block <block-B> ^bottom-block <table>)
(<ontop-C> ^top-block <block-C> ^bottom-block <table>)}

Goal
# final state = tower: A on B, B on C, C on table
# action 'halt' denotes exit, i.e. no further
sp {blocks-world*detect*goal
(state <s> ^problem-space blocks
^ontop <AB> { <> <AB> <BC>} { <> <AB> <> <BC> <CT> } )
(<AB> ^top-block <A> ^bottom-block <B>)
(<BC> ^top-block <B> ^bottom-block <C>)
(<CT> ^top-block <C> ^bottom-block <T>)
(<A> ^type block ^name A)
(<B> ^type block ^name B)
```

[5] Problem: three blocks A. B, and C are on a table T, not being on top of each other. The aim is to build a tower with T on the bottom, C on top of T, B on top of C, and A on top of B (nothing on top of A). Allowed actions (or available operators) move one block on top of another one. This is an artificial problem commonly used to illustrate AI technology, see [Russell and Norvig, 2003].

[6] taken from [Laird and Congdon, 2004]; see also for syntax and semantics of the Soar language. Just to understand the example listing: *sp* means soar production, the part inside the curly brackets denotes the problem space name (here: blocks-world) and a natural language description, the condition is before -->, the action after -->; *^name* is an attribute followed by its value, *<name>* is a variable, <> means not equal in prefix notation; *x{y}* denotes that y is a complex sub-structure of x.

```
(<C> ^type block ^name C)
(<T> ^type table ^name TABLE)
-->
(halt)}

Operator
# condition: thing1, thing2 = blocks, thing1 not on-top of thing2,
#  no other on-top relation with thing1, thing2 as bottom-block
# action: create accept preference for operator o (denoted '+', s.b.),
#  o moves thing1 on top of thing2
sp {blocks-world*propose*move-block
(state <s> ^problem-space blocks
^thing <thing1> {<> <thing1> <thing2>}
^ontop <ontop>)
(<thing1> ^type block ^clear yes)
(<thing2> ^clear yes)
(<ontop> ^top-block <thing1> ^bottom-block <> <thing2>)
-->
(<s> ^operator <o> +)
(<o> ^name move-block
^moving-block <thing1>
^destination <thing2>)}
```

3.1.2 Memory Types and Problem Solving Mechanisms

While problem spaces encompass the knowledge items, three interrelated memory types (long-term memory (LTM), working memory, and preference memory) hold different information used in the central problem solving mechanism as explained below in more detail. In addition, the so-called *perception and motor behavior* allows Soar to interact with the external environment by retrieving additional input-information (perception) or creating new output information (motor) for external systems. This is realized by special types of productions stored in the LTM. These are processed independently of the central problem solving in the decision cycle, but can influence it immediately when new knowledge is perceived.

The *long-term memory* (LTM) holds general domain knowledge in from of productions as exemplified above. Although not explicitly specified, productions can perform 4 functions for problem solving: *operator proposal* and *operator comparison* by creating preferences for operators (see below), executing a state transition by *operator application*, and *state elaboration* by performing the action specified in a production. The productions in the LTM provide the information processing used for automated problem solving.

The *working memory* holds the current situation of the problem solving procedure. It consists of so-called WMEs (working memory elements) that represent the current state of problem solving and operators available in this state. Each WME is a triple of *identifier-attribute-value*: the identifier allows grouping of WMEs into objects (e.g.: a block named "A" with the identifier B1 that is on top of the table in a state s is an object represented by 3 WMEs: `B1 ^type block; B1 ^name A; <ontop> ^top-block B1 ^bottom-block TABLE`), and ensures unique identification of WMEs, respectively objects; attribute (denoted by `^attr-name`) define the slot for concrete values

(e.g. attribute "name" has value "A" in the above example). All WMEs in a state must be linked to each other, so that a state definition in the working memory is represented as a set of augmented identifier-attribute-value triples; WMEs that are not connected to any other WME in a state are removed from the working memory.

The working memory is manipulated by the so-called *decision cycle* as the central problem solving mechanism of the Soar architecture. The decision cycle consists of three main phases and is repeated until the goal of the current tasks is solved (i.e. a final state is reached in the problem space). Independent of this, the Soar system might interact with its external environment by the perception and motor function (newly received input data are incorporated in the decision cycle immediately, and output is created orthogonally). Sequentially executed, the three phases are:

1. **Elaboration:** LTM productions fire whose condition is satisfied, meaning that the new data are interpreted and the working memory is "elaborated" to the current state of the problem solving procedure. The created WMEs are considered to be I-supported, meaning that they have been created by instantiating productions with concrete values. Also, those I-supported WMEs that are no longer interlinked with others are removed from the working memory (retraction). All matching productions fire in parallel; the elaboration continues until quiescence (i.e. no more matching productions can be executed, and no more WMEs can be retracted).
2. **Decision:** then, a new operator is selected for application on the current state. Operator selection is based on preferences (see below). If a clear decision of operator usage can not be made, a so-called *impasse* is created (see next section for more details on impasse handling).
3. **Application:** the chosen operator is executed by firing the respective LTM productions for operator application. The WMEs created by operator application are called O-supported (in contrast to I-supported WMEs, O-supported WMEs are not removed from the working memory until the goal is reached as they define the constituting elements of the problem solving progress). After the application phase, a new state is created that is processed by repeating the cycle.

In essence, problem solving in Soar is realized as goal-directed application of most adequate operators in order to reach the final goal state in a problem space. Thereby, selection of the most appropriate operator is supported by preferences. A preference denotes a suggestion on the applicability of an operator in a situation. The following types of preferences are distinguished: *acceptable (+)* means the operator is suitable candidate, *reject (-)* is the opposite; *require (!)* denotes that the operator must be selected for reaching the goal, *prohibit (~)* is the opposite; *best(>)* means the operator is the best choice, *worst(<)* is the opposite; *better(> o1 o2)* means o1 is more applicable than o2, *worse(< o1 o2))* is the opposite; *indifferent(= o1 o2)* denotes that none of the operators is better; also, *numeric-ordering(= number)* can be defined fro expressing preference relations on operators. Preferences are kept in the preference memory which is linked via the identifier of a WME that stands for an operator; preferences are removed when their corresponding production rules do not match the working memory any longer. On this basis, a straight-forward *operator preference resolution*

process provides the mechanism for operator selection in the decision phase of problem solving decision cycle explained above (see [Laird and Congdon, 2004], App. A.).

3.1.3 Impasse Handling and Learning

An important feature of the basic Soar mechanisms is that its building blocks are strongly decoupled which might lead to inconsistencies. The productions in the LTM are independent of each other and no consistency check is applied; hence, the creation of preferences is "decoupled", so that so-called *impasses* can arise during the decision phase for determining the next operator to be applied. An impasse means that no clear decision can be derived from the preferences defined for applicable operators in the current state; this is handled by automatically creating and solving so-called *sub-goals* as we explain in more detail.

Four types of impasses are distinguished that can arise: the *tie-impasse* denotes that more than one operator is proposed to be applicable (i.e. several operators have *acceptable* or *require* preference, and no further preferences allow to make a selection between them); the *conflict-impasse* denotes an irresolvable contraction of preference definitions (i.e. A is better than B; B is better than A); the *constraint-failure-impasse* denotes a contraction of require and prohibit preferences; and *no-change-impasses* denote that no operator has been selected or that the selected operator is not capable of changing the state. Impasses arise due to incomplete or inconsistent preference definitions, so because of a lack of knowledge that hampers an unambiguous operator selection decision. Hence, so-called sub-goals are created in order to attain the missing knowledge for resolving the impasse by determining the missing knowledge.

Figure 2 (next page) shows an illustration of impasse handling by creation of sub-goals. In some state *S1* of the problem resolution procedure, the decision phase has detected a tie-impasse as there are acceptable preferences for two operators *O1* and *O2*. S1 is the top-level state (denoted by `superstate = nil`) that occurs during the resolution of the actual goal that is to be achieved. Hence, a sub-goal *S2* that aims at resolving the tie-impasse between *O1* and *O2* in *S1* is created automatically. Sub-goals are defined as states that carry structural information on the impasse occurrence. In the example, the sub-goal *S2* would be defined as: (`S2 ^type state ^superstate S1 ^impasse tie ^choices multiple ^attribute operator ^item O1 O2 ^quiescence t)`. This means: *S2* is a state, its super-state is *S1* wherein a tie-impasse has arisen between multiple operators *O1*and *O2* after reaching quiescence in the decision cycle for *S1*. Although not explicitly specified, this is interpreted by the system as a goal for solving the tie-impasse by determining sufficient knowledge on preferences that allow to make an unambiguous selection decision between *O1*and *O2* in *S1*. It is assumed that a production exists in the LTM that states (omitting the formal representation): "*If there is a goal for resolving a tie-impasse in problem space PS1, then use the Problem Space PS2 with an initial state that contains the tied operators*". Because of this, the standard problem solving procedure as described above is initiated for resolving *S2*.

Impasses occurring during the resolution of *S2* create a stack of further sub-goals that are processed in the same way (e.g. *S3* in the figure). The working memory for the super-goal and its sub-goal stack is the same, so that the problem solving results are available to all levels in the sub-goal stack. In each new cycle, the sub-goal stack is processed in a top-down manner. As soon as there is sufficient preference knowledge to resolve impasses on a level x in the sub-goal stack, the processing on all levels > x is terminated and the respective WMEs are removed from the working memory (the aim of resolving an impasse at level x has been achieved, so all sub-goals of level x have become obsolete).

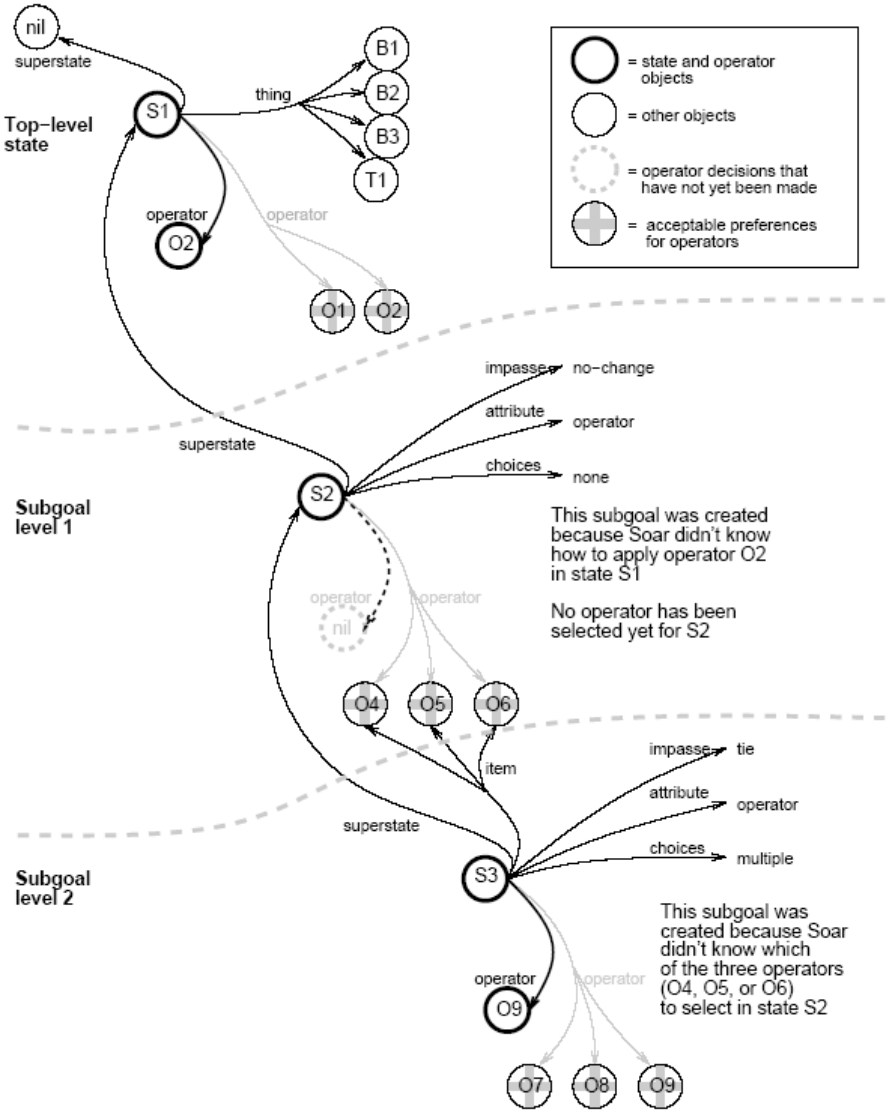


Figure 2: Illustration of a sub-goal stack[7]

[7] taken from [Laird and Congdon, 2004].

This sub-goaling technique allows automated goal-decomposition in order to attain knowledge for problem solving that is missing during run time. In order to improve the problem solving capabilities of a Soar system during its life time, the impasse-resolution by sub-goaling is extended by an automated learning technique referred to as *chunking* [Rosenbloom and Newell, 1982]. As a form of explanation-based learning, the chunking mechanism produces a new LTM production that contains the results of a solved sub-goal. The condition of this new production contains the super-state wherein the impasse has occurred that was solved by the sub-goal, and the actions are the preference creations that result from the sub-goal resolution. In the above example, if *S2* has been successfully resolved by determining a new preference *PrefS2* that allows resolving the tie-impasse between *O1*and *O2* in *S1*, then a new production is created: IF (S1) THEN (PrefS2). This is called a chunk that is stored in the LTM. The next time when the decision cycle comes to the state *S1*, this chuck can be used instead of creating a sub-goal for impasse resolution. Thereby, chunking allows simplifying the problem solving procedure by automated learning.

3.2 Conclusions

Soar provides a system for automated, goal-oriented problem solving based on productions and an elementary problem-solving procedure, including automated goal-decomposition and learning mechanisms. The inventors claim the Soar architecture to be an adequate cognitive architecture, i.e. representing and simulating intelligent human problem solving behavior as it realizes the following aspects [Newell, 1990]:

- Soar realizes a hierarchical architecture that is claimed to represent the structure of human cognition and problem solving, wherefore Newell distinguishes four levels (called *bands*): the lowest is the *biological band* that enables fast information processing by physiological neurons; this is represented in Soar by the LTM as productions that allow efficient, decoupled information processing. Above, the *cognitive band* is concerned with processing of symbols and low-level logical relation; this is represented in Soar by the working memory and the decision cycle. At the highest level, the *rational band* is concerning with problem solving strategies and tasks-driven behavior, which corresponds to the Soar mechanisms for impasse-handling and learning. Orthogonal to the previous levels, the *social band* is concerned with cooperation and collaboration by interaction with other individuals; this is represented by the perception and motor mechanisms of Soar.
