Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research
Report
RR-98-01

Methodological Comparison of Agent Models

Christoph G. Jung and Klaus Fischer

October 1998

Deutsches Forschungszentrum für Künstliche Intelligenz
GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210
E-Mail: info@dfki.uni-kl.de

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341
E-Mail: info@dfki.de

WWW: http://www.dfki.de

Methodological Comparison of Agent Models

Christoph G. Jung and Klaus Fischer

DFKI-RR-98-01

This work has been supported by a grant from The Federal Ministry of Education, Science, Research, and Technology (FKZ ITW-95004).

ISSN 0946-008X

Methodological Comparison of Agent Models

Christoph G. Jung[*] and Klaus Fischer
GK Kogwiss. & MAS Group, FB Informatik

Universität des Saarlandes & DFKI GmbH
Im Stadtwald, D-66123 Saarbrücken, Germany
{jung,kuf}@dfki.de

April 1998

[*]supported by a grant from the "Deutsche Forschungsgemeinschaft" (DFG).

Abstract

Hybrid agent architectures comprise the radical change of paradigms in AI over the past decades by reconciling the different styles of reactive, deliberative, even social systems. They have been successfully applied to a range of complex real-world domains. Due to their originally informal background, a verification of design goals in derived implementations, theoretical foundations, and a detailed comparison with other agent models have not yet been obvious. The present work proposes a formal methodology to bridge the gap between theoretical and practical aspects especially of hybrid designs, such as the layered InteRRaP. The employed, connected stages of specification, i.e., architecture, computational model, theory, proof calculus, and implementation, also provide a yet unique framework for comparing heterogeneous agent models including unified and logic-based ones. Based on recent work on InteRRaP, we demonstrate that this methodology allows to compare state-of-the-art designs from robotics, AI, computer science, and cognitive science with respect to a spectrum of inherent properties along the two dimensions of abstraction and declarativity. This supports our claim that InteRRaP is a coherent and advanced account of layered agency including goal-oriented abstraction planning in on-line interaction with reactive skills and social reasoning. We also derive particular research issues to guide the future development of InteRRaP.

Contents

Chapter 1

A Methodology for (Hybrid) Agents

Today, the application areas for autonomous and intelligent agent technology open up enormously. Especially demanding real-world domains, such as (tele-) robotics, flexible manufacturing, transport telematics, as well as upcoming virtual worlds, such as global information networking, and interactive movies, require broad agents. These are agents that solve a range of cognitive problems, from reactive behaviour and deliberative tasking up to social interaction, in a practical manner[1].

Hybrid systems are systems that integrate different functionalities of separate modules by rather pragmatic and operational interaction patterns. They are especially successful in constructing broad agents. InteRRaP [28], for example, models the smooth transition from sub-symbolic reactivity, over symbolic deliberation up to social reasoning by *layering*, i.e., by defining a hierarchical control structure between these modules. Such descriptions are straightforwardly done in an architectural manner. Since this informal method of specification introduces very crude and abstract concepts, the space of possible implementations does not necessarily reflect the original aim, such as the practical integration of reactivity and deliberation in the case of the InteRRaP.

On the other hand, formal and declarative languages are used in the tradition of theories of rationality. For example, [33, 31] build on earlier work in temporal and epistemic logics. [26] rely on the power of first-order logic augmented with abduction as a declarative representational and inferential basis for a unified agent. Similar to the purely architectural approach, the conceptual level

[1]The term *broad* agent is due to [2]. He claims that synthetic characters, for example, should rather exhibit a range of 'shallow' cognitive abilities than a particular ability exceedingly well. Our definition of *broad* in contrast is derived from the *bounded rationality* paradigm: the broad agent is required to exhibit a range of cognitive abilities in an approximately optimal manner. Approximately, because depending on environmental and computational constraints, these agents trade-off decision quality versus computation and interaction cost.

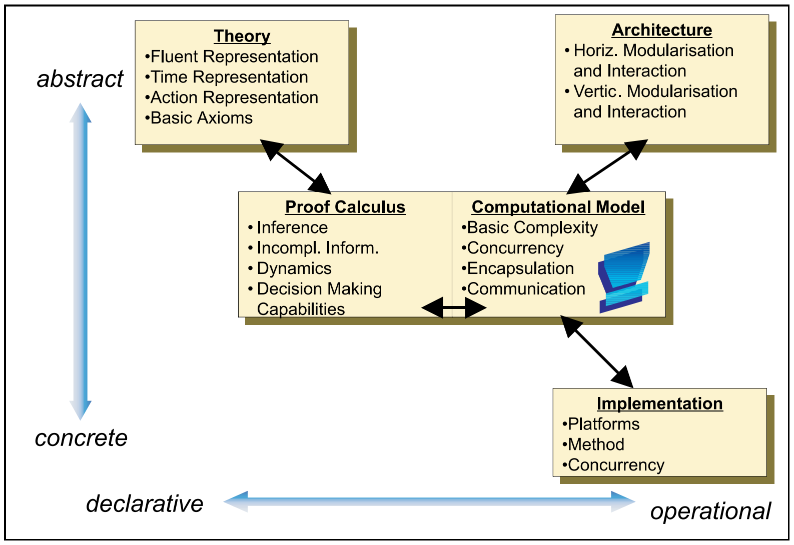

Figure 1.1: The Design Space of Agents

of logic-based agents remains too abstract to derive concrete and practical programs. Straight implementations of corresponding proof calculi, although being promoted by Logic Programming (LP)[24], are either not feasible or build on a restricted expressivity. Of course, there are subtle differences between theories for specifying the agent's computation and theories for describing agent behaviour. In a normative setting such as the present report presumes, we could however regard the agent as a proof procedure for a restricted part of a descriptive theory making general statements about its state. Operational considerations to 'make the theory run' are however seldom part of the corresponding agent model.

Both approaches to agent design are thus rather complementary. Both types of specifications either lack declarative (in the architectural case) or operational (in the theoretical case) aspects which makes them hard to compare. Furthermore, both do not address important implementation issues. Thus, a formal methodology that connects theoretical and practical aspects of agent design is desirable. Such a methodology furthermore helps to set up a well-understood collection of design methods and their realisation in programs.

We have depicted the relations between specification languages as we assimilate them in Figure 1.1. Basically, we recognise two independent dimensions of specification, namely the degree of abstraction and the declarativity of the described concepts. The hereby spawned space of design identifies the purely architectural attempt as a rather operational and abstract enterprise while the complementary theoretical issues focus on declarative definitions. Furthermore, implementations are the most operational and concrete agent descriptions. Although they are a too low-level medium of research, their connection to the higher-level specifications is nevertheless extremely important. This has been often neglected in the past; even LP cannot fully realise this connection.

As Figure 1.1 suggests and as we have argued in [20], the point of concern is to find an operational extension to the stage of a proof calculus that explicitly captures the practical features of, e.g., hybrid systems: concurrent and encapsulated inference, partial inconsistency, non-monotonic behaviour, explicit communication, and meta-control. We call such an extension a *computational model*, inspired by the calculi developed in formal programming methods [18]. Standardised specification languages, such as Z [35], are used to describe state and operation of an agent abstract machine [22, 20] that embraces an injected logic and its inference principle. In combination with the proof calculus, computational models complete the 'methodological puzzle' because they are sufficiently:

- **abstract** to formally connect to both theories and architectures,
- **concrete** to derive concise implementations,
- **declarative** to verify the theoretical design, and
- **operational** to capture practical considerations.

Interestingly, such a complete methodology does also provide a reasonable framework for the comparison of heterogeneous agent models. By investigating specification languages and thus specification properties along the two dimensions of abstraction and declarativity, it is possible to identify detailed criteria of design across models. This has not been trivial before and, often, overview papers have treated models from different backgrounds separately, such as those from robotics, Artificial Intelligence (AI), Distributed Artificial Intelligence (DAI), computer science, and cognitive science. In contrast, our categorisation into specification stages (*architecture*, *theory*, *computational model*, *proof calculus*, and *implementation*) merges these various research threads and highlights particular similarities and differences. These stages are necessarily overlapping because there should be intimate, formal connections (see Figure 1.1) to comprise a consistent overall model. This implies that a particular issue attributed to a certain stage has impact to other design options on other stages. Although many models lack a complete characterisation, appropriate properties can thus be inferred. In the following, we like to discuss a reasonable assignment of such detailed properties to the proposed stages of specification.

Architecture Its architecture characterises the agent by *modularisation*, i.e., the identification of functional roles, and by the (structured) *interactions* between these modules. Modularisation could be hierarchically structured, if modules are able to control subordinate modules. This is called *horizontal* modularisation[2].

[2]Horizontal modularisation is often used equivalent to *layering*. We will elaborate in this report that layering in INTERRAP, for example, is a special form of horizontal modularisation building on meta-control.

Equally 'privileged' modules servicing each other are introduced by *vertical* modularisation. Hereby, the direction of services determines the flow of computation, thus the flexibility of the architecture.

Computational Model Being an abstract machine for the agent, the computational model explores operational ingredients, such as the *encapsulation*, i.e., the granularity of state and computation, into *processes*. Encapsulation often follows the architectural modularisation and determines the degree of independence and thus possible inconsistencies between states.

Encapsulation is also relevant for obtaining *concurrency* in which the overall behaviour of the agent is composed out of the rather independent and interleaved behaviour of separate processes. Fully independent concurrency runs asynchronously which supports responsiveness and flexibility. This is not to say that non-concurrent computational models could not exhibit interactive behaviour. To gain a similar degree of responsiveness, however, the agent programmer has to add additional interaction and scheduling mechanisms into the domain-dependent part of the agent. A good computational model, however, integrates those facilities required in most domains, thus eases the programmer's task.

Process *communication* is responsible to transfer computational results between encapsulated computations, thus incarnating the module interactions or services. We distinguish explicit forms of communication, such as directed *signals* and undirected *events*. *Shared memory*, for example by logical data structures, is an implicit form of process communication. Horizontal interactions are modelled by influencing computation and communication of subordinate processes.

Similar to concurrency, the transmission of information either follows an asynchronous or a synchronous scheme. The paths of transmission between encapsulated processes can be either determined in advance (static) or being dynamic which leaves the choice of destination to the processes themselves. Unsafe communication models the loss of information.

In the context of computation, design options with particular advantages and drawbacks are to choose between complex[3] and simple procedures as the basic building blocks. Complex system behaviour can be both modelled by a single, complex procedure as by the interaction of separate simple

[3] *Complex* results denote long-term structures out of primitive system operations whose composition produces an optimal answer. Generally, the corresponding procedures turn out to be complex, too, in the sense that their amount of computation increases exponentially in problem size. *Simple* results are therefore regarded as primitive measures that maximise the system's performance just for the next, single step. Often, they can be computed using fairly un-demanding, therefore simple procedures.

ones. Optimality is easier to achieve with a monolithic approach, whereas robustness and flexibility favourite distribution.

Theory As aforementioned, our methodology proposes the computational model to embrace computations that are interpreted as inferences of a single or separate agent logics. Of course, this interpretation is mainly relevant for the 'symbolic', cognitive facilities of the agent.

The theoretical investigation of their possible states therefore reveals representational issues: are facts (fluents) used to describe environmental situations (sub-symbolic, symbolic), to denote the 'mental' state of other agents (epistemic) or even reflect inner-agent circumstances (auto-epistemic)? Is there a notion of time? Is time explicitly present or rather implicit within other representations? What is the underlying model of time ? How are primitive actions encoded? Is there the ability to describe macro actions?

Similarly, theory also captures the declarative part of the inference of cognitive processes, thus the basic axioms with which the agent reasons about facts, time, and actions. Here, we distinguish between situation-based approaches whose reasoning focusses on representing situations and plan-based approaches that rather explore temporal structures between actions.

Proof Calculus A proof calculus transfers declarative considerations of the theory into a computational recipe by means of possible inference steps. Perhaps a minor aspect as mostly being satisfied is to investigate soundness and completeness in the agent context.

It seems more important to highlight the inherent inference principle, such as deduction or abduction. For example, its coherent treatment of incomplete knowledge (restricted perception, dynamic changes, inconsistencies), renders abduction a reasonable basis for situated agents as well as for particular modules within the situated agent. In a deductive setting, this is not to be reached without particular non-monotonic extensions. These extensions typically do not cover all of those aspects of situatedness.

Theory and underlying proof calculus form the basic decision making capability of the agent. Traditionally, such planning procedures are characterised by the temporal horizon that their decisions are based on: present-oriented approaches use the current situation of the agent to select actions to be performed first. Often, these approaches tend to be myopic since turning down the role of the ultimate goal of the plan. In contrast, future-oriented planning rather uses goal-subgoal relations to build a plan from its tail. It however tends to stick with future, irrelevant details before coming to the currently executable decision. Abstraction planning is an intermediate form of both approaches in which abstract decisions (macro-operators) quickly connect the ultimate goal to the current situation.