- The concept of problem spaces specific segments of the world that are relevant for solving a problem is claimed to represent human behavior in problem solving. The underlying model says that human beings first select the domain that appears to be relevant for solving a problem, and subsequently consider other segments of the world if the goal can not be reached directly.
- Also, the goal-directed behavior and reflexive learning as realized in Soar is claimed to represent cognitive theories. Humans are considered to act in a goal-driven manner, having an objective to be reached wherefore the path from the current status is not known, and then subsequently perform the most adequate ac-

tions in order to achieve the objective; this is represented in the design of the Soar decision procedure. The Soar chunking mechanism is asserted to simulate human learning on basis of experiences in earlier situations.

A detailed discussion or verification of these aspects exceeds the aim and the scope of this paper. Nevertheless, inspecting Soar with respect to the six requirements on goal-driven architectures defined in Section 2.2, we reveal that all of them are fulfilled to a more or less satisfactory extent: the Soar elements are equivalent to the elements required for goal-driven architectures; Soar goals allow specifying objectives as desired final states with constraints on the known problem space; an optimal goal resolution plan is determined in a highly automated manner; formal methods are applied, and the resolution of arbitrary complex goals is (theoretically) support. Furthermore, we gain the following insights on the realization of goal-driven technology from the Soar architecture:

- goal-directed behavior can be represented in state-based models, wherein goals are defined as final states and intelligent mechanisms dynamically determine the resolution path by detecting and applying the most appropriate action in each situation
- production systems can be used as an efficient, low level technology for information processing in state-based systems
- functional operator descriptions (i.e. service-side functional descriptions) can consist of conditions that need to hold before the operator can be applied, and changes on the world that result from operator application
- the main mechanisms for are *determination of operator applicability* that can be realized by matching the current state against the operator description, and *operator selection* that can be realized on basis of preferences
- efficient control and management techniques seem to be required in order to ensure that the correct elements needed for goal resolution are available; only the minimal number of elements needs to be considered for expensive operations
- the Soar mechanisms for automated sub-goaling and learning appear to be appealing techniques for advanced, highly automated goal resolution.

The Soar knowledge representation can be considered as a shortcoming: all knowledge needs to be defined as productions in the LTM; although productions can fulfill different functions, their definition is implicit and thus hard to use; also, knowledge representation is not semantically supported (although formal languages are applied, and WME definitions realize a triple-structure). However, this does not hamper Soar to be an interesting goal-driven architecture.[8]

[8] An unrelated remark: the Soar technology seems to be very much related to the current efforts in WSMO service interfaces. Production systems with if-then rules that are fired in parallel as the basis might be a better / other / additional basis for the WSMO service interface model; also, the Soar decision cycle realization might be an interesting approach wrt WSMO service interface technology.

4 Agent Technology

The research field of intelligent agents is concerned with systems wherein agents reside as autonomous computational elements and satisfies their particular objectives in an interacting manner. Commencing in the early 1970ies, the aim has been to develop a novel paradigm for system design along with respective technologies following contemporary socio-psychological insights [Shoham, 1993], [Jennings and Wooldridge, 2001]. Imitating human problem solving behavior, a software agent shall act autonomously in its environment and collaborate with other agents in an effective manner if this is helpful for achieving its individual objectives. Therefore, architectural models and technologies have been development as general purpose facilities for *intra-agent management* (i.e. the internal management of one agent) as well as for *inter-agent management* (i.e. interaction and collaboration between several agents) [Luck et al., 2003]. The following very briefly replicates the aims and fields of research of agent technology in order to identify the aspects of interest for goal-driven architectures, referring to various resources for detailed information.[9]

A software agent is supposed to be a sovereign software unit that performs some kind of task in a (semi-)automated manner. Regarding the operation mode, the following generic properties for agents are defined that reflect theoretic models from socio-psychology on human behavior [Wooldridge and Jennings, 1995]: an agent acts self-directed and controls its own actions (*autonomous*), it interacts with humans or other agents for collaborative problem solving by means of communication (*social ability*), it observes its environment and reacts to changes therein (*reactivity*), and exhibits a rational, goal-driven behavior in order to achieve its tasks (*proactiveness*). Agent technology can be used for various application scenarios wherein agents fulfill different functional capacities. Topologies like in [Nwana,, 1996] distinguishes five functional agent types: collaborative, interface, mobile, information, reactive. However, the aim of agent technology is to provide general purpose infrastructures and mechanisms for handling the behavior and interaction of software agents independent of a concrete functionality or application scenario. Agent technology is commonly differentiated into two main aspects: so-called *agent architectures* that are concerned with the internal technical realization of a single software agent in order to realize the agent properties mentioned above, and so-called *multi-agent systems* as general purpose infrastructures that provide execution environments for agents along with management and multi-agent coordination facilities [Wooldridge, 2002].

---

[9] Collective overviews on agent technology are provided in [Wooldridge and Jennings, 1995], [Nwana,, 1996], [Wooldridge, 2002], [Russell and Norvig, 2003], [Luck et al., 2003] as well as on several Internet portals like UMBC AgentWeb (http://agents.umbc.edu), Agent Portal (http://aose.ift.ulaval.ca/), agents & multi agent technology, AgentLink Portal (www.agentlink.org), and MultiAgent Sytems (www.multiagent.com); agent technology standardization efforts are undertaken in FIPA (Foundation of Intelligent Agents, homepage: www.fipa.org) and MASIF (Mobile Agent System Interoperability Facility, homepage: www.fokus.gmd.de/research/cc/ecco/masif/).

The notion of intelligence in agent technology refers to usage of respective AI techniques in order to enhance the problem solving and collaborative capacities of software agents. Within single agent architectures, the essential aim is to realize the concept of rationality as a core principle for economic behavior by utilizing appropriate 'intelligent' techniques. Rationale agency means that an agent is able to choose the best possible action that is applicable in the current situation in order to achieve its individual objective [Wooldridge and Rao, 1999]. Therefore, intelligent agent architectures of incremental complexity have been developed that utilize AI techniques for simulating human behavior in software agents. The simplest form a *stimulus-reflex agents* that determine their behavior on basis of received information on the environment; so-called *model-based reflex agents* in addition have knowledge about how applicable actions will change the world. More complex models are *goal-based agents* whose behavior is determined by goals as the desired final state to be reached, knowledge about the effects of applicable actions, and information on the external environment that are perceived continuously. As an extension of goal-based agents, so-called *leaning agents* gain new knowledge about the applicability of actions during acting in an environment [Russell and Norvig, 2003].

Obviously, the aspects of interest with respect to examining approaches for goal-driven architectures as the aim of this paper are the concepts and techniques employed for goal-based agents. Out of several approaches have been developed therefore, the most prominent and mature works are referred to as belief-desire-intention (BDI) architectures that we hence will examine in detail. These consist of formal models as well as system architectures for intra- as well as inter-agent management for cooperations of multiple goal-based agents for collaborative problem solving that introduces an aspect of goal-driven architectures we have not addressed so far. Following the methodology of the previous examinations, we concentrate on the foundational principles of BDI agent technology while referring to respective resources for further information.

4.1 The Belief-Desire-Intention (BDI) Model and Formalization

The model of beliefs, desires, and intentions is a philosophical theory on the motivation and behavior of rationale action by humans presented in [Bratman, 1987]. Roughly speaking, beliefs denote information on the world that an agent (regardless of being a human or a machine) considers to be true, desires are the eventual objectives that the agent wants to achieve, and intentions are actions that the agent has committed to achieve as sub-steps towards achieving a final desire.

These three notions denote mental attitudes whose interrelations is considered to determine rationale action of agents. In contrast to other theoretical models, Bratman defines intentions to be a first-class citizen that determine rationale agent behavior on the level as beliefs and desires do. This model has been formalized in so-called BDI logics, and has been implemented in several systems so that BDI architectures encompass a solid philosophical foundation, software architectures for intelligent agent systems, and a sound logical formalization. Before addressing the latter issues, we first clarify the core notions in an example.

4.1.1 The notions of Belief, Desire, and Intention

Let's consider some agent named Michael that wants to write a book.[10] This is his ultimate aim and he is just about to get started with the book writing process. Here, 'write a book' denotes a *desire* of agent Michael that is considered to be achieved when the book has been published. Michael has some knowledge about the world gained from previous activities, for example that writing a book is time consuming and usually is not compatible with enjoying an exhaustive social life. This knowledge is individually considered to be true by an agent, called *beliefs*. This indicates that it the facts known by the agent must not be true universally but they hold for the individual understanding of the world from the single agent's perspective – for example, for somebody it might be true that writing a book in time and enjoy an exhaustive social life are not contradicting. Now, at midday time Michael receives an invitation for joining some friends to watch a movie at 8 p.m. that day. Michael knows (i.e. his beliefs are) that watching a movie with his friends is a joyful, time consuming activity that does not contribute to progress in book writing. Hence, Michael creates a plan for that day that states to stay at home that night and work on the book instead of going to the cinema. So, at midday time Michael has the *intention* of going home after the office hours and continues working on the book.

While the notions of belief (knowledge on the world as individually observed by an agent) and desire (final objectives to be achieved) are intuitively clear, we need to closer investigate the notion of intentions and their role within determination of rationale agent behavior. [Bratman et al., 1988] define an intention as *a partial plan of future action that an agent is committed to execute to fulfill its desires*. This means that the agent creates a plan of actions for be performed for achieving its objectives.[11] An intention denotes a plan fragment that an agent considers to be constructive for achieving its overall desire and hence commits to. When the intention is achieved (i.e. the partial plan has been executed), the agent is a new state. There might be changes in the agent's beliefs or desires resulting from continuative interaction with its external environment. On basis of all knowledge available to the agent in some state, it creates new intentions, commits to these and executes them; this process is repeated until a desire has been achieved and then goes on for other desires. Referring to the above example, three actions might be available to agent Michael: take the bus to the cinema (a_1), take the bus home (a_2), go for a coffee after the office hours (a_3). As a_1 and a_3 are less suitable than a_2 for executing the intention of going home and continue work on the book, Michael will choose a_2 as an act of rationale action.

---

[10] adopted and extended from [Wooldridge, 2000].

[11] A plan denotes a possibly multi-step process of executing actions that results in a state of the world wherein the agent's desire is achieved; a partial plan refers to a fragment of a plan; plans are executed by properly performing appropriate actions. An action in this terminology denotes manual activities as well as computational resources that are applicable for problem solving. Given numerous actions available for problem solving, a (partial) plans allow reducing the number of suitable actions as well as determining an appropriate action execution sequence (see Section 5 for a more detailed examination of plans, their properties, and AI planning techniques).

The main merit of the belief-desire-intention model for rationale action is that an agent does not determine a complete resolution plan for a goal starting from the initial state when a desire is formulated and executes this, but follows a step-wise procedure where in each step all knowledge available to the agent (gained by continuative interchange with its external environment) is taken into consideration for finding the most appropriate next action for achieving its goals. For instance in our example, imagine that agent Michael has worked on the book in the evening and has made good progress. In this new situation, he might determine that joining his friends for a drink after the cinema will be advantageous for his final desire as he needs to relax and free his mind, so he defines this as a new intention – which would not be included in a complete goal resolution plan defined earlier that day.

BDI technologies, i.e. logical formalisms and respective system implementations are technical realization of practical reasoning. In contrast to theoretical reasoning on logical formulas, this is concerned with determining intentions for agents with respect to their individual beliefs and desires in order to control and manage the behavior of rationale software agents [Wooldridge, 2000]. While the definition of desires (referred to as *deliberation*) is commonly allocated in the interaction of an agent with its owner or with other agents, the construction of intentions realizes means-ends reasoning as we have introduced introductory (how to achieve a goal by finding the best possible resolution plan in a given problem domain). Before investigating BDI technologies below, it is to remark that the underlying philosophical model is hard to validate with respect to correctness and sufficiency for explaining rationale behavior. It complies with constructivist theories of individuals as sovereign and autonomously acting entities in society as well as with modern socio-psychological models for human behavior in groups. However, while this discussion is out of the scope of this paper, the BDI model provides a thorough conceptual foundation for goal-driven technology of intelligent, rationale software agents which is our main concern in this study.

4.1.2 Formalization in BDI Logics

BDI logics are specific logical formalisms developed for specifying BDI structures of agents and serve as the basis for practical reasoning about them. The most significant contributions with respect to formalization and dealing with beliefs, desires and intentions are Cohen and Levesque's intention logic [Cohen and Levesque, 1990] and Rao and Georgeff's BDI logics [Rao and Georgeff, 1991] that we hence examine here in detail while referring to more extensive overviews like [Woodridge, 2000] for discussion of other approaches.

As most BDI logics, both approaches follow a common design of being modal logics with possible world semantics. Modal logics allow combination of different logic types into one common logical framework [Blackburn et al., 2001]. Therein, so-called *modalities* are used for specifying aspects that can not be expressed in first-order

logic; semantics of modalities are defined in Kripke structures.[12] For example, the expression `(Bel michael ◊hasPhD(michael))` says that Michael believes that he eventually will hold a PhD degree. Therein, `Bel` is a modality denoting a belief of an agent, and ◊ is a modality denoting a logical formula will be true sooner or later (common symbol for 'eventually'). Modal logics are very helpful to support reasoning on assimilated object structures. For instance, reasoning in an integrated manner on objects that contain action and knowledge requires combining epistemic and dynamic logics, see [Moore, 1977] as an early work. Possible world semantics denote that BDI logics are concerned with future actions; these are understood as sets of all states an agent can achieve by performing actions currently known by it.

Following [Wooldridge, 2000], a BDI logic needs to consist of four components in order to formally describe and reason about possible worlds. (1) a first-order logic component for expressing epistemic aspects of objects, (2) modalities for beliefs, desires, and intentions, (3) a temporal component for denoting dynamic aspects, and (4) a component for describing actions performed by agents and their effects. While we refer to the referenced papers for the formal definition of these components in the BDI logics to be investigated, the following concentrates on how beliefs, desires, and intentions are used therein for determining rational agent behavior.