Implementation For reasons of concreteness and operationalisation, the implementation stage is not an appropriate medium for comparison. It is nevertheless enlightening to observe the range of platforms that a model is successfully applied to according to the conciseness of its higher-level specification.

Implementations can be derived in different ways from a high-level specification. Mostly, they are loosely coupled, i.e., the high-level specification illustrates the principles somehow turned into source code. On the other hand, formal approaches allow to directly translate at least parts of the code from well-defined relations between high-level languages and the implementation language.

A particular point of interest is the final realisation of concurrency. Often, a left-open function in the specification for scheduling computation is not chosen until this stage of design but could have great impact onto the agent's behaviour.

Our proposed full-range methodology has been especially developed in the context of the layered InteRRaP [28] agent architecture. Thus it is one of the first models for which all specification stages have been instantiated. In Chapter 2, we will comprise these results and demonstrate the applicability of the methodology to this practical agent framework while retaining a clear categorisation of properties. In particular, the architectural extensions (Section 2.1) to the original publication [28] lead to the definition of the COOP machine (Section 2.2), a computational model that reflects the architectural design in the form of concurrent and continuous (sub-)cognitive processes. These processes are then uniquely described in Section 2.3 as sub-languages of coherent theory of time and action under-pinned with an abductive proof calculus. From both the proof calculus and the computational model, a well-defined implementation (Section 2.4) can be derived.

Since the description of InteRRaP covers the complete space of specification, it is an excellent tool for comparing various agent approaches from very different scientific backgrounds, such as robotics, AI, DAI, computer science, and cognitive science.

For example, Chapter 3 discusses similarities and differences to the sub-symbolic Subsumption agents of [5] (Section 3.1). We also explore reactive reasoning in unified agent models in Section 3.2 (dMars of [16, 9]), Section 3.3 (Agent0 of [33]) and Section 3.4 (ALP agents of [26]). With respect to their reactive schemes (and also the InteRRaP Behaviour Based Layer — BBL) , we find InteRRaP to enrich the myopic decision making into a more deliberative, goal-oriented scheme.

Deliberative, cognitive agent frameworks, such as PRODIGY [7] (Section 4.1), SOAR [29] (Section 4.2), and ACT-R [1] (Section:4.3) are discussed in Chapter

4. In this context, InteRRaP is unique with its independent, reactive core that is nevertheless steadily interacting with situated planning in local and social settings. We also state that the up-to-date deliberative models emphasise the adaptiveness of reasoning, a perspective that extends the role of layering in InteRRaP.

Finally, we investigate other hybrid architectures (Section 5.1: 3T [4]) and generic toolkits (Section 5.2: DESIRE [10]) in Chapter 5. This comparison also supports our claim of InteRRaP being an advanced model of hybrid agents due to a coherent and reasonable principle of layering and a fully-instantiated methodology ranging from a flexible, theoretical perspective down to a practical implementation.

Due to their heterogeneous origins, the compared models focus on particular parts of the specification space, such as architectural or theoretical questions. Therefore, we have chosen to rely on the most specific, 'official' instance of the models, for example the Event Calculus (EC) based agents of [8] in the case of the more general Abductive Logic Programming (ALP) agents of [26]. Most of the lacking specification properties can be inferred from these descriptions, such as the representational issues of a (possible) theory. As it is not straightforward to suppose a common logic behind every aspect of computation, we focus these investigations on symbolic, cognitive decision making abilities, thus planning.

Finally, Chapter 6 concludes to a complete overview of the investigated specification properties. Indeed, this characterisation inherent to our methodological attempt turns out to identify a collection of independent design features used across different threads of research. Although there are numerous other papers that give thorough overviews to agent design, see, e.g., [38, 28], such a result has not yet been presented, because we do not stick with scientific backgrounds and the applied notion of agency.

Chapter 2

InteRRaP: Theory and Practice of Hybrid Agents

This chapter instantiates the proposed methodology for (hybrid) agent design with the InteRRaP agent model that originally started as an architectural enterprise [28] towards the integration of reactive, deliberative, and even social capabilities in DAI. Up to now, InteRRaPis applied to robotic domains, such as an automated loading dock, transport telematics, and virtual-world applications such a telerobotics and interactive simulation.

In Section 2.1, we recognise that InteRRaP takes a hybrid, layered approach, i.e., the incorporation of these capabilities as separate, horizontal modules. The absence of a formal specification and thus the open space of implementations has led to the definition of a computational model [22, 20] that reflects the architectural design, but ties the framework to a unique abstract machine (Section 2.2). Indeed, the approach taken herein further splits up the reactive, deliberative, and social capabilities into separate (sub-)cognitive processes with a flexible communication structure. Each of those processes is then regarded as a special sub-logic of a unified theory (Section 2.3). In Section 2.4, the relation of its proof calculus and the computational model to a logical host language is described.

2.1 The InteRRaP Architecture

As a hybrid attempt, [28] employs the device of modularisation to integrate the originally complementary perspectives of reactive, situated agents , e.g., [5], with deliberative reasoning agents, e.g., [11]. Additionally, the proposed architectural solution, InteRRaP, caters for social capabilities, an aspect that is often neglected in single-agent designs. Indeed, these three modules interact in a very subtle manner. They are regarded as *layers* stacked on top of one another (Figure 2.1). Each layer in turn implements a vertical chain from knowledge update and goal activation (SG), into decision making and plan execution (PS). The PS module

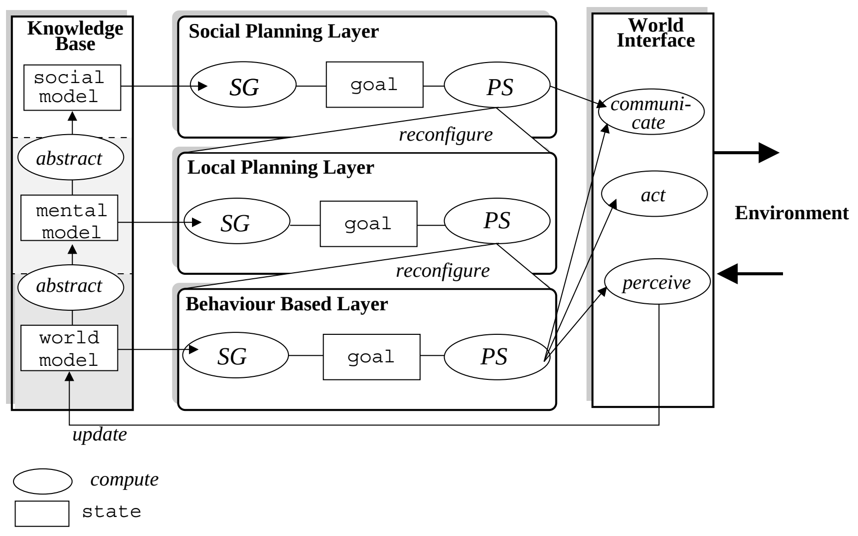

Figure 2.1: The InteRRaP Architecture

is not only feeded with goals and situations, but also reports its decisions and actions back. These vertical modules operate on a particular level of abstraction that is characteristic for each layer.

The most concrete level is implemented by the behaviour-based layer (BBL) that applies procedural knowledge, or so-called patterns of behaviour, to install a fast feedback loop with the environment. On top of this reactive module, the local planning layer (LPL) reasons (plans) about symbolic goals and hypothetical states of the world. Its decisions or primitive actions influence the reactive, underlying module in order to obey the more abstract goals.

[22] argues that this is a form of meta-reasoning. A plan is not just an abstraction of behaviour pattern activation. It merely represents a temporally structured series of configuration constraints to the whole BBL module. These configurations include the parameterisation of behaviour patterns as well as their (de-)activation. An example from the automated loading dock is the ability of forklift agents to deliberately shut off collision avoidance facilities in the BBL while approaching a particular object to manipulate.

Similarly, the relation between social reasoning and local reasoning can be characterised as a meta-object relationship. Negotiating with other agents involves speech acts that have the state of the local planning module (the knowledge, the goals, the plans) as their topic and, e.g., a commitment results into reconfiguring the LPL accordingly (adopting or removing knowledge and goals; changing a plan; etc.).

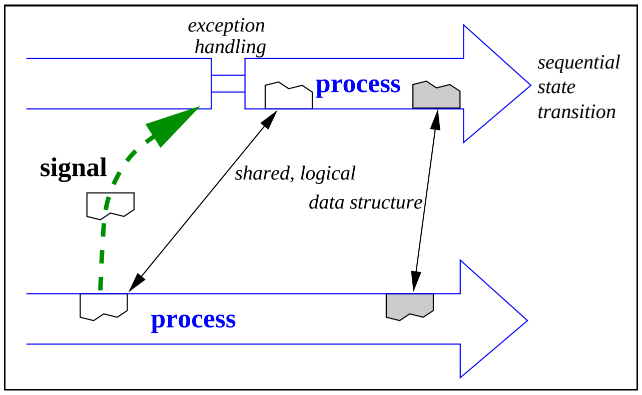

Figure 2.2: COOP: Concurrent, Continuous Processes

2.2 The COOP computational model

InteRRaP has given rise to several implementations on different platforms. In the first implementation the BBL was implemented with a forward chaining rule interpreter and the two higher layers LPL and SPL in PROLOG. Because of the use of two different programming languages the separation of the BBL and the two higher layers was rather strict. This led to the idea to implement InteRRaP in a uniform manner using the high-level programming language Oz. However, the first approach of doing so had problems to realise the original design goals of rational and reactive behaviour, especially in demanding robotic domains. This has been due to the fact that concurrency has been applied per layer which diminished the responsiveness of the BBL with respect to the former, production rule system.

[22, 20], attribute this circumstance to a more general problem of purely architecturally as well as of purely theoretically driven specification: both are too abstract to define a concrete implementation space. We therefore proposed to set up an intermediate stage of specifying computational models as an appropriate device for bridging this gap. The COOP model provides such a basis for InteRRaP in form of a term rewriting calculus [22] and in form of a Z specification [20] and introduces a higher degree of concurrency throughout the architecture, but especially for the BBL.

There are sophisticated constructs in operating systems and modern programming languages, such as those based on *processes* and *signals* (Fig. 2.2), that can be used to encapsulate the inferences within an agent. Thus, a process can be seen as sequential state transition whose state corresponds to a restricted logical sublanguage and whose transition relation implements a well-defined, corresponding subset of the proof procedure. Processes compute continuously in an asynchronously *concurrent*, fair manner. A critical computation, such as an inference based on incomplete knowledge, is shielded by a stack of *exception handlers*.

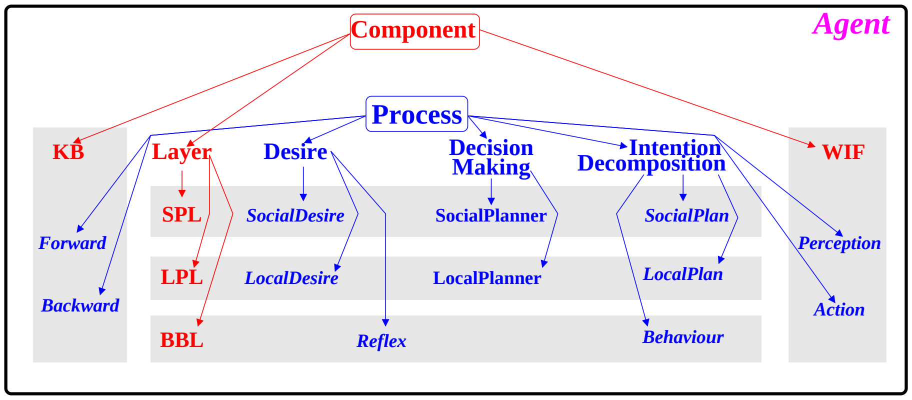

Figure 2.3: InteRRaP: Processes and Control

Asynchronous signals that are dynamically routed indicate explicit, exceptional situations which require immediate reactions, such as revision of inference. Exception handling mechanisms designed for concurrent settings determine an appropriate continuation state for the computation. Because signals contain *logical formulae* to be incorporated into the processes' state, the processes refer to *shared memory* which establishes an additional, implicit form of communication.

In keeping with the architecture, Figure 2.3 does not model the agent as a heterogeneous mixture of such processes communicating directly. Instead, we introduce the *component* as an additional control process that encapsulates other inferences like a membrane. This membrane controls the activation of fresh, internal computations and serves as a fast relays and filter for every incoming and outgoing signal. Components either contain special services, such as the world interface (WIF) and the knowledge base (KB) do, or they represent a layer (BBL, LPL, and SPL).

2.3 The Hierarchical Event Calculus: Theory, Proof Calculus

Originally, Event Calculus [27] (EC) reasoning has been used as an LP approach to specify and implement the planning processes in InteRRaP [23]. Solving the classical frame problem in an efficient manner by a special axiomatisation of strict inertia, the EC is a flexible base for complex reasoning about partially-ordered plans. Following [26, 8] that identifies different styles of computation of definitions and implications in a single logic, it has soon become clear that indeed all processes, including the simple ones, such as reflexes, desires, and behaviour execution, within all three layers of InteRRaP, could be described as special instances of this advanced theory of fluents, time, and action. Hereby, the represented fluents either denote sub-symbolic sensor data, symbolic and abstract knowledge, and even social knowledge about other agents and the agent's own computations on a deeper layer.