Intention Logic of Cohen and Levesque

Chronologically the first approach towards a formalization of the BDI model, the intention logic of Cohen and Levesque [Cohen and Levesque, 1990] has been broadly recognized and serves as the basis for BDI-based models for multi-agent cooperation and dialogue management.

Initially, the approach was intended so serve as a partial theory of rationale agency. Therein, intentions are considered as the central mental attitude that determines goal resolution behavior that have the following properties. Adopted from Bratman's philosophical model: (1) intentions pose problems for agents, who need to determine ways of achieving them; (2) intentions provide a "filter" for adopting other intentions, which must not conflict; and (3) agents track the success of their intentions, and are inclined to try again if their attempts fail. Cohen and Levesque denote four additional properties of intentions: (4) agents believe their intentions are possible; (5) agents do not believe they will not bring about their intentions; (6) under certain circumstances, agents believe they will bring about their intentions; and (7) agents need not intend all the expected side effects of their intentions. Intentions are

[12] A Kripke structure (named after its inventor Saul Kripke) is a non-deterministic finite state machine whose nodes represent the reachable states of the system and whose edges represent state transitions. It is formally defined as a 4-tuple $\mathbf{M} = (S,I,R,L)$ consisting of a countable set of states (S), a set of initial states ($I \subseteq S$), a transition relation ($R \subseteq S \times S$) with $\forall s \in S\ (\exists\ s' \in S\ ((s,s') \in R))$, and a labeling (or *interpretation*) function (L: $S \rightarrow 2^{AP}$). The condition associated with the transition relation R states that every state must have a successor in R, which implies that it is always possible to construct an infinite path through the Kripke structure. This is commonly used to define the formal semantics of modal logics as well as other non-classical logics [Blackburn et al., 2001].

considered to be determined by the rational balance of an agent's mental attitudes, i.e. the interrelations of the beliefs and desires as well as existing intentions that an agents has a certain point in time. The formalization is based on atomic modalities shown in Table 1and on several propositions as explained below.

Table 1: Atomic Modalities in Cohen and Levesque's Intention Logic[13]

Operator	Meaning
(Bel i φ)	agent i believes φ
(Goal i φ)	agent i has goal of φ
(Happens α)	action α will happen next
(Done α)	action α has just happened

The first-order logic component is denoted by commonly used symbols like δ, ψ, etc. that can be arbitrarily complex epistemic formulas. BEL, GOAL (corresponds to 'desires' in Bratman's model), and INTEND are the modalities for the BDI component. The temporal modalities HAPPENS and DONE are augmented the standard future time modalities □ ("always") and ◊ ("eventually"), and by some action component operators for expressing sequences of actions: sequence of action (α;β), non-deterministic choice (α|β), concurrent occurrence (α||β); α? is a test operator: p?;α "when p is true, action α occurs next"; α;p? "action α occurs, after which p holds". On basis of this, future directed expressions can be defined as LATER p = ¬p^◊p.

With respect to the above mentioned properties of intentions and the desired balance of the mental attitudes of agents, the propositions listed in Table 2 are defined. These denote general relations that need to hold between all beliefs *B* and all Goals *G* of an agent *x* at a given point in time.

Table 2: Propositions on Mental Attitudes in Intention Logic

Definition	Meaning
(KNOW x p) = p ^ (BEL x p) (BEL x p) => ¬(BEL x ¬p) (BEL x p) ^ (BEL x (p -> q)) => (BEL x q)	Knowledge is 'true beliefs' beliefs are consistent beliefs are consistent under implication
(GOAL x p) => ¬(GOAL x ¬p)	Goals are consistent (an agent can have inconsistent desires; goals are the subset of an agent's desires that are consistent)
(BEL x p) => ¬(GOAL x ¬p) (GOAL x p) ^ (BEL x (p -> q)) => (GOAL x q)	Goals and Beliefs are consistent Goals and Beliefs are consistent under implication
◊(GOAL x (LATER p)	all goals are eventually dropped
(BEL x (HAPPENS e)) => (HAPPENS e))	if an agent believes that the event (an atomic action) happens next, this is a goal

[13] taken from [Hoek and Wooldridge, 2003].

On this basis, we get to the definition of the first major construct in Cohen and Levesque's logic that determines rational behavior of agents, so-called *persistent goals*. Abbreviated as P-GOAL, a persistent goal denotes an agents desire that will be kept until it is achieved or considered to be unachievable.

(P-GOAL x p) ≅	an agent x has a persistent goal of p if:
(GOAL x (LATER p)) ^	it has a goal that p eventually becomes true and
(BEL x ¬p) ^	believes that p is not currently true;
(BEFORE	one of the following must hold before the goal is dropped:
((BEL x p) V (BEL x □¬p))	(a) the agent believes the goal has been satisfied
¬(GOAL x (LATER p))	(b) the agent believes the goal will never be satisfied

So, an agent will continue to search of best possible goal resolution plans for P-GOALs as long as they are not dropped. Following Bratman's model outlined above, intentions are considered as the mental attitude that determines the actions an agent commits to execute in order to achieve a P-GOALs. Hence, the definition of an intention INTEND is that an agent *x* intends to perform action α if it has a persistent goal to have brought about a state where it had just believed it was about to perform α, and then did α:

$$(\text{INTEND } x\ \alpha) \cong (\text{P-GOAL } x\ [\text{DONE } x\ (\text{BEL } x\ (\text{HAPPENS } \alpha))?;\ \alpha])$$

For clarification, consider the goal of chopping down a tree and an action chop(T) that needs to be performed several times before a tree will finally be brought down. For some agent *x* we model (BEL x T) ^ (BEL x ¬down(T)) ^ (GOAL x down(T)) whereby the goal becomes a P-GOAL following the above definition. Furthermore, agent *x* has the above knowledge about the action *chop(T)* that can be defined as follows: (BEL x (chop(T)?; chop(T) => ◊ down(T))). So it knows that in the state where the tree has just been chopped down it has just executed chop(T). This is the condition for attaining the intention (INTEND x chop(T)) in some state wherein the tree is not yet chopped down and hence the P-GOAL is not yet dropped. Hence, it will continue executing the action chop(T) until the tree is finally brought down.

Although the example does not showcase how this formal model allows to model and reason about mental attitudes for determining rational agent behavior, the disposed reader can imagine how the approach works in more complex settings. However, the definition of intentions within the approach of Cohen and Levesque is criticized to not be compliant with the theoretical model of Bratman, or at least that the formal definition is not sophisticated enougbh for representing the intended properties and relationships. Following [Hoek and Wooldridge, 2003], the main deficiency is that intentions are reducible to beliefs and desires, and hence only denote temporal sequences of these. In contrast, Bratman's model defines intentions to be first-class mental attitudes that influence rational behavior in the same way as beliefs and desires do. The approach of Rao and Georgeff aims at overcoming this as described below.

BDI Framework of Rao and Georgeff

The BDI framework of Rao and Georgeff has been presented in a series of papers, starting with the formal model in [Rao and Georgeff, 1991] and resulting in a comprehensive BDI system definition in [Rao and Georgeff, 1998]. The principal structure of the formalization is similar to the one of Cohen and Levesque – i.e. defining the modalities BEL, GOAL, and INTEND along with common temporal as well as action component modalities. The main differences are that intentions are treated as first-class construct, and that beliefs, desires (resp. goals as the subset of an agent's desires that are consistent), and intentions are understood as possible worlds. This approach is considered to better Bratman's model [Hoek and Wooldridge, 2003].

All mental attitudes of agents are represented in so-called *time trees*, a temporal structure that represent the current situation of an agent at a point in time to have a single past as the known behavioral history of the agent, and a branching future. Called *accessible worlds*, this denotes all possible situations the agent can access with respect to its current knowledge, whereby the transition between accessible states are denoted by events (i.e. atomic or non-atomic actions). So, in each state an agent has, possibly several, belief-, goal-, and intention-accessible worlds. The interesting feature that provides the core for practical reasoning are ontological relationships that hold between mental attitudes of an agent. So-called *belief-goal-compatibility* states that if an agent adopts a goal on some δ it also believes δ (desiring something that is not believed to become true anyway is inadequate); similar, *goal-intention-compatibility* states that if an adopts an intention on α it also believes α. Referring to the referenced papers for formal definitions, a sub-world relationship in possible world semantics denotes that the sub-world only consists of a subset of the situations and paths in the super-world and has no additional situations and paths. In consequence, it holds that for each situation of an agent the goal-accessible worlds are a sub-world of the belief-accessible worlds, and the intention-accessible worlds a subset of goal-accessible worlds: $\forall w' \in B_t^w \exists w'' \in G_t^w . w'' \subseteq w'; \forall w'' \in G_t^w \exists w''' \in I_t^w . w''' \subseteq w''$.

The example shown in Figure 3 refers to an agent that needs to get a tooth filled *f*, i.e. (GOAL x f). The agent believes that it is inevitable (always true) that pain *p* accompanies having a tooth filled (f): (BEL x □(f->p)). The belief-accessible world *b1* has three events: *d1* and *d2* result in *p* and *f*, *b* results in ¬*p* and ¬*f*. As event *b* will not lead to a state where the goal is not satisfied, the goal-accessible world *g1* has two states reachable by *d1*, respectively *d2*; *g1* is a sub-world of *b1* according to the above definition. The agent chooses event *d1* to commit to for execution (INTEND x d1) – the reason of choice between *d1* and *d2* is not depicted here – and hence has a intention-accessible world *i1* that is a sub-world of *g1*. The goal- and intention worlds *g2* and *i1* are not sub-worlds of *b1* and hence not accessible to the agent with respect to the belief-goal compatibility and goal-intention compatibility that always need to hold.

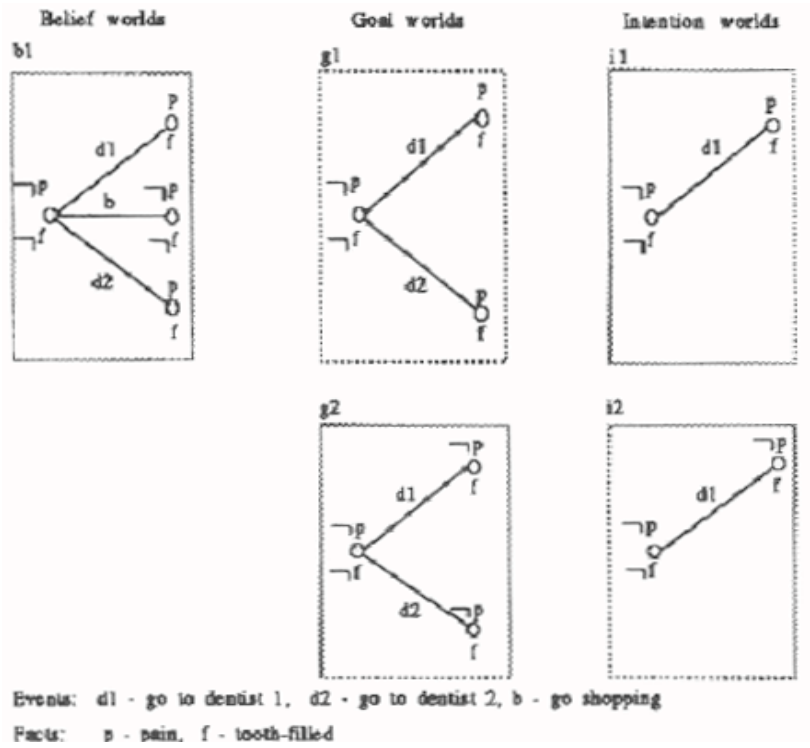


Figure 3: Example of B-G-I accessible worlds[14]

There are two main differences in comparison to Cohen and Levesque's model. First, the agent in this example does not need to adopt (GOAL x p) although it believes that filling teeth will always lead to pain. After executing event *d1* as the intention picked to commit for execution, it will be in a state where *f* holds, so the goal is solved, and it excepts *p* as a side effect without having to desire *p*. Secondly, the formal definition of mental attitudes as possible worlds along with the relationships between these allows specifying and reasoning on beliefs, desires, and intentions as constructs of equal importance in a much more realistic fashion. In fact, this allows formalizing the so-called *desired properties of rational agents* with respect to their mental attitudes in conformance to the philosophical model. Table 3 shows these properties which denote the final results of Rao and Georgeff on BDI formalisms.

[14] taken from [Rao and Georgeff, 1991].

Table 3: Desired Properties of Rational Agents in BDI logic[15]

Name	Semantic Condition	Corresponding Formula Schema
BDI-S1	$\mathcal{B} \subseteq_{sup} \mathcal{D} \subseteq_{sup} \mathcal{I}$	$(\mathsf{Intend}\ i\ \mathsf{E}(\varphi)) \Rightarrow (\mathsf{Des}\ i\ \mathsf{E}(\varphi)) \Rightarrow (\mathsf{Bel}\ i\ \mathsf{E}(\varphi))$
BDI-S2	$\mathcal{B} \subseteq_{sub} \mathcal{D} \subseteq_{sub} \mathcal{I}$	$(\mathsf{Intend}\ i\ \mathsf{A}(\varphi)) \Rightarrow (\mathsf{Des}\ i\ \mathsf{A}(\varphi)) \Rightarrow (\mathsf{Bel}\ i\ \mathsf{A}(\varphi))$
BDI-S3	$\mathcal{B} \subseteq \mathcal{D} \subseteq \mathcal{I}$	$(\mathsf{Intend}\ i\ \varphi) \Rightarrow (\mathsf{Des}\ i\ \varphi) \Rightarrow (\mathsf{Bel}\ i\ \varphi)$
BDI-R1	$\mathcal{I} \subseteq_{sup} \mathcal{D} \subseteq_{sup} \mathcal{B}$	$(\mathsf{Bel}\ i\ \mathsf{E}(\varphi)) \Rightarrow (\mathsf{Des}\ i\ \mathsf{E}(\varphi)) \Rightarrow (\mathsf{Intend}\ i\ \mathsf{E}(\varphi))$
BDI-R2	$\mathcal{I} \subseteq_{sub} \mathcal{D} \subseteq_{sub} \mathcal{B}$	$(\mathsf{Bel}\ i\ \mathsf{A}(\varphi)) \Rightarrow (\mathsf{Des}\ i\ \mathsf{A}(\varphi)) \Rightarrow (\mathsf{Intend}\ i\ \mathsf{A}(\varphi))$
BDI-R3	$\mathcal{I} \subseteq \mathcal{D} \subseteq \mathcal{B}$	$(\mathsf{Bel}\ i\ \varphi) \Rightarrow (\mathsf{Des}\ i\ \varphi) \Rightarrow (\mathsf{Intend}\ i\ \varphi)$
BDI-W1	$\mathcal{B} \cap_{sup} \mathcal{D} \neq \emptyset$ $\mathcal{D} \cap_{sup} \mathcal{I} \neq \emptyset$ $\mathcal{B} \cap_{sup} \mathcal{I} \neq \emptyset$	$(\mathsf{Bel}\ i\ \mathsf{A}(\varphi)) \Rightarrow \neg(\mathsf{Des}\ i\ \neg\mathsf{A}(\varphi))$ $(\mathsf{Des}\ i\ \mathsf{A}(\varphi)) \Rightarrow \neg(\mathsf{Intend}\ i\ \neg\mathsf{A}(\varphi))$ $(\mathsf{Bel}\ i\ \mathsf{A}(\varphi)) \Rightarrow \neg(\mathsf{Intend}\ i\ \neg\mathsf{A}(\varphi))$
BDI-W2	$\mathcal{B} \cap_{sub} \mathcal{D} \neq \emptyset$ $\mathcal{D} \cap_{sub} \mathcal{I} \neq \emptyset$ $\mathcal{B} \cap_{sub} \mathcal{I} \neq \emptyset$	$(\mathsf{Bel}\ i\ \mathsf{E}(\varphi)) \Rightarrow \neg(\mathsf{Des}\ i\ \neg\mathsf{E}(\varphi))$ $(\mathsf{Des}\ i\ \mathsf{E}(\varphi)) \Rightarrow \neg(\mathsf{Intend}\ i\ \neg\mathsf{E}(\varphi))$ $(\mathsf{Bel}\ i\ \mathsf{E}(\varphi)) \Rightarrow \neg(\mathsf{Intend}\ i\ \neg\mathsf{E}(\varphi))$
BDI-W3	$\mathcal{B} \cap \mathcal{D} \neq \emptyset$ $\mathcal{D} \cap \mathcal{I} \neq \emptyset$ $\mathcal{B} \cap \mathcal{I} \neq \emptyset$	$(\mathsf{Bel}\ i\ \varphi) \Rightarrow \neg(\mathsf{Des}\ i\ \neg\varphi)$ $(\mathsf{Des}\ i\ \varphi) \Rightarrow \neg(\mathsf{Intend}\ i\ \neg\varphi)$ $(\mathsf{Bel}\ i\ \varphi) \Rightarrow \neg(\mathsf{Intend}\ i\ \neg\varphi)$

Three categories of desired properties of rational agents are distinguished, whereby each denotes tenary relationships between beliefs, desires, and intentions for so-called A-formals that denote *inevitabilities* (first-order formulas that are true in all states of all goal- and intention accessible worlds of an agent – e.g. *f* in the above example) and so-called E-formulas that denote *options* (first-order formulas that are true for at least one path in the goal- and intention accessible worlds of an agent), and then generalize this. The first group (S-properties) allows inferring desires and believes when an intention is given in a situation of an agent belief-goal compatibility and goal-intention compatibility. For example, in some state an agent has (INTEND x α) but does not yet belief α, we attain that it believes and desires α from now on. The second group (R-properties) is used for verifying the correctness of belief-, goal-, and intention accessible worlds as depicted in the above example. For all intentions it has to hold that these are desired, and for all desires it has to hold that they are believed. The third group (W-properties) is concerned with weak realism, i.e. consistency of an agent's mental attitudes in a specific situation with respect to its rational balance. If the agent believes something, it can not desire the opposite, and similar for desires to intentions and beliefs to intentions.

The final aspect that is improved in Rao and Georgeff's framework in comparison to the model of Cohen and Levesque is the differentiation of so-called *commitment strategies*. These formalize when and under which conditions an agent drops or reconsiders its intentions and hence determines the rational behavior of an agent. The following three commitment strategies are distinguished (see the 1991 paper for for-

[15] Source: [Rao and Georgeff, 1998, p. 321].

mal definitions): *blind commitment* denotes that an agent keeps its intentions until it believes it actually has achieved them; *single-minded commitment* denotes an agent keeps its intentions until they are achieved or not longer achievable, and *open-minded commitment* denotes that the agent maintains its intentions as long as they are considered to be possible. Due to handling intentions as a construct of equal importance as desires and beliefs, this refines the fanatical commitment implied in Cohen and Levesque's P-GOALs that is similar to blind commitment as the strongest, but not necessarily most appropriate strategy.

Concluding investigating formalizations of the BDI model, we denote that modal logic allows capturing the semantics of beliefs, desires, and intentions as the mental attitudes that determine the rational behavior. While Cohen and Levesque have laid the foundation by formalizing the basic properties of Bratman's BDI model, the work of Rao and Georgeff enhances this by future branching time trees and the formal specification relationships between the mental attitudes that allow sophisticated practical reasoning on basis of beliefs, desires, and intentions.

4.1.3 Collaboration of Multiple BDI-Agents

So far, we have examined BDI techniques for intra-agent management as a realization of intelligent goal-based agent technology. A main aspect of agent technology is that individual agents interact and collaborate with other agents in order to achieve their individual objectives. This means that agents do not only interchange information needed for their internal computation, but they might cooperate in a collaborative manner in order to achieve objects that require purposeful interaction of several agents. Therefore, an agent needs to detect other agents as potential partners and determine its collaborative behavior during the collaborative problem solving process.

In order to expound what we are interested in with regard to goal-driven architectures, we first need to clarify aspects and terminology before investigating respective approaches. First of all, interaction with its environment is an inherent characteristic in models of agency denoted by the *social ability* property of agents (see above). Therefore, an agent has *sensors* through which it perceives information from the environment as input for its internal computation, and *activators* through which it submits information to its environment; information interchange is performed constantly and orthogonally to internal computations of an agent [Russell and Norvig, 2003]. Coordination in agent technology is a general term concerned with techniques establishment and management of agent interactions [Luck et al., 2003]. Some agent architectures implement coordination mechanisms by defining a specific agent type that coordinates the interaction of other agents by means of procedural control structures, occasionally enhanced by semantic techniques. For example, the RETSINA system [Sycara et al. 2003] defines so-called middle agents that determine agents of other types like interface, task, or information agents as appropriate interaction partners can control the interaction between them; the Open Agent Architecture OOA [Martin et al., 1999] defines so-called *facilitators* that fulfill the same purpose. However, these systems realize central control architectures for managing interaction of

agents on basis of functional categorizations wherein the agents do not necessarily need to be goal-driven.

In contrast, we are interested in how to determine goal-based agents as partners for collaborative problem solving. Here, we understand collaboration to be concerned with interaction of individuals that want to achieve objectives wherein some entity exhibits an object or a facility that another entity needs in order to achieve its individual objective; thus, these entities need to interact in a cooperative manner. While [Stollberg et al., 2005] discusses the epistemology of collaboration in more detail, the techniques for determining goal-based agents as potential collaboration partners and successively control the interaction with respect to goal achievement are commonly referred to as MAC-theories, short for Multi-Agent Collaboration [Wilsker, 1996]. The general idea is that several agents can successfully complete collaboration if they have a common goal, agree on a sequence of actions to accomplish the common goal, each agent is able to perform collaborative actions and intends to do so, and each agent commits to the overall success of the collaboration [Grosz and Sidner, 1990]. Out of several existing approaches, we choose the Joint Intention theory that provides seminal work on this field [Levesque et al., 1990] for explication of rational agent collaboration; more recent approaches integrate several other aspects while following the same idea [Wooldridge, 2000].

Based on the BDI logic of Cohen and Levesque examined above, the joint intention theory defines the concepts of *mutuality* that denotes mental attitudes of cooperative BDI agents and *joint persistent goals* that denote the objective to be reached by collaborative problem solving. The former notion deals with either beliefs and desires on objects that are commonly shared by agents, or respectively with beliefs and desires that an individual agents has on some other agent. On basis of mutual beliefs and desires, a joint persistent goal denotes a desire that is shared between two or more agents and that can only be achieved if each agent achieves its respective part. In consequence, the agents autonomously determine and commit to intentions for solving its individual goal; if each agent has achieved its individual goal, then the joint goal is achieved as well.

The following shows the definition of a following the notation and definition of persistent goals (that comply "desires" in the intention logic) explained above. It states that two agents *x* and *y* have a joint persistent goal JP-GOAL on some predicate *p* if (1) both *x* and *y* believe that *p* is not true, (2) both have the desire to eventually achieve *p*, and (3) both *x* and *y* know that each one of them will behave rationale, i.e. not dropping the goal to achieve *p* as long as it is not believed to be achieved or never achievable. Because the beliefs and desires of each agent are dependent on the one of the other, the behavior of *x* and *y* appears to be rational collaborative although each agent individually and autonomously plans and commits to intentions for solving it respective parts of the joint persistent goal.

$$(\text{JP-GOAL } x\ y\ p) \cong (\text{MBEL } x\ y\ \neg p) \wedge (\text{MGOAL } x\ y\ (\text{LATER } p)) \wedge (\text{MKNOW } x\ y\ (\text{UNTIL } [(\text{BEL } x\ p) \vee (\text{BEL } x\ \Box\neg p)]\ (\text{MGOAL } x\ y\ p)))$$

$$(\text{MGOAL } x\ y\ p) \cong (\text{MBEL } x\ y\ (\text{GOAL } x\ p) \wedge (\text{GOAL } y\ p))$$

4.2 System Architectures and Implementation for BDI Agents

The Procedural Reasoning System (PRS) form SRI International implements the BDI model in a very sophisticated way. That is why we decided to use this model for detailed investigation. At the time when the paper was written this framework represented the most developed approach to the theory of BDI logic. [Artificial Intelligence Center, 2001].

4.2.1 Introduction

The Procedural Reasoning System (PRS) from SRI International is a framework for constructing real-time reasoning systems for performing complex tasks in dynamic environment. The framework implements the BDI Logic in very exact and detailed way. It works on the base of procedural knowledge which describes actions to fulfill a goal. For example washing clothes: If we want o clean clothes we have to take the dirty clothes put it into the washing machine, put some soap to it and then we have to wait one hour or so to reach our goal: clean clothes. PRS provides an environment in which this knowledge about action and goals are expressed and executed.

PRS can operate in highly dynamic environment as an embedded execution system. The system can reach any goals which it already knows in its world meanwhile it reacts to any new events and goals. In this way it can be easily integrated into goal driven and event driven applications.

The system contains some powerful capabilities for real time applications such as multiple copies of objects, work as an agent, or also the possibility to run action parallel. PRS also supports metalevel capabilities which can be used for complex control and scheduling behaviors which are required for individual applications.

4.2.2 Overview

The PRS System bases on five main elements, a database in which includes the actual knowledge of the system, goals which describe the states that should be reached finally, acts can be seen as a temporarily states during the resolution process, intentions are tasks which response to posted goals or facts and finally the interpreter which handles the complete goal resolution. In the following we will specify and describe the elements in a more exact way.

(1) Database:

The database contains the actual information about the world. The database supports dynamic information as well as static information about a domain. Static information describes fixed properties about the application domain such as the structure of subsystems or physical laws that must be considered by mechanical components. This information is saved in the database for life. Dynamic objects are not the same all time and so they have to actualize from time to time. For example: observation of the world may change from time to time.

Database facts can also describe the internal state of PRS e.g.: metalevel-facts. Metalevel facts describe the current goals and actions of the system. They are very important for the implementation of alternative control strategies for PRS.

(2) Goals:

Goals are normally expressed as conditions over a sequence of time that means over a sequence of world states. They are specified as a combination of a goal operator and a logical formula. In the following the accepted goal operators:

Table 4: Goal Operators

Definition	Meaning
Achieve C	achieve the condition C
Achieve-by C (A1…An)	achieve the condition C with restricted sets of acts (a1...An)
TEST C	test condition
USE-RESSOURCE R	take the resource C
WAIT-UNTIL C	wait as long as condition C is true
REQUIRE-UNTIL G C	check that goal G stays true until condition C is satisfied
CONCLUDE P	add P to database
RETRACT P	: remove P from database

As database facts can describe internal states of PRS goals can characterize internal behavior of the system. This is called metalevel-goals.

(3) Acts:

The way to reach a goal or to react to a certain situation is specified by a declarative procedure specification which is called Acts. An Acts consists of a plot in which the steps of the procedure are described. The environment specifies the preconditions for which the Act can be used. Both components environment and plot specifies declaratively in which way an action can be used to respond to a goal or event in certain situations.

The plot of an act can be viewed like a plan to reach a goal. A graph is representing the starting situation the way and also the goals and subgoals which have to be done to reach the final state or goal. To fulfill the act ever goal and every subgoal must be reached or be successful.

Each PRS application contains two kinds of acts. On the one hand the act formalized by the user and on the other hand some predefined default acts that are built into the system by itself. User-specified acts can contain both acts, predefined ones that pertain to the application domain and also metalevel acts which manipulate the beliefs, goals and intentions of PRS. Metalevel Acts can be used to encode actions that influence the operation of the system.

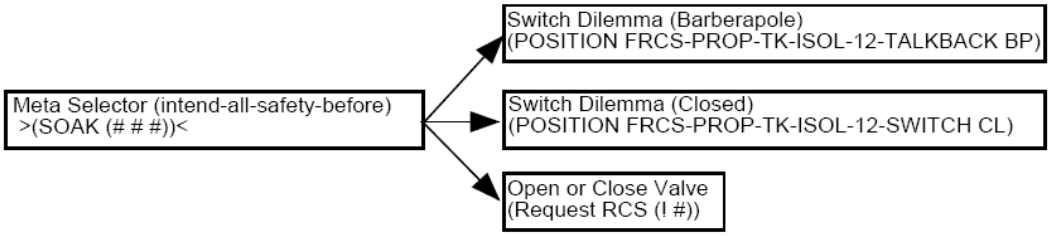


Figure 4: Example of an Intention Graph[16]

(4) Intentions

An intention corresponds to a task to be performed by the system which response to some posted goals or facts. It consists of some acts with all sub acts to satisfy the subgoals of the original act.

The intention graph orders the intentions with possible multiple least elements. The order must be either realized or dropped (disappears from the intention graph) before it can be executed. This gives the system the possibility to prioritize execution of intentions.