Especially for specifying planning, however, an abduction principle for hypothesising about incomplete knowledge, especially about the future and the agent's own options has to be applied [13]. A respective proof calculus for Abductive Logic Programming (ALP) has been developed in [15] and can be used to describe situated processes. On the one hand, this includes an on-line interface to the process shell of COOP that copes with additional knowledge and dynamic changes. On the other hand, this includes the treatment of user-defined plan hierarchies as heuristics and as a means for providing timely, partial solutions in goal-oriented planning. The latter requirement is attacked by extending the EC to the Hierarchical Event Calculus (HEC) [21] that handles explicit abstraction hierarchies of plans. Several issues had to be revisited, such as how to model duration and how macro-operators can be expressed. The respective decomposition of macros covers sequential, concurrent and disjunctive composition. It covers loops and also test actions: These are actions that allow to delay decisions from planning time to plan execution time. Finally, particular strategies are developed that ensure completeness and a reasonable behaviour of the overall proof calculus.

2.4 Implementation

Interestingly, ALP can be mapped onto another, practical form of logic programming, namely Constraint-Logic Programming (CLP) in which basic logical formulae, or constraints, can represent hypothesised information about the world. A unifying perspective to both forms of LP has been given by [36]. This observation makes it possible to translate HEC into an equivalent, executable constraint program, for example, in Oz [34] core code. Adding appropriate constructs to implement the operational strategies and the computational model around (search machine, threads, exception handling, object-orientation) straightforwardly derives the hybrid agent implementation from the specification.

2.5 Bottom Line

In this section, we have demonstrated the power of the proposed methodological attempt by exploring the complete specification space for the practical, hybrid agent design InteRRaP. On an abstract level, this has comprised the operational modularisation of functions in the architectural design and the declarative theory behind the computations within those modules. Using proof calculus and computational model, these considerations are connected to concrete and practical implementation techniques. The particular features that we have highlighted on all these stages could now be compared to those of other approaches from heterogeneous backgrounds.

Some work that we could not mention here due to space constraints reconciles the logic-based, symbolic reasoning with utilitarian reasoning inspired by decision theory. Extending the ideas of [30] to EC based calculi, a decision-theoretic planning approach can be gathered. We will elaborate in a later chapter the importance of such a research for the layered architecture that we propose.

With respect to layering, [17] elaborates the meta-object relationship in InteRRaP and defines the configuration possibilities of the super-layer also to cover computational (time, space) and environmental aspects or *resources* (tools, fuel, workspace) that are to be assigned to the subordinate computations. Indeed, we regard the control as present in current InteRRaP to be a *resource-adaptive* scheme. Changing resources in the environment or in the computation device influence the quality of the actually executed decisions, e.g., if the environment becomes more calm, it is more likely that the deliberative module can timely influence the fast decisions of the reactive module and vice versa. This is however implicitly encoded in the model. A more elaborate notion of reconfiguration shifts the resource-adaptive into an advanced *resource-adapting* scheme where resources are explicitly represent and reasoned about. Finally, *resource-adapted* systems have already compiled these resource decisions implicitly into their computations.

Chapter 3

InteRRaP versus Reactive, Unified Models

In the eighties, a radical shift of paradigm has been initiated by researchers discontent with the latency of the yet pursued deliberative, symbolic intelligence. Coming mainly from a robotic background, they have argued that situated intelligence could not be modelled via abstract, monolithic computations but rather emerges from the interplay of comparably simple mechanisms that do not employ representation and reasoning at all. Hybrid architectures, such as InteRRaP, try to reconcile both complementary perspectives of intelligence. A comparison with the prominent architecture of sub-symbolic robotics, the Subsumption architecture [5], therefore gives much insight into the impact of modular integration. Furthermore, it is a first proof of concept in order to our methodological categorisation to be generally applicable.

As a parallel thread, a more complete methodological extension to logic programming in particular, or symbolic reasoning in general, is provided by unified agent models. These are models that describe the agent as a more or less monolithic, but declarative state and its computation as a single, rational inference. As we will recognise in the following, those models are much more in the tradition of reactive reasoning than of deliberative planning. They have no immediate analog in the hybrid InteRRaP scheme which makes them an interesting subject of comparison and a further milestone for a clean methodological description.

3.1 Layered Competences: The Subsumption Architecture

[5] describes an architecture for situated and embedded robotic applications that sets on the power of emergent, sub-symbolic computations arranged to the well-known Subsumption architecture. Being opposed to the traditional deliberative AI approach, it abolishes the use of any representation and symbolic reasoning

('Let the world be its own model.'). Although InteRRaP insists on the manipulation of symbols, architectural and computational properties have much in common.

3.1.1 Architecture

The principle of layering, or horizontal modularisation, has been indeed first analysed in detail within the Subsumption architecture. [5] proposed this structure as an alternative to the purely vertical organisation found in many system before. In his argumentation, he offers a horizontal separation of functionalities that get more and more complex (they subsume the lower functions) the higher they are stacked. Emergent complexity is gathered by constructing circuits of domain-dependent modules (unidirectional interaction, but feedback loops are allowed) with comparably less power to a static network. Higher layers inhibit or suppress the output of the lower modules and thus frequently influence their communication. Nevertheless, the lower ones stay active and still contribute to the overall behaviour.

Layering has then become a standard technique also in hybrid agent research, although with a slightly different flavour. In the Subsumption sense, layers represent an optional, more capable flow of computation through the agent. Their 'final' decisions have basically the same status and are arbitrated in between. In contrast, a lower layer in InteRRaP, such as the BBL, does indeed implement **all** the functionality of the agent. To be guided towards exhibiting a specific, rational function, it is, however, monitored, reasoned about, and reconfigured by its super-layer (the LPL, e.g., by shutting off an avoid-collision reflex). It is thus not subsumed, but supported by its super-layer. Layers do no more stand in competition, but in a structured, cooperative relation with their super-layers.

Besides the clear horizontal separation into the reactive, deliberative, and social part, InteRRaP furthermore introduces a common vertical structure for every layer. In the Subsumption architecture, both choices are domain-dependent and focus on reactive contents. In a later publication [6], Brooks recognises some reconfiguring, deliberative account, e.g., in path-planning. Still, he abandons any central, absolute representation and complex reasoning.