The example of an Intention Graph illustrates the fault diagnosis in the Reaction Control System of the NASA Space Shuttle. In handling a malfunction, the system might have in some instants four tasks to handle.

(5) The Interpreter

The PRS interpreter runs and handles the entire system. The following describes the function of the interpreter by using an example:

Certain goals are established and certain events occur that the beliefs of the database changes (1). These changes lead to various acts (2). The acts will then be chosen and put into the intention graph (3). Then PRS selects an intention (task) from the intention graph (4). Afterwards one step will be executed (5). This will return a result either in doing an primitive action in the world (6) and establishing a new subgoal or it will return a modification of the intention graph.

At this point the cycle of the Interpreter starts a new session until the actions are finally executed and the goal is reached. Primitive actions can take place in two situations: As a part in the world or as an internal state of the system. The action may operate directly on the beliefs of the system or indirectly in a growing knowledge of the system. On the on hand some goals lead to a new Act, PRS will also try to fulfill already saved actions or goals. That means acts can be expanded in a manner analogous to the execution of subroutines in the programming system.

[16] Taken from Procedural Reasoning System, User's Guide, AI Center SRI International 333 Ravenswood Avenue Menlo Park, CA 94025

If some important facts or goals do become known, PRS will realize this and can also decide to change its current intention to another one and starts searching in a completely different way. In this way the intention graph can be changed after every cycle. In this way the system can easily react on changes of the world or environment.

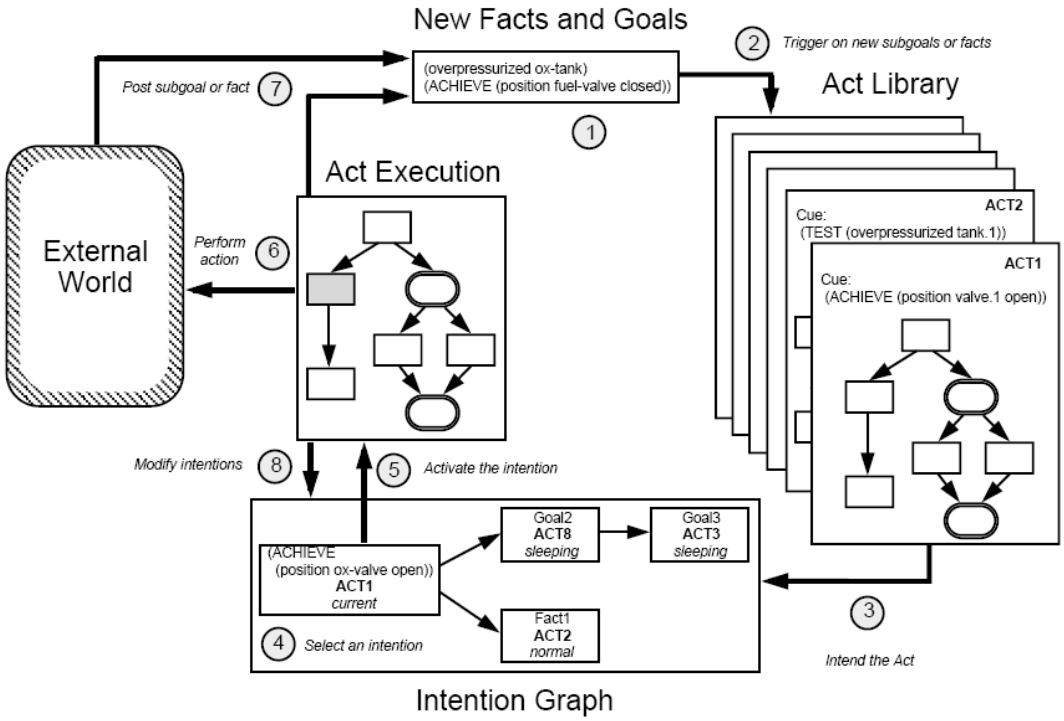


Figure 5: The Interpreter Loop[17]

4.3 Conclusions

The BDI agent is an approach to simulate the human behavior in problem resolution. Every BDI Agent acts as a so said individual unit which has some knowledge about the world, a knowledge which was influenced by the surrounding and also by the way the unit was used. Like a human the agent can interact with other agents to enlarge its knowledge. Every agent acts self-directed and controls its own actions, it communicate with humans or other agents for collaborative problem solving by communication (social ability), it observes its environment and can autonomous react on changes and exhibits a rational goal driven behavior in order to achieve its tasks.

The BDI agent system bases on the philosophical theory of beliefs, desires an intentions. In this situation beliefs denote information on the world that an agent considers to be true, desires are the eventual objectives that an agent wants to achieve and intentions are actions which must be done to achieve a sub step on the way to the final

[17] Taken from Procedural Reasoning System, User's Guide

desire. Rational agents provide a realization of the means-end-analysis which is the general problem solving strategy employed by humans. As it is described in Section 2.2 there are six requirements to AI technologies to provide Goal Driven Architecture.

- BDI Agents support the possibility to define a goal as the desired final state which should be reached. In BDI Agents a goal is defined as a condition which should be reached by finding a path through several sub-goals to a final desired state. A BDI Agent also supports the possibility to build a not completed resolution plan that means it is not necessary to start with a complete plan to find the final desired state it is also possible to change the plan after one step because of new knowledge about the world.
- As in chapter 2.2 described goal definitions can be "accompanied by additional constraints" which are depended on the knowledge level of the agent. In BDI implementation the database fulfills this requirement. The database contains the actual knowledge about the world of the agent and with every task this knowledge will be expanded.
- BDI agents support an automated goal resolution on a high level with very low information requirement on the goal. In this way the amount of information to be needed to fulfill a certain task depends on the experience of the BDI agent which is directly depended on the number of similar tasks which were already fulfilled by the agent. Like a human a BDI agent which has already booked thousands of flights for users needs less information than a BDI agent which has more experiences with making insurances for cars.
- BDI agents are using two types of formalization: The "**intention logic**" from Cohen and Levesque, 1990 and the "**BDI logics**" from Rao and Georgeff, 1991. Both have an unambiguous formal description which leads to a high automated goal resolution plan.
- The possibility of BDI agent not to build a complete resolution path to the final desired state gives the agent the possibility to use new information which where stored during the resolution path directly to find a more efficient way for goal resolution.
- BDI agents are able to split a goal in multiple sub goals which leads to a better resolution of complex problems.

BDI agents are a prominent approach in AI. Every BDI agent uses methods like sensors to get information about the actual environment. An interpreter communicates between the sensors and the agent. In order to other agent technologies BDI agents have another data structure. It consists of three elements namely *beliefs*, *desires* and *intentions*. *Beliefs* define the actual knowledge of the agent about the world. This knowledge changes permanently. *Desires* are representing the main goals of the agent, which influences the behavior of the agent in general. The *intension* is representing a goal which the agent wants to achieve. To achieve this goal the agent uses hierarchical plans, stored in a database, to come closer to the final desired state (goal). Every plan consists of several subplans called *intentions*.

5 AI Planning

AI planning in concerned with automated techniques for determining plans as combinations of several operators for solving more complex problems. A plan is sequence of actions that leads form the initial state of a problem description to the final desired state [Ghallab et al., 2004]. There are two main different types of planning: *Classical planning* and *Nonclassical planning*. The former defines the basic techniques within environments that are fully observable, deterministic, finite, static and discrete; the latter is used in only partially observable or stochastic environments [Russell and Norvig, 2003].

In the following chapter we will give an overview of what planning is about, followed by the planning problem and planning techniques. Then we will have a closer view on two widely used model and description languages for goals called STRIPS and a further developed language from CTL called EAGLE [J. Allen, T. Austin and J. Hendler, 1990].

5.1 Overview

On a regular basis, it occurs that applying a single operator might not be sufficient for resolving a goal, but executable combinations of several operators might be. For this reason it is important to have a so called *plan* for the resolution process. A *plan* is a sequence of actions to reach a goal [Russel, Novig, 2003]. For example: If a person wants to open a business he needs basically the following requirements: The knowledge about how to manage a business, some money for the beginning, an office, some employees, etc. Now he orders them that means, first the knowledge, second the money, third the office and last but not least the employees. This order is called a *plan*. In AI a plan describes the optimum order of continuing actions that are applied to reach goals, subgoals and the final state.

5.1.1 The Planning Problem

The classical planning problem consists of the following task: given the initial state of the world, several actions and their (deterministic) effects, and a sequence of actions (viz. a plan) to achieve a certain goal state. The aim and purpose of planning is to provide techniques for dynamically combining several operators that provide smaller functionalities into an executable sequence for solving problems that require more complex functionality for resolution.

Classical planning techniques apply forward- or backward-chaining as the underlying inference mechanisms for planning algorithms. Informally, the idea of forward chaining is to iteratively apply a possible operator *O* to a set of input parameters provided by the initial state of the goal formulation *G* (i.e., all inputs and preconditions required by *O* have to be available). If applying *O*does not solve the problem (i.e., the desired final goal state of *G* is not achieved), then a new query *G'* can be computed from *G*and *O*; the whole process is iterated. Backward chaining planers start from the

desired final goal state of *G* and at each step of the process we choose an operator *O* that will provide at least one of the required parameters. Applying *O* might result in new parameters being required which can be formalized as a new goal *G'*; again the process is iterated until a solution is found.

Planning algorithms have a large computational complexity because, in each iteration, all available operators need to be checked. Several techniques have been developed to address this problem. The most important ones are **heuristic functions** and **problem decomposition**. The former is used to split goal formulations in order to reduce computational complexity. For instance, an agent shall buy more than one book, for example 3 books. For finishing this action he would need if we have again 10 digit ISBN numbers 10^{30} actions (10 for each digit number and 30 for each book). In this situation an agent acts differently to a human who would consider that it would obvious to have one action for each of the remaining books. For an agent this is not obvious because a goal can only result true or false. To solve this problem it is necessary that an agent is able to split the goal into several subgoals in which each subgoal is finished independently. That means the action *Buy(object 1)^Buy(object 2)^Buy(object 3)* would be split into three different actions *Buy(object 1), Buy(object 2)* and *Buy(object 3)*. In this case the agent is able to use the right heuristic for each action. Similar, problem decomposition is concerned with de-constructing a complex goal into smaller subgoals than can be addressed independently for planning. For instance, consider the scenario of packet delivery by UPS. The company has several packets which should be transported to addresses all over the world. Normally it would make sense to find the nearest airport for every destination. In this case it can happen that it will take O(n!) time if the problem cannot be decomposed. But if it is possible to split the problem into k equal parts it will only take $O((n/k)! \times k)$ times to solve the problem. This helps the planner to work on the different subgoals independently but with the knowledge that perhaps he needs some additional work to combine the subplans. Every agent bases on the assumption that nearly every problem can be decomposed into several subplans.

5.1.2 Planning Techniques

As we already mentioned above there are different kinds of planning systems. For every plan it is necessary to have a goal. Normally this is called final state or resolution of a problem. In this context we will mention two different kinds of goal resolution processes, the basis or **classical planning** where the Hierarchical Task Network Planning (HTN) is an extension and the **nonclassical planning** environments.

The basic of the **classical planning** have already been mentioned above (5.1.1); hence we will concentrate on the extension of the classical planning HTN. The main differences between the classical planning and the extension of HTN are the way how a problem will be decomposed. In Classical Planning a problem is decomposed into a *large* number of individual actions. This method can lead to high cost for finding the resolution to a given problem. That means using such a method for large problem solution would lead to high inefficiency. In HTN every problem is spitted into several

sub problems and sub problems can be again spitted into several sub problems. HTN makes a top down hierarchy in which every branch illustrates one sub goal. This method can result to linear- instead of exponential- time planning algorithms. In HTN planning the description of a problem is viewed as a very high level description of what should be done. Plans are refined by applying **action decompositions**. Each decomposition reduces a high-level action to a number of lower actions. For example: We want to public a website and so we will split this goal into the following subgoals: finding a provider and a domain name for the website, making a design for the site, writing the content and uploading the final site. This process will be continued until only **primitive actions** remain in the plan. Such primitive actions can be executed by the agent without the help of humans.

In the real HTN planning, plans are generated only by action decomposition. HTN sees planning as a process to simplify and concrete a given problem. However this method is for some task helpful most planning agents are using some kind of hybrid, means parts of HTN combined with other technologies.

5.2 Goal and Action Description in Planning

5.2.1 STRIPS

The main task of representing a planning problem is to split it into states, actions and goals which should help to find the logical structure of a problem for the planning algorithm. For this we need a language which is on the one hand complex enough to describe a wide variety of problems and on the other hand restrictive enough to use efficient algorithms. STRIPS [R. Fikes and N. Nilsson. *STRIPS: A new approach to the application of theorem proving to problem solving*. Artifcial Intelligence, 2:189-208, 1971] is representing one of the first languages of classical planning. The main elements of the language are: Representation of states, representation of goals, and representation of actions.

Representation of states. The world will be decomposed by the planner into logical conditions and any state is represented as a conjunction of positive literals. Propositional literals are used for describing the state of an agent, for example *Poor^Unknown* represents a hapless agent. STRIPS is also using first order literals for representing a state. In this context it is important to outline that any literal in first- order logic must be **ground** and **function- free**. Conditions which cannot be seen as a state are false and will be ignored. This property is called **closed-world assumption**.

Representation of goals means that a goal is a partially specified state, represented as a conjunction of positive literals. Let us consider we have a goal A which is satisfied by a state f if this state contains all elements or more of the goal A. For example *Fast ^Expensive ^Rare* is representing the goal *Fast^Expensive*.

A representation of actions is the effect after it was executed. Normally an action cannot start until all preconditions for this action are fulfilled. For example our packet delivery: we have the *Action(Ship(packet s, from Shanghai, to Vienna)*.

To fulfill this action we need the following preconditions: *At(packet s, from Shanghai)^Ship(packet s)^Harbour(from Shanghai)^Harbour(to Amsterdam)^At(packet s, from Amsterdam)^Train(from Amsterdam)^Train(to Vienna)*. Such an illustrated schema is representing a so called **action schema** instead of one single action. An action schema consists of three parts:

- Name of the action including a parameter list
- The **Precondition** which is a conjunction of function free literals which must be fulfilled to execute the action
- The **effect** which is also a conjunction of function- free literals which describes how the state changes after the action is executed.