3.1.2 Computational Model

At the more concrete stage of formal machines, we recognise that the (simple) augmented finite state machines building Brooks' modules can be mapped to the (possibly complex) continuous process transitions within InteRRaP. Since [5] does not explicitly determine a class of functions (linear, polynomial, no side-effects, etc.) to be embedded into the state machine, the reverse direction would not be impossible — although not in the original spirit. Similarly, a concurrent evaluation of the Subsumption agent model is not given in detail, but clearly envisaged.

The communication links that build the static network of Subsumption modules find an analog in the signal-based mechanisms in InteRRaP. InteRRaP, however, does not model loss of signals due to limited buffers. Rather, it extends the possible communication paths as being dynamically encoded into process interaction.

In the light of purely simple, reactive state machines of the Subsumption architecture, a shared memory model is not necessary. This eases the possibility of distribution, i.e., spreading the computations to a cluster of parallel computing devices. Control of communication (inhibition and suppression versus the control process) is available in both computational models; control of computation only in InteRRaP and derived versions of Subsumption.

3.1.3 Theory, Proof Calculus

It may be blaspheming to look for theoretical, thus representational issues in the case of the Subsumption architecture that explicitly denies these sources. Though, it is important for specification purposes that the employed data structures have a particular meaning to the designer of the system. In the case of the InteRRaP BBL, data contains rather analogue symbols, e.g., the fact that infrared-sensor four has reading 235.12. Subsequently, we identify computations on this level as specific inferences, e.g., we interpret sensor fusion as a deductive process including some arithmetics. Therefore, it is justified to compare the granularity of representations rather than to abolish the terminology at all. Accordingly, we identify states in the Subsumption architecture and the BBL of InteRRaP as sub-symbolic data. Symbolic representations as in the LPL and epistemic informations as in the SPL are not present in Brooks-style agents.

3.1.4 Implementation

Because [5] stays vague about the evaluation of his modular network, he puts the burden of reasonable concurrent or even simultaneous interpretation onto the final, pragmatically derived implementations. Its description matches hypothesised implementations with a serialised stepping of state machines or modules. If the embedded, domain-dependent functions are not well-behaving this would result in an unfair, not responsive scheme. Its description does also match existing, fair and asynchronous agent machines on host computers and even hardware-oriented, parallel computations on-board of robots.

The choice of the implementation basis and its domain-dependent instantiation does however have a great impact onto the behaviour of the Subsumption agent: in combination with unsafe communication, there is no prediction possible from the computational model as described in [5]. Therefore, such considerations are already part of the domain-independent computational model of InteRRaP. Any implementation which satisfies it should guarantee responsiveness. Even distributing the shared memory model to parallel processors or between host and robot has become straightforward due to recent developments in transparent distributed computing.

3.1.5 Bottom Line

We have argued that the layered design of InteRRaP shares similarities with Brooks' ideas and incorporates much of its features in a more advanced and deliberative setting. The question of when to call a data structure a representation and when a computation an inference remains a point of discussion. However, Brooks [6] accuses hybrid architectures as simply shifting the horizon of deliberative AI by a limited amount of complexity. We have objected that layering in InteRRaP does not mean to incrementally add subsuming facilities in Brooks' sense, but to introduce another meta-level of control and configuration. This certainly goes beyond a simple shift of horizon.

3.2 Belief, Desire, Intention: dMARS

Rather than to abolish symbolic, logical methods in situated agent design, there have been many attempts to specify some form of unified, *reactive reasoning* as an alternative. Unified hereby relates to the common paradigm of agents being a single, rational inference upon a consistent, logical state. The perhaps most successful agent model that has evolved from this background bases on the Procedural Reasoning System [16]. This model has a mature theoretical background by means of the well-established Belief, Desire, Intention (BDI) tradition [31]. It boils down into a widely applied (work-flow management, combat simulation, air traffic control) implementation basis, dMARS, that has been recently formalised in terms of a Z specification [9]. With respect to the methodology that we have proposed, dMARS is yet the most complete DAI approach to agent design, because the BDI theory uncovers to be a description of the overall behaviour of agents, whereas InteRRaP yet focusses on the decision making processes within.

3.2.1 Architecture

The PRS architecture [16] as a unified enterprise excludes horizontal modularisation. Any reasoning is done on a single object-level, thus control has to be performed implicitly. Nevertheless, the applied, famous vertical modularisation has been adopted by InteRRaP in each layer: the perception of the agent (events) update a knowledge base (belief) in which situation patterns can be recognised. These patterns are closely linked to goal (desire) activation. Decision making or planning transforms these goals into changed intentions or plans that are executed afterwards.

While InteRRaP puts much computational efforts into decision making on its upper two layers, PRS exhibits a tight, simple coupling of plans with goals. This amounts to a unidirectional cycle of computation. While in InteRRaP failure handling, dynamic changes, thus immediate feedback from plan execution to planning and goal activation is possible, PRS simulates this by feeding internal events into the perception of the next cycle. The control inherent in internal events is however merged into object-level decisions and thus also reactively reasoned about.

Although the BDI theory explicitly caters for multi-agent scenarios, we do not find a relevant, dedicated part of the architecture. Similarly to control of computation, it is not straightforward to put social decisions into a single object-level reasoning because negotiations reflect the computation and state within agents. Flexible interaction in a heterogeneous environment thus cannot be easily encoded into flat decisions that mix 'physical' actions with speech acts.

3.2.2 Proof Calculus, Computational Model

[9] provides an operational dMARS interpreter for PRS. Using the Z specification language, a concise definition of the agent's state and its inference cycle is given.

The related process-level of the InteRRaP specification in contrast models the agent as a number of encapsulated, concurrent process cycles. For the monolithic state of dMARS, explicit communication between modules therefore plays a subordinate role and relies on shared memory and the already discussed internal events. Internal events have a similar state as perceived data and are thus not directed.

The inferential aspect of dMARS covers a simple (complexity-bounded), reactive deduction within modules. Bounded complexity turns down the role of concurrency between modules. At least for flexible reorientation in intentions, or plans, competition and concurrency is however important. Although its BDI cycle appears to be serialised, dMARS' open selection functions, e.g., for processing events, activating plans, selecting plans, and selecting branches in plans, are able to express asynchronous concurrency per plan. This is however not an explicit constraint onto the selection functions.

In fact, the structure of decisions in dMARS agents are pretty much precompiled into the plan library that is annotated with goals and activation conditions. This resembles the reflex-based activation and execution of patterns of behaviour in the behaviour-based layer of InteRRaP. If building more rational dMARS agents for complex environments, the designer has to put a lot of effort into analysing the domain and possible situations for gaining a 'working' set of plans.