Some planning systems are using two types of lists to improve the readability, the **add list** for positive literals and the **delete list** for negative ones.The best way to describe the syntax for planning problems is to declare in which way an actions effect states. To do this we must say that an action is **applicable** if it satisfies the precondition. For exemplification, the following syntax specifies a precondition needed in the car industry:

At (electric, Augsburg)^ At(motor, Steyr)^ train(electric)^ train(motor)^train(Augsburg)^ train(Steyr)

Now we formalize a state which satisfies the precondition:

At(electric,from)^train(electric)^train(from)^train(to)

The next step is a substitution: {electric/E1, from/Augsburg, to/Steyr}. This results in an action which is **applicable**. In the easiest way the solution of a problem is just a sequence of actions which results in a state which satisfies the goal.

Regarding knowledge, state, goal, and operator descriptions, STRIPS requires several restrictions, among which the following are considered as most important [Ghallab et al., 2004]:

- Only positive literals in states: *Poor ^ Unknown*
- Closed world assumption: unmentioned literals are false
- Effect P ^ ¬Q means Add P an delete Q
- Only ground literals in goals: *Rich ^ Famous, Modern ^ Dynamic*
- Goal are conjunctions: *Modern ^ Dynamic*
- No support for equality
- No support for types.

When STRIPS was developed the core idea was to develop a language in which planning algorithm are more simple and more efficient on the one hand and on the other hand it should not be to complicated to describe real world problems. STRIPS

understands a goal as a final desired state which must be reached to solve the problem. For a lot of real world problems this approach is not satisfactorily.With this idea one of the key elements was the fact that literals must be *function free*. But in the recent year it became clear that STRIPS is inefficient for some real world domains. In this way a lot of other languages or extensions were developed. Among these, we inspect EAGLE below which gives a solution to the weak points of STRIPS.

5.2.2 EAGLE

Several situations require applications in which planning needs extended goals. Actions will lead to different outcomes which can't be predicted at planning times, and where goals are not only states to be reached but also conditions in the whole plan execution paths. This is called extended goals. In such systems algorithms are needed which include the possibility of different outcomes and the extension of the search space.

CTL (Computation Tree Logic) is a well known language for expressing goals [Emerson, 1990] that – in contrast to STRIPS – allows expressing temporal behaviors of goals. This is described by an *universal quantifier* and an *existential quantifier*. However CTL is not able to express different kind of goals which are relevant for non-deterministic domains. In the real world there are situation in which a goal cannot be satisfied. If such a situation would be formalized in CTL the whole process would be finished with a failure. To avoid this problem it would be necessary that the agent is able to recover from failure. This can be done by using a new formalization which *weak* the goal. That means the agent tries to reach a new goal which nearly satisfies the original goal. For example: We have a robot which should deliver things from one room A to room B. Room A and room B are separated by an automatic door which only opens after a certain time. Now our robot arrives the door at a time when the door is closed. Now the robot cannot deliver the things to the certain room. If we formalize this situation CTL the robot would end the process with a failure. If we can weaken the goal we are extend the plan that if the door is closed the robot should wait a certain time until he start a new approach to deliver the thing into room B. This extended language is called Eagle.

Eagle is based on CTL with the ability for extended goals in non deterministic domains. It provides basic goals for expressing conditions which the system should reach or maintain, and conditions that the system should *try* to reach or maintain. It is possible to define basic goals which should be achieved in reaction to a failure and goals that should be repeated until a failure occurs. Table 5 shows the new elements introduced with Eagle.

Table 5: Elements of Eagle

Element	Description
DoReach	Specifies a property that should be reached
DoMaint	Property that should be maintained true
TryReach	Specifies a property that the agent should try to reach but with the ability to weak this goal
TryMaint	Property that the agent tries to be maintained true
Fail	This property is used to recover from failure. It can be used very flexible with a lot of other operators, at planning time as well as at execution time
Repeat	Repeat defines the possibility that in case of failure the agent should repeat the action after a defined cycle

These additional description elements allow specifying trial- and compensation strategies for goals. If a given goal can't be reached Eagle is able to analyze the origin goal and formalize, by including the actual situation of the environment, a new goal which is achievable. This is not the origin goal but still satisfies the original objective; hence, we can understand this as a mechanism for weakening goals.

5.3 Conclusions

AI Planning uses a plan for automated problem resolution processes. While the goal resolution techniques for SOAR and BDI agents use a sequence of actions which can lead closer to the finals state AI Planning tries to construct a complete plan at the beginning. This can be done by forward chaining, means the agent start at the front-end (initial state) and continues until it reach the final state, or backward chaining, where the agent start at the final state and works backward step by step until it reaches the initial state. For both methods AI planning is using heuristic algorithm which should help to find the best resolution path. It uses problem decomposition to divide the problem into several sub problems. In AI planning all subgoals can work independently from the others which can lead to some additional work to combine the different result to one plan. Each step in a plan is called an action or operator description which consists of two elements a precondition and a postcondition. The preconditions define the elements which are required to fulfill this action. The post-condition represents the new situation after the action was performed.

An interesting aspect is HTN (Hierarchical Task Network Planning). It divides the problem into a small number of actions an each of this action has the possibility to continue dividing the sub problem into several sub problems. This allows to minimizes the number of individual actions that need to be taken into consideration for planning, thereby potentially leading to higher scalability of planning techniques.

However the pure HTN viewpoint is rather unnatural and so most planning systems are using hybrid approaches. For goal driven architecture it would be interesting to combine the HTN method on the backend and the natural aspect on the front end.

STRIPS is a well known approach, commonly understood as the basis of AI Planning. It introduces the basic concepts regarding the representation of states and goal states and operator descriptions by preconditions and effects, and the basic techniques for planning on basis of backward-chaining. A disadvantage of STRIPS as a goal-driven architecture is that the input must be given in a special format consisting of predefined commands. This can lead to communication problems with other agents which do not "speak" the same language. STRIPS is using a closed world assumption which makes it difficult to enlarge the knowledge with the help of other agents. The fact that STRIPS only support positive literals makes it impossible to recover from failure which is an important aspect for any kind of goal driven architecture.

This recovery from failure is supported by EAGLE, a specification language for extended goals. Eagle supports with ontologisms like **Fail** or **Repeat** a recovery from failure which is in an suitable extension in order to provide a higher expressivity for goal formulation and success of goal resolution in real world settings. Another interesting fact of Eagle is the possibility to weaken a goal. Especially in dynamic environments it is not always possible to reach the predefined goal. In this case a usual agent will give up after a certain time. In this case EAGLE would weak the goal to reach a satisfied state.

6 The UPML Framework

Problem Solving Methods (PSMs) have been developed in the area of Knowledge Engineering as a methodology for formally describing the reasoning process for resolving given problems in Knowledge-based Systems [Fensel, 2000]. The Unified Problem Solving Method Development Language (UPML) provides a framework for formally describing PSMs, consisting of the core element definitions, their formal description specification, and development guidelines in order to ensure consistency and resolvability of a PSM definition [Fensel et al., 2003].

6.1 Introduction

UPML unifies and generalizes the conceptual models for describing knowledge-based systems that have been developed by several approaches in knowledge engineering. Problem Solving Methods (PSMs) have been developed in the area of Knowledge Engineering as a methodology for formally describing the reasoning process for resolving given problems in Knowledge-based Systems [Fensel, 2000]. The Unified Problem Solving Method Development Language (UPML) provides a framework for formally describing PSMs, consisting of the core element definitions, their formal description specification, and development guidelines in order to ensure consistency and resolvability of a PSM definition [Fensel et al., 2003].

Knowledge based systems are roughly spoken computer system which are using knowledge systems to solve problems. This knowledge is represented declaratively, meaning that the functionality of computational facilities is described extensively instead of realizing it in procedural algorithms. The purpose of UPML is to provide a general framework for extensively describe the reasoning steps of knowledge based systems in a declarative manner. UMPL does not aim at describing different software components, but to develop a generalized used knowledge-based system which can be used for different tasks.

In fact, UPML can be understood as an extensive and expressive framework for formally describing the core elements of systems for automated problem solving. The following analyzes the UPML framework with respect to its characteristics as a goal-driven architecture and investigates the model and languages used for the declarative element descriptions.

6.2 Structure of UPML

UMPL identifies six elements that are considered to be relevant for describing the reasoning behaviour of knowledge based systems by applying Problem Solving Methods (PSM). These elements are: (1) ontologies provide the formalized general terminology and knowledge of a domain; for specific applications, (2) domain models extend ontologies with specific domain knowledge these are extended; a (3) a task specifies the problem to be solved, and (4) a PSM is a generic methodology for prob-

lem solving; (5) refiners allow weakening or strengthening of PSMs to make them applicable to concrete tasks, and (6) bridges that allow resolving terminological and teleological mismatches between ontologies, domain models, tasks, and PSMs. Figure 6 shows the interrelation of the six UMPL elements.

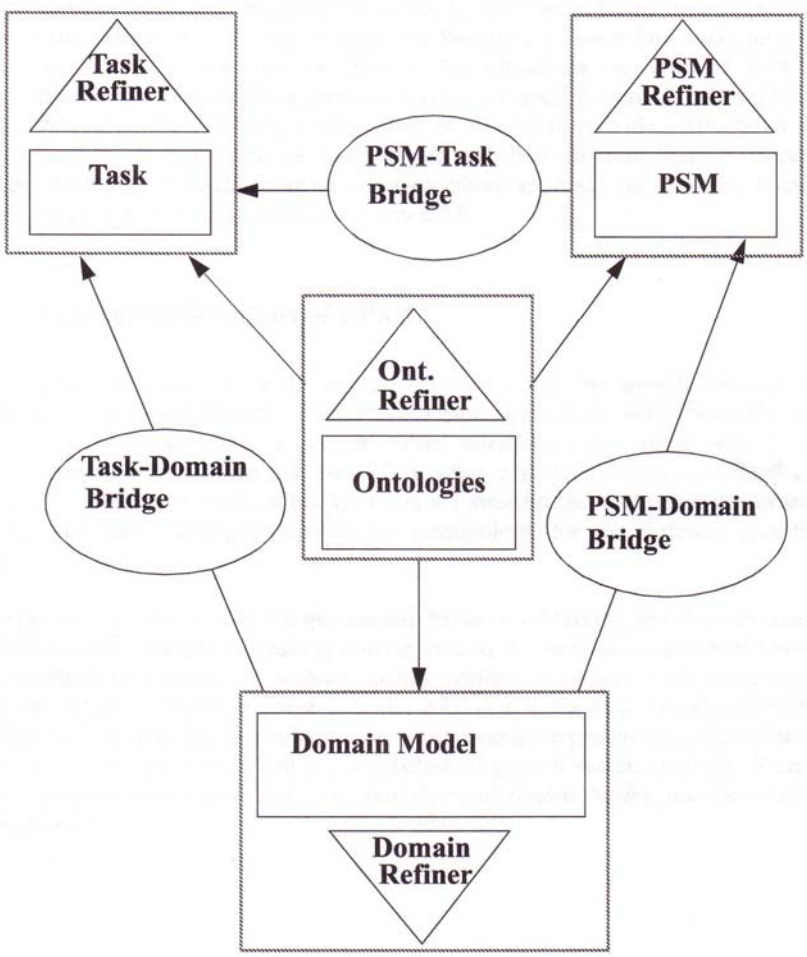


Figure 6: UPML Elements and their Relation[18]

Comparison this with the general structure of goal-driven architectures identified in Section 2.2 reveals the following correlation. UPML tasks denote the client-side element for formal specification of user objectives, and PSMs denote the service-side as general purpose strategies for problem solving; ontologies, domain models, refiners, and bridges denote the auxiliary elements. Hence, UPML can be understood as a description model for goal-driven architectures. Below, we investigate the description models for the different elements and their interrelation in detail.

[18] source: [Fensel et. al., 2003].

6.3 UPML Element Descriptions

The following investigates the distinct UPML elements with respect to their usage purpose and their declarative description models. In particular, we analyze the semantics of the element descriptions. We group the elements in accordance to their allocation within the model for goal-driven architectures.

6.3.1 Ontologies and Domain Models

Ontologies and domain models provide the formal domain terminology and knowledge definitions that are used within all other UPML elements. While ontologies are used as generic, re-usable domain knowledge conceptualizations, domain models refine and extend ontologies for a specific application scenario.

An ontology provides "an explicit specification of a conceptualization" [Gruber, 1993], which can be shared by reasoning components which are communicating during problem resolution process. As in the other AI technologies ontologies are used to define the terminology task for problem resolution. In contrast to the other surveyed approaches UPML does not have a specific language for defining ontologies but rather supports different languages. These are KARL [Fensel, Angele, Studer, 1998], a frame-based language for specifying epistemological aspects close to F-Logic [Kifer et al., 1995], and the Modal Change Logic MCL [Fensel, Groenboom, Lavalette, 1998] for the dynamic aspects; these languages have been combined into the OIL language [Fensel et al., 2001]. Table 6 summarizes the description elements of ontologies in UPML.

Table 6: UPML Ontology Description Model

Descriptor	Explanation
pragmatics	non-functional aspects (e.g. creator, date, natural language description, references)
signature	defines the data schema of an ontology in terms of sorts, constants, and predicates.
axioms	domain knowledge specification in form of logical expressions on basis of the signature

A domain model uses one or more ontologies and extends the knowledge definitions with application specific knowledge specifications. In essence, this means that a domain model provides an ontology that is refined and extended for a specific application purpose. For specification, a domain model imports a general purpose ontology and defines extended knowledge in for of new predicates, axioms, and facts. These are distinguished into *properties* and *assumptions*: while the former can be derived from the domain knowledge, the latter denotes aspects that cannot not be derived but are needed to be assumed to be true. Table 7 summarizes the description structure of domains models.

Table 7: UPML Domain Model Description

Descriptor	Explanation
pragmatics	non-functional aspects (e.g. creator, date, natural language description, references)
ontology	imported ontology (one or more)
properties	schema of additional domain knowledge that can be derived
assumptions	schema additional domain knowledge that is assumed
domain knowledge	concrete facts of domain

6.3.2 Task

The purpose of tasks is to specify problems that a knowledge-based system shall solve. In contrast to goal definitions in the previously surveyed approaches, a task is a generic problem description that can be re-used, i.e. for solving several requests that have the same semantic structure. Therefore, a task represents a schematic problem description that can be initiated for several specific requests.