Deliberative, goal-oriented planning as in the InteRRaP LPL has the ability to flexibly recombine the designers knowledge at run-time. Not every situation thus has to be analysed at design time. Furthermore, abstraction planning appears to be a trade-off between both approaches. It applies as much predesigned decisions as possible, but within a goal-oriented framework.

The stringent decisions of dMARS furthermore diminish the possibilities of treating incomplete knowledge. Although dynamic changes are matched against the maintenance conditions of active plans, the ultimate reasons for setting up the plan are not checked. This leads to a highly non-monotonic behaviour in the light of negative activation conditions. In InteRRaP , the planning process itself is responsible for checking his hypothesised intentions against new or completed data. Non-monotonicity coherently comes through committing to particular hypotheses with respect to the agent's own behaviour.

In dMARS, limited computational control is possible by using the stated annotation features in combination with internal events, such as asserting or retracting facts from the knowledge base.

3.2.3 Theory

The BDI theory [31] is deliberately taken (and tried to be established by formal proof) as the rational basis for dMARS. We have argued that, although it should be seen as a way of describing dMARS agents behaviour from an observer's point of view, any such description can be used to specify its computation in an abstract way. Thus, we expect a restricted, well-defined subset of BDI to be incarnated by dMARS. Indeed, many concepts from the (modal) theory have been grounded in dMARS, such as a situation-based representation scheme with implicit, discrete time.

[37] discusses the theoretical foundations of a plan library or recipes for the agent. The expressivity of such plan includes sequential and concurrent composition, loops, test actions and disjunctive branching. While tests and branches are merged into a conditional construct, the actual dMARS system has no concurrent composition. This has to be emulated by an internal event spawning another plan. Activation conditions and maintenance conditions in dMARS introduce some notion of precondition and effect.

Especially conditionals play an important role for dMARS agents. They allow for run-time decisions otherwise not to encode into the activation conditions of plans without full planning capabilities. By the way, test actions are also reasonable in a more goal-oriented setting, since they cope with a limited, predictable indeterminism in the environment. For treating unlimited, complex environments, a flexible planning approach is still not to be replaced.

We have noticed the gap between the (auto-)epistemic basis of BDI and its realisation in dMARS: knowledge as being present in the social model of InteRRaP agents has to be explicitly reified in programming dMARS agents. This widely excludes the possibility of accessing active goals or plans in reasoning. We suppose this facility, however, as an important basis for any social activity of agents.

3.2.4 Implementation

The C implementation of dMARS against which [9] has been checked is an efficient and widely used agent framework that has been applied, e.g., in sophisticated air traffic control. The straight choice of depth-first selection rules in the implementation favourites persistent decisions but turns down the role of competing intentions. In order to gain flexible reorientation of the agent, this requires a careful design of the plan libraries taking these (implicit) operational decisions in account.

3.2.5 Bottom Line

The dMARS model shows a tight, complete methodological unity including an advanced theoretical background and very efficient and widely used implementation. This makes it a good evaluation partner for hybrid agents, such as InteRRaP. We have made clear that in our opinion, future-oriented planning capabilities and the thereupon based reflection of social reasoning are necessary features of flexible agents that should be traded off against undoubtedly required reactive reasoning. Concurrency as being emphasised by InteRRaP is an important concept to mediate between different, possibly conflicting motivations.

Nevertheless, the unified dMARS model only restricts the full-blown BDI theory, it does not violate against its constraints. This makes it likely to finally bridge the formal gap. In InteRRaP, we have not yet addressed how a theory for the complete agent model looks like, rather what comprises a theory of its embedded computations. In Chapter 6, we discuss the possibility of using BDI also for specifying a layered agent with encapsulated inferences.

3.3 Agent-Oriented Programming: Agent0

Using the intelligent agent paradigm of AI as an extension to object-oriented programming has been impressively motivated by the Agent-Oriented Programming (AOP) framework of [33]. Agentification hereby means the enhancement of traditional applications in order to coordinate their services and inter-operate with humans. Mental states, such as believes, commitments, and capabilities as well as speech acts provide the ingredients for a methodological attempt quite similar to dMARS. It is a restricted form of logic-based reasoning according to an overall, but not yet connected theory that led to the Agent0 language and interpreter applied in collective robotics and information management. It focusses on social aspects of agency and therefore connects to the SPL in InteRRaP.

3.3.1 Architecture

Similar to the dMARS case, the architecture of the Agent0 interpreter is a cycle-oriented, unified model. Incoming perception, especially speech acts, and internal events provoke a change in knowledge which is intimately coupled with some fresh and some retracted commitments to other agents or to the agent itself. Commitments or intentions ate executed according to the system time are and the next cycle is initiated. These steps are elaborated in a serialised, unidirectional manner.

Goals or motivations are not given any status in the architecture. The decision making immediately bases on belief, thus the current situation. By identifying goals as a particular part of the agent's state that steadily influences the decision making over subsequent cycles of computation, dMars or InteRRaP are able to introduce persistence in behaviour. The lack of any such representation is a heavy drawback to Shoham's design and simply prohibits any dynamic adaption of long-term intentions.

Since Agent0 interfaces an embedded application and accesses its state and services, we however identify an additional horizontal modularisation in which the application is treated like a monolithic state and the Agent0 interpreter comprises its social capabilities. This is a simple form of meta-control, because Agent0 has introspection into the application state and reconfigures it by invoking services.

Internal deliberation, however, is seen as a special case of social reasoning, i.e., a decision is a commitment to oneself, and therefore no separate layer as in the InteRRaP model is introduced. This has severe implications: decisions or commitments are bound to speech acts from the outside; Agent0 agents are not pro-active because of changes in their own mental state.

3.3.2 Computational Model

The basic computational building blocks in Agent0 are simple rules and primitive (one-step) commitments. Since there is no conflict resolution of commitment rules and since all active commitments are performed, this is a form of synchronous concurrency: each cycle of the agent synchronises their simultaneous activity.

Dynamic communication between these data structures in Agent0 is done over logical variables (shared memory) and over performing private actions (events) in order to update the mental state.

Events also allow a limited amount of control in Agent0. For example, REFRAINing permits certain actions to be used as commitments in the future. How to reactivate these, is however not clear. A possibility is to encode switching the activity of commitments into the belief structure and their precondition annotations, the so-called capabilities. In any case, REFRAIN as an object-level decision is only to be triggered by social interaction (see before). Additionally, the Agent0 agent cannot retract a choice that it has made without external, social trigger[1]. Control of (internal) communication is not supported in Agent0.

3.3.3 Theory, Proof Calculus

Decision making in Agent0 immediately connects believes with commitments by means of deductive rules. It is thus comparable with the present-oriented activation of plans in dMars or the reflex-based behaviour activation in InteRRaP. The commitments in Shoham's framework are primitive, except a binary conditional. Although they are annotated with a point-based time representation and so-called capability conditions, it is not possible to construct and reason about complex structured decisions.

Based on its (auto-)epistemic representations, a predefined interpretation of speech acts is applied. Due to the lack of goals, this restricts Agent0 especially for the envisaged social settings: Apart from standardisation and portability, non-benevolent settings contain agents that are able to easily deceive the standard Agent0. The argument is similar, if not more important, than in the dMARS case: future-oriented planing capabilities are necessary for the agent to decide at run time about the current situation that the designer cannot completely predetermine in advance.

Furthermore, situatedness plays a minor role in Agent0 agents. Decisions based on incomplete knowledge, especially obsolete ones due to a changed environment, cannot be retracted in Agent0. Also, there is no goal, or effect to be established by the commitment from which we could infer its being obsolete. Combined with limited control facilities, careless domain axiomatisations render Agent0 agents into extremely inconsistent behaviour. While temporary conflicts between encapsulated processes in InteRRaP are a feature, to this extent, it is certainly an undesirable property.

3.3.4 Implementation

Being a very concise language and interpreter, Agent0 is easily implementable from an algorithmic point of view on many platforms, such as Unix systems in Common Lisp and Prolog. The idea is to augment a large spectrum of standard applications with the facilities of mental state and communication. This is more a softbot perspective, whereas InteRRaP certainly addresses both a physical and virtual existence of agents.

[1]Unless we assume that an agent can send UNREQUEST messages to itself !

3.3.5 Bottom Line

Agent0 is undoubtedly a path-setting experiment in the effectiveness of reactive reasoning in social, but benevolent settings. We have shown that competitive, heterogeneous settings as they are more and more developing in the global network question these ad-hoc solutions: persistent behaviour requires long-term goals and control facilities, situatedness requires treatment of incomplete and dynamic knowledge, negotiation requires flexible reasoning mechanisms and complex intentional structures. Nevertheless, having the standard Agent0 interpretations as 'defaults' in the SPL of InteRRaP combines the best of both worlds.

3.4 A Unified Agent Logic: ALP

The previous examples of unified agents looked at their theoretical model from an observer's view. In [25, 26], Kowalski and Sadri extend the perspective of Logic Programming, i.e., the straightforward implementation of declarative problem solving via logical inference engines, towards reactive situated agents. Abduction is taken as the basis for dealing with incomplete knowledge, hypothesised decisions, and dynamic changes to the environment. Their logic furthermore integrates two different styles of computation resembling reactive and more deliberative schemes. [8] instantiates these Abductive Logic Programming (ALP)[2] agents by incorporating Event Calculus (EC) [27] axioms to gather a programmable framework that reasons about fluents, time, and actions and that is exemplary applied in reactive databases and simulated elevator control.

3.4.1 Architecture

[25] proposes the use of meta-programming to describe the role of situatedness for logical agents. Dynamic changes in the environment force the agent to interrupt its reasoning in order to gather and integrate new data. Situatedness also requires early reactions from the agent, i.e., to perform actions and thus commit to certain open choices, before knowing the exact consequences. The construction of a logical proof in order to predict future developments has thus to be interleaved with interfacing the environment. A meta-cycle thus injects new data, commits to certain choices, and resumes the proof procedure for a bounded number of cycles.

Layering on the lower levels of InteRRaP incarnates the deliberative control of reactive object-level reasoning in a subordinate inference engine. In ALP, the deliberative and reactive facilities of the agent are both merged into the object-level. The meta-predicate itself makes simple and fixed decisions. Thus we

[2]In fact, Kowalski and Sadri do not propose any acronym for their model. We have chosen ALP for reasons of simplicity.

distinguish between a resource-adaptive scheme in InteRRaP, while ALP agents follow a resource-adapted scheme: the designer has put the static allocation of resources into the meta-level. Social issues are not mentioned in Kowalski's work.

In ALP agents, we do also find no vertical modularisation: the functional roles within the object-logic cannot be easily assigned to particular logical formulae (goal, constraint, definition).

3.4.2 Theory

[26] resides to first-order logic as the general representational and reasoning device. By discussing the different, operational styles behind equivalence definitions and integrity constraints, they are able to demonstrate deliberative (backwards reasoning) and reactive (forward reasoning) abilities within a single logic. Therefore, this logic is also suitable to comprise a general theory for the encapsulated inferences in a hybrid setting, such as InteRRaP.

On top, [8] sets up an appropriate notion of fluents, time, and actions by incorporating EC [27] into this framework. The flexibility of this event- or plan-centred axiomatisation of strict inertia has also been recognised in our own research [23, 21] and thus been taken over to InteRRaP.

Both [8] and [21] provide a variant of the original EC to deal with macro-actions: time is modelled in intervals and actions decompose into partially-ordered substructures. The logical framework hereby supports sequential, concurrent, disjunctive, and recursive decomposition. With the EC, even test-actions can be formalised. While both approaches annotate actions with preconditions, [8] does not incorporate traditional effect axioms, such as conditional effects as present in [21]. This has implications to the decision making capabilities of the agent.

3.4.3 Proof Calculus, Computational Model

In combination with an abductive proof calculus, such as proposed by [26, 15], the result of [8] is thus not an abstraction planning algorithm as in InteRRaP, but a present-oriented, reactive plan decomposition much like in dMARS.

Abduction hereby solves the second aspect of situatedness: the treatment of incomplete knowledge, especially with respect to the agent's own future decisions. A consistent set of hypotheses about the dynamic state of the world is constructed as a by-product of the (possibly complex) inference. Therefore, the basic mechanisms to adjust these assumption against new information from the environment (injected by the meta-interface) are already present.

ALP agents are unified agents. Their inference is not encapsulated, thus communicates over logical variables (shared memory). Concurrency nevertheless comes in a very fine-grained manner: strategies of an abductive theorem prover need much operational knowledge to mediate between the integration of new information, decision making, and plan execution. For example, such a strategy could emulate the encapsulation, communication, and exception handling in InteRRaP. But this is not part of the ALP specification as in our hybrid case.

With respect to strategic behaviour in decision making, we have investigated preference structures that allow to reasonable abstraction planning (partial solutions, reorientation, complete) with HEC. Davila [8] argues that for his reactive framework, each utility measure