A task description consists of two aspects: at first it defines a goal to be achieved, and secondly it has information about the in-and output roles as well as preconditions that need to hold before the task can be initiated. Therewith, the problem definition is kept independent from specific applications which give the possibility of reuse for other applications. Another aspect is the explication of assumptions as conditions on the world that need to hold in order to achieve the task. Assumption can be checked during the whole problem solving method while preconditions can not. In this way assumptions can ensure that the given task can be solved for premissable input. Table 8 shows the description model of tasks in UPML.

Table 8: UPML Task Description Model

Descriptor	Explanation
pragmatics	non-functional aspects (e.g. creator, date, natural language description, references)
ontology	imported ontology / domain model (one or more)
specification - roles - goal - precondition - assumptions	specifies the task by: - in- & output constraints - state of the world to be achieved - conditions that need to hold before execution - explicated assumptions and conditions that need to hold during the task resolution

6.3.3 Problem-Solving Methods

PSMs denote the service-side element in the UPML framework. It is to note that a PSM is a procedure or methodology for problem solving and hence is not comparable to operators, i.e. computational facilities available for automated usage and execution. Rather then describing the functionality of an operator as an implemented, executable program, a PSM provides a goal resolution strategy that defines the reasoning procedure to be undertaken. However, under the assumption that an implementation for a PSM exists, this can be understood as an operator for automated problem solving on a higher level of abstraction.

UPML distinguishes between two different types of problem solving methods: complex problem solving method which divides a task into several subtasks and primitive problem-solving methods that makes assumptions about the knowledge of the domain to perform a reasoning step. The central aspects for PSM descriptions in UPML are the *competence* that specifies what the PSM does and the *operational description* that specifies how the PSM achieves it's competence. Table 9 summarizes the PSM description model in UPML.

Table 9: UPML Problem Solving Method Description Model

Descriptor	Explanation
pragmatics	non-functional aspects (e.g. creator, date, natural language description, references)
ontology	imported ontology / domain model (one or more)
competence	specifies WHAT the PSM does by:
- roles	- in- & output constraints
- precondition	- conditions that need to hold before execution
- postcondition	- conditions that hold after execution with dependence to the precondition
- assumptions	- explicated assumptions and conditions that need to hold during the task resolution
- sub-tasks	- lists of tasks the PSM consists of
operational specification	specifies HOW the PSM works by:
- intermediate roles	- in- & outputs needed for PSM execution
- procedures	- programs / procedures used
- control	- control & data flow between sub-tasks and procedures

6.3.4 Refiners and Bridges

As a novel element not existing in the other investigated approaches, so-called adapters in UPML connect other elements in order to make them interoperable if this is not given a priori. Conceptually, this follows the idea of mediators proposed by [Wiederhold, 1992].

A *Refiner* adopts tasks and PSMs by refining their specification, respectively competence such that either a PSM is usable for a task if this has not been given a priori, or such that a tasks or a PSM occurs to be a functional refinement of another one. This is achieved by defining additional conditions and assumptions that constrain the respective functionality descriptions. A *Bridge* adopts two elements and resolves ontological mismatches between them that hamper interoperability. The source and target can be any UPML element. Table 10 summarizes the description model of both refiners and adapters.

Table 10: UPML Adapter Description Model

Descriptor	Explanation
pragmatics	non-functional aspects (e.g. creator, date, natural language description, references)
ontology	imported ontology (one or more)
auxiliary terminology	additional domain terminology & knowledge needed for adapter specification
source element	UPML element that the adaptation starts from
target element	UPML element that the adaptation results in
SPECIFIC FOR REFINERS	
refined x	additional axioms / rules that specify the refinements, whereby “x” can be any sub-element of a task specification or a PSM competence
SPECIFIC FOR BRIDGES	
rename	terminology renaming
mapping axioms	mappings between heterogeneous knowledge definitions

6.4 Conclusion

UPML represents a description model for reasoning procedure of knowledge based systems. In this case UPML is not representing in order to BDI Agent an implemented framework problem resolution. It is using different problem solving methods without the ability of goal resolution techniques, these have been addressed in related system implementations, with IRS II as the most prominent one [Motta et al., 2003].

The central aspects of UPML with respect to exhaustive declarative description of goals and related aspects are:

1) usage of ontologies as the knowledge representation formalisms, which is more expressive than first-order predicate logic and more suitable for concisely describing domain knowledge
2) the description model for tasks, the client-side description element of UPML, represents a more general way for formally specifying client objectives to be achieved. They are described by in- and output roles for computational execution, and the desired final goal state; in contrast to the previously examined approaches, preconditions and assumptions that constrain possible initial state of the world for solving a problem.
3) PSMs represent the service-side element in UPML. Rather than denoting an operator than can be used and executed for automated problem solving, a PSM specifies a specific reasoning behaviour for problem solving; it is described by a competence as a black box description and optionally by an operational specification that consists of sub-tasks and the control flow between them. We can understand this concept as a means for explicitly specifying the goal resolution procedure in a declarative manner.
4) As a novel aspect, UPML introduces the concept of adapters as intermediate elements for connecting and re-use of resources for automated problem solving. Therefore, two types of adapters are distinguished: *refiners* that tries to render the problem more precisely, and *bridges* which can be variable that matches different elements from tasks to each other.

7 Summary, Findings, Conclusions

After detailed analysis of the four approaches for goal-driven architectures, this section summarizes and depicts the central findings of this survey. With respect to the research interest identified introductory, we explicate the commonalities and differences of the approaches with respect to the notion of goals, their formal description, and techniques for goal-driven automated problem solving.

7.1 Summary

Pursuing the grand aim of Artificial Intelligence of creating intelligent systems that perform tasks automatically in a similar way that humans do, the aim of goal-driven architectures is to lift IT-system usage for clients to the knowledge level. Instead of formulating technical requests for available computational resources clients shall be enabled to formulate their objectives as goals that abstract from technical details and - at the same time - carry all information that is needed for detecting and executing the appropriate resources for achieving the objective.

As the central element of such architectures, a goal is a formal, machine-process able knowledge level specification of a client objective that needs to satisfy the following requirements: (1) abstracting from technical details to the highest possible extent, (2) support all possible kind of objectives that clients may have, (3) carry all information needed for automated goal resolution. For determining the state of the art in goal-driven architectures, we have identified the AI disciplines of cognitive architectures, intelligent software agents, planning, and knowledge engineering as relevant. Out of field, we have examined respectively one approach with respect to how goals are formally described, which types of objectives are supported, and which techniques are applied for automated goal resolution.

The first technique investigated in Section 3 is SOAR, a cognitive architecture based on a conceptual model of human cognition. SOAR is using production rules instead of a database for the representation of the knowledge. Goal resolution techniques are implemented by a decision cycle. This cycle consists of 3 main elements: elaboration (represents a condition which is satisfied to attain the current state), decision (determines the operator to be applied next along with preferences for them) and application which is the execution of the defined task. This decision cycle is extended with subgoaling called impasse handling and learning (chunking) which increases the efficiency of a problem resolution process.

The second approach investigated in Section 4 is the belief-desire-intention (BDI) model developed as a philosophical theory for rationale behavior along with a logical formalization for goal-driven intelligent software agents. The approach consists of three main aspects: Belief, desires and intentions. Belief describes the actual knowledge about the world which is considered to be true by the agent. This information is stored in a database. Desires are describing the final desired state the agent wants to achieve. The last element, intentions are representing one or multiple actions that the

agents has committed to achieve a substep towards the final desired state. At the beginning of a resolution process the agent has knowledge about the world. Starting from the knowledge about the world the agent gets the desires by using the knowledge as well as the given input and problem description. Using these two elements the agent tries to find a way through multiple actions to the final desired state by choosing and executing intentions. In this context an intention is representing a fragment of plan which is created by the agent at the beginning and during the resolution process. This so called plan can also be changed during the resolution by getting new facts about the world. This process continues until the agent reaches the final desired state.

As the third approach, Section 5 has investigated AI Planning as a technique for automated construction of *plans* as a valid sequence of actions for reaching a goal state from an initial state. The purpose of planning is to provide automated support for combing several applicable operators for possibly more complex problems that can not be solved by applying just a single operator. In classical planning, objective descriptions in classical planning techniques a restricted to an initial state and a goal state to be achieved; operators are described in terms of preconditions and effects, and plans are determined on basis of forward- or backward chaining. Extensions of the basic planning techniques include conditions on for plan determination (conditional planning), as well as descriptions languages like EAGLE that allow trial- and compensation specifications for goals. In contrast to the BDI approach of interleaved action and planning, the result of successful planning is a suitable sequence of operators for achieving a goal that is determined *a priori*, i.e. the complete control and data flow between used operators is determined before any of the operators is executed. Although this might cause failures in plan execution because of changes in the world that timely occur between the planning and its execution, AI Planning currently receives a renaissance as the basis for functional Web service composition.

The fourth and final approach investigated in Section 6 is the Unified Problem Solving Method Development Language UPML, an exhaustive framework for explicitly describing the reasoning behavior of knowledge-based systems that utilize problem solving methods (PSM). PSMs are a generic problem solving methodologies that are formally described in order to allow their application for different specific problems via refinement. UPML identifies six elements: *ontologies* and *domain models* that specify the domain terminology and knowledge, *tasks* for specifying objectives to be achieved, *problem solving methods* as generic methodologies for problem solving, and adapters for connecting tasks and problem solving methods by refinement (*refiners*) and solving terminology mismatches between them (*bridges*); UPML defines a description meta-model analyzed above in detail. In contrast to the other approaches surveyed above, UPML is merely a description framework but does not encompass any goal resolution techniques. However, it provides valuable insight for advanced declarative descriptions of goal-driven systems: using ontologies as the data model allows more expressive knowledge definitions, and the concept of tasks denotes a generic way for specifying problems or client objectives; furthermore, when understanding PSMs rather as a way to generically specify goal resolution strategies than as operators for automated execution, their UPML specification provides a way for describing more complex client objectives including resolution plan constraints.

As already mentioned UMPL is mainly a description language without a technique for automated goal resolution. However several approaches has been done to implement this description language into a functional framework including automated goal resolution techniques. The most famous framework is called IRS2 (Internet Reasoning Services) form KMI.

7.2 Findings

After summarizing the investigations, we can now examine the commonalities and differences of the examined approaches. Recalling from the introductory examinations in Section 2.4, the aspects of interest for determining the state of the art in goal-driven architectures are how are goals and related client-side elements defined and described, what kind of objectives and problems can be handled by the technology, and which techniques are applied for automated, goal-driven problem solving. In order to attain a concise overview of the state of the art in goal-driven architectures, the following summarizes the central findings with respect to the types and definition of goals as well as the techniques used for automated goal resolution.

7.2.1 Goal Types and Descriptions

Regarding the first two aspects of interest, we observe that three types of problems or objectives can be supported by the examined techniques. We refer to these as **Goal Types**. In addition, we can differentiate two types of constraints that are supported by advanced models for goal description. We refer to them as **Constraint Types**.

The three Goal Types are:

G-I. **Desired State of the World**
This denotes client objectives for creating a new object or state in the world. Examples are to buy a travel ticket by providing an origin, destination, and date as input, or the goal state of the blocks world example (i.e. the state where there is a tower of 3 blocks on the table); commonly, such goals are specified by the desired final state with respect to an initial state. Typically, we find goals of this goal type within classical planning as well as in the SOAR technology.

G-II. **Functions to be Performed**
This refers to objectives for performing a certain function, e.g. *multiply(a,b)* or *withdrawFromAccount(x)*. While the client desire is to change the state of the world by executing a specific operation, the formal description of such goals is commonly given as a \emph{state transition}, i.e. a pre-state constraints and post-state constraints that denote the epistemic change between them. We can find this goal type in all of the surveyed approaches.

G-III. **Temporal Abiding Goals**
This group denotes client objectives that remain over a longer period of time, and typically require several steps for resolution as well as adoption and goal refinement during the resolution process. Examples are to write a book or attain a PhD degree, which we typically find as desires delegated to intelligent agents.

The two Constraint Types are:

C-I. **Goal Resolution Invariants**
This denotes additional constraints that need to hold during all states of the world traversed when a goal is solved. For instance that the account shall never become negative while a series of purchases is performed. We find such constraints implicitly hidden with the domain knowledge in the SOAR technology, as well as in BDI Agents and AI Planning.

C-II. **Goal Resolution Procedure Constraints**
This refers to constraints on the process for resolving a goal. Instead of merely a black-box description for goal from an initial state to the desired goal state, requirements and constraints on intermediate steps and their order are defined in a goal formulation. We find this in form of constraints in conditional AI planning languages, as trial- and compensation specifications in the EaGLe language, and as goal decompositions in terms of collections of sub-goals with control- and data flow between them in UPML for complex PSMs.

Regarding the formal specification of goals, we observe that all investigated approaches apply a state-based model as the underlying logical framework for formally describing goals and operators. For all three goal types, the desired goal states are defined in terms of logical expressions in some static knowledge representation language (propositional logic or some ontology language). The state of the world that holds before the goal resolution procedure is started is either considered to be given as the initial state in terms of facts and rules (SOAR, BDI agents, and planning) or specified as a respective pre-state constraint (UPML).

Both constraint types represent extensions of the respective goal type description. For type C-I, the constraints are commonly modelled as logical conditions in the respective framework with the meaning that these conditions need to hold during all intermediate states traversed during the goal resolution. For constraint type C-II, the constraints are either modeled as logical conditions on particular states that are required to be traversed during the goal resolution procedure (condition AI planning and EaGLe), or as declarative descriptions of goal decompositions in terms of collections of subgoals along with control- and data flow between them (operational specification of complex PSMs in UPML).

Summarizing, defining goals in terms of preconditions, effects, and additional constraints appears to be the least common characteristic of the surveyed approaches.

7.2.2 Goal Resolution Techniques

Regarding the third aspect of interest, we have determined three approaches for automated goal resolution techniques. The decision cycle of SOAR along with subgoaling applies *forward-chaining* in order to subsequently choose the operators for reaching the goal state from the initial state. The central characteristic of automated goal resolution within the BDI framework is *interleaved action and planning*, meaning that the agent observes the world, then determines intentions and executes, and

repeats this process until the final desire is solved. As the third one, AI Planning techniques automatically create the goal resolution plan by matchmaking and forward- or backward-chaining.

Each of these techniques allows to resolve the specific type of goals, whereby the AI Planning as well as the SOAR technique appear to be most suitable for goal types I and II while the BDI technique in principle supports resolution for all goal types but is mainly designed and applicable for type III. As the commonalities of the goal resolution techniques, we observe that each one encompasses facilities for planning and for operator detection. The former is concerned with determining the goal resolution plan as the steps to be performed for reaching the final state from the initial state of goal formulation, and the latter is concerned with . Therefore, AI planning techniques provide the technical core as automated plan determination by finding and combing available operators. This is extended towards interleaved action and planning by subgoaling and means-to-end analysis for operator detection in SOAR, and further into interleaved observation, planning, and action within the BDI framework.

Concluding, Table 11 summarizes the commonalities and differences of the investigated approaches with respect to the goal and constraint types identified, the formal languages used for goal specification, and the applied goal resolution techniques.

Table 11: Goal-driven Architectures – Commonalities and Differences

	SOAR	BDI Agent	AI Planning	UPML
supported Goal & Constraint Types	G-I,G-II C-I	mainly G-III C-I	G-I, G-II C-I, C-II	G-I, G-II C-I, C-II
Goal Description Model	initial state & goal state in problem space	beliefs, desires, intentions	initial & goal state + constraints	**tasks** in- & output precondition goal state assumptions
Specification Language	production rules + propositional logic	**BDI logics** modalities FOL temporal logic possible worlds	propositional logic + plan description (operators, control- & data-flow)	ontologies + MCL for dynamics
Goal Resolution Technique	forward chaining with sub-goaling	interleaved observation / planning / action	a priori plan determination	not in the scope

7.3 Conclusions

Above, we have summarized the survey and explicated the central findings on goal types, their description, and goal resolution techniques developed in existing approaches. In order to avoid duplication and referring to the summarizes for the investigated approaches within the respective sections, the following merely summarizes the central aspects of this survey.

- pursuing the grand aim of AI research, goal-driven architectures provide sophisticated client-side support for automated IT-system usage on the knowledge level with its philosophical origins in Cognitive Science
- the general structure of goal-driven architecture consists of 3 elements: goals as the client-side element for objective formulation, service-side elements as available computational facilities with declarative descriptions, and auxiliary elements for enhancing the goal resolution quality
- goals are *formal, machine-processable knowledge level specification of client objectives* that (1) abstracts from technical details to the highest possible , extent, (2) support all possible kind of objectives that clients may have, and (3) carry all information needed for automated goal resolution
- we have determined 3 goal types: (1) desired states of the world, (2) functions to be performed, and (3) temporal abiding goals; the common underlying model for formal descriptions of goals and related aspects are state-based so that the commonly goal descriptions consist of preconditions and effects as state constraints on the initial and final state of the world
- goal descriptions can be extended with constraints, wherefore we have distinguished (1) invariants that are requested to hold during each state traversed during the goal resolution, and (2) procedural constraints that need to hold on particular states that are required to be traversed for goal resolution; the former is described as additional constraints on the requested functionality, and the latter by control- and data flow of goal decompositions
- the common core of goal resolution techniques are facilities for planning and operator detection; specific techniques extend this with sub-goaling and means-to-end analysis for operator selection, respectively interleaved observation, planning, and action for temporal abiding goals.

References

Allen, J, Austin, T., Hendler, J. *Readings in Planning*. Morgan Kaufmann. Publishers, 1990.

Anderson, J. R.: Cognitive Architectures in a rational analysis. In K. van Lehn (ed.), *Architectures for Intelligence*, pp. 1-24, Lawrence Erlbaum Associates, Hillsdale, N.J, 1991.

Anderson, J. R.: *Cognitive Psychology and Its Implications. 5th Edition*. New York, USA: Worth Publishers and W. H. Freeman, 1999.

Blackburn, P; de Rijke, M.; Venema, Y: *Modal Logic*. Cambridge University Press, 2001.

Boerger, E. and Staerk, R.F: *Abstract State Machines. A Method for High-Level System Design and Analysis*. Berlin, Heidelberg: Springer 2003.

Booth, D. et al.: *Web Service Architecture*. W3C Working Group Note 11 February 2004. available at: http://www.w3.org/TR/ws-arch/.

Bratman, M. E. ; Israel, D. J.; Pollack, M. E.. *Plans and resource-bounded practical reasoning.* Computational Intelligence 4 (4), pp. 349 – 355 1988.

Bratman, M. E.: *Intention, Plans and Practical Reason.* Harvard University Press, Cambridge, Massachusetts, 1987.

Chen, Y. and Cheng, B. H.C.: A Semantic Foundation for Specification Matching. In G. T Leavens and M. Sitaraman: *Foundations of Component-based Systems*, Cambridge University Press, 2000.

Codd, E. F.: *Relational Completeness of Data Base Sublanguages*. In R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California, 1972.

Cohen, P. R. and Levesque, H. J.: *Intention is choice with commitment*. In Artificial Intelligence 42, pp. 213 - 261, 1990.

Diller, A.: *Z: An Introduction to Formal Methods. 2nd edition*. Wiley, 1994.

E.A. Emerson. Temporal and modal logic. In. J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier Science Publishers,1990

Erl, T.: *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services.* Prentice Hall PTR, 2004.

Fensel, D. and Bussler, C.: *The Web Service Modeling Framework WSMF.* Electronic Commerce Research and Applications, 1(2), 2002.

Fensel, D. et al.: *The Unified Problem Solving Method Development Language UPML*. In Knowledge and Information Systems Journal (KAIS) 5(1), 2003.

Fensel, D.: *Problem Solving Methods: Understanding, Description, Development and Reuse*. Berlin, Heidelberg: Springer 2000.

Fensel, D.; Angele, J.; Studer, R.: *The Knowledge Acquisition and Representation Language KARL*, In IEEE Transactions on Knowledge and Data Engineering, 1998.

Fensel, D.; Groenboom, R.; Renardel de Lavalette, G. R.: *Modal Change Logic (MCL): Specifying the Reasoning of Knowledge-based Systems*. In Data and Knowledge Engineering (DKE), 26(3):243-269, 1998.

Fensel, D.; van Harmelen, F.; Horrocks I.; McGuinness, D. L.; Patel-Schneider, P.: *OIL: An Ontol-*

ogy Infrastructure for the Semantic Web. IEEE Intelligent Systems, 16(2), 2001.

Forgy, C.: *OPS5 User's Manual*. Technical Report CMU-CS-81-135, Carnegie Mellon University, 1981.

Ghallab, M.; Nau, D., and Traverso. P. *Automated Planning. Theory & Practice*. Morgan Kaufmann Publishers, 2004.

Grosz, B. and C. Sidner.: Plans for Discourse. In P. Cohen, J. Morgan, M. Pollack (Eds.): *Intentions in Communication*, Bradford Books, MIT Press, 1990.

Hoek, W. v. d. and Wooldridge, M.: *Towards a Logic of Rationale Agency*. Logic Journal of the IGPL 11(2): 135-159, 2003.

Hofstadter, D. R.: *Goedel, Escher, Bach: an Eternal Golden Braid*, New York, USA: Basic Books, 1979.

Jeng, J.-J. and Cheng, B. H. C.: *Using Formal Methods to Construct a Software Component Library*. In Proc. of the 4th European Software Engineering Conference on Software Engineering, LNCS (717), p. 397-417, 1993.

Jennings, N.R.; Wooldridge, M.: Agent-oriented Software Engineering. In J. Bradshaw (Ed.): *Handbook of Agent Technology*, AAAI Press / MIT Press, 2001.

Kifer, M.; Lausen, G.; Wu, J.: *Logical Foundations of Object-Oriented and Frame-Based Languages.* Journal of the ACM, 42(4):741–843, 1995.

Krueger, C. W.: *Software Reuse*. In ACM Computing Surveys (CSUR), vol.24(2), p.131-183, 1992.

Laird, J. E.; Congdon, C. B.: The Soar User's Manual, Version 8.5, Edition 1. University of Michigan, June 3, 2004; http://www.eecs.umich.edu/~soar/sitemaker/docs/manuals/Soar8Manual.pdf.

Lamsweerde, A. van: *From System Goals to Software Architecture*. In SFM 2003, pp. 25-43.

Lamsweerde, A. van; Letier, E.: *From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering.* In Proc. of the 9th International Workshop on Radical Innovations of Software and Systems Engineering (RISSEF 2002), Venice, Italy, 2002.

Lausen, H.; Polleres, A.; Roman, D. (Eds.): *The Web Service Modeling Ontology (WSMO).* W3C member submission 3 June 2005; http://www.w3.org/Submission/WSMO/.

Levesque, H. J.; Cohen, P.R.; Nunes, J. H. T.: *On Acting Together*. In Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90), pp 94-99, Boston 1990.

Luck, M.; McBurney, P.; Preist, C.: *Agent Technology: Enabling Next Generation Computing - A Roadmap for Agent-Based Computing*. Version 1.0, AgentLink II, 2003.

Marco Pistore, Fabio Barbon, Piergiorgio Bertoli, D. Shaparau, Paolo Traverso: Planning Monitoring Web Service Composition. AIMSA 2004: 106-115

Martin, D. L.; Cheyer, A. J.; Moran, D. B.: *The open agent architecture: a framework for building distributed software systems,* Applied Artificial Intelligence 13 (1-2), 1999.

McCarthy, J.: Artificial intelligence, Logic and Formalizing Common Sense. In R. Thomason (Ed.): *Philosophical Logic and Artificial Intelligence*, pp 161--190. Kluwer Academic Press, Dordrecht, Holland, 1989.

McCarthy, J.: *From Here to Human Level AI.* Stanford CS Department Technical Report, 1996; available at: http://www-formal.stanford.edu/jmc/human/.

Meyer, B.: *Object-Oriented Software Construction*. Prentice-Hall, Englewood Cliffs, 2nd edition, 1997.

Moore, R. C.: *Reasoning about knowledge and action*. In Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77), Cambridge, MA, 1977.

Motta, E., Domingue, J., Cabral, L. and Gaspari, M. (2003) IRS-II: A Framework and Infrastructure for Semantic Web Services. 2nd International Semantic Web Conference (ISWC2003) 20-23 October 2003, Sundial Resort, Sanibel Island, Florida, USA.

Nadel, L. (Ed.): *Encyclopedia of Cognitive Science*. London, UK: Nature Publishing Group, 2003.

Newell, A. and Simon, H. A.: GPS, a program that simulates human thought. In E. A. Feigenbaum and J. Feldman: *Computers and Thought*. New York: McGraw-Hill, pp. 279 – 293, 1963.

Newell, A. and Simon, H. A.: *Human Problem Solving*. Englewood Cliffs, New Jersey: Prentice-Hall, 1972.

Newell, A. *The Knowledge Level*. In Artificial Intelligence (18), pp. 87-122, 1982.

Newell, A.: *Unified Theories of Cognition*. Cambridge, Massachusetts, USA: Harvard University Press, 1990.

Nwana, H. S.: *Software Agents: An Overview*, Knowledge Engineering Review 11(3), 1996; pp 1-40.

Paolucci, M.; Kawamura, T.; Payne, T.; Sycara, K.: *Semantic Matching of Web Services Capabilities*. In Proceedings of the First International Semantic Web Conference, Springer-Verlag, 2002; pp 333-347.

Preist, C.: *A Conceptual Architecture for Semantic Web Services*. In Proceedings of the 3rd International Semantic Web Conference (ISWC 2004), 2004, pp. 395 - 409.

Rao, A. S. and Georgeff, M. P.: *Modeling rational agents within a BDI-architecture*. In Proceedings of Knowledge Representation and Reasoning (KR&R-91), pages 473 – 484, San Mateo, CA, 1991.

Rao, A. S. and Georgeff, M: *Decision procedures for BDI logics*. Journal of Logic and Computation 8(3), pp. 293- 344, 1998.

Rosenbloom, P. S., and Newell, A.: *Learning by chunking: Summary of a task and a model.* In Proceedings of AAAI-82 National Conference on Artificial Intelligence. AAAI, Menlo Park, CA, 1982.

Rosenbloom, P. S., Laird, J. E.; and Newell, A.. *The Soar Papers: Readings on Integrated Intelligence*. Cambridge, MA: MIT Press, 1993.

Russell, S. and Norvig, P.: Artificial Intelligence. A Modern Approach. 2nd edition. Prentice Hall, 2003.

Shoham, Y.: *Agent-oriented Programming.* In Artificial Intelligence (60), pp. 51-92, 1993.

SRI: Procedural Reasoning System, User's Guide, Artificial Intelligence Center SRI International 333 Ravenswood Avenue Menlo Park, CA 94025

Stollberg, M.; Strang, T.; Fensel, D.: *Automated Collaboration on the Semantic Web*. In GESTS International Transactions on Computer Science and Engineering 17(1), September 2005.

Studer, R.; Benjamins, V.R.; Fensel, D.: *Knowledge Engineering. Principles and Methods*. In Data and Knowledge Engineering 25 (1-2) 1998.

Sycara, K.; Paolucci, M.; van Velsen, M.; Giampapa, J. A.: *The RETSINA MAS Infrastructure*. In Special Joint Issue of Autonomous Agents and MAS, Volume 7 (1,2), July 2003.

Turing , A. M.: *Computing Machinery and Intelligence*. In Mind (49), pp. 433-460, 1950.

Wiederhold, G.: *Mediators in the Architecture of Future Information Systems.* In IEEE Computer, 25(3):38.49, 1992.

Wilsker, B.: *A Study of Multi-Agent Collaboration Theories*. ISI Research Report, ISI/RR-96-449, November, 1996.

Wilson, R. A., & Keil, F. C. (Eds.): *The MIT Encyclopedia of the Cognitive Sciences*. Cambridge, MA: MIT Press, 1999.

Wooldridge, M. and Jennings, N.: *Intelligent Agents: Theory and Practice*, Knowledge Engineering Review 10(2), 1995; pp 115-152.

Wooldridge, M. and Rao, A. (eds.): *Foundations of Rational Agency*. Kluwer Academic Publishers, 1999.

Wooldridge, M. J.: *Reasoning about Rationale Agents*. Cambridge MA: The MIT Press, 2000.

Wooldridge, M.: *An Introduction to Multi Agent Systems*. Wiley and Sons, 2002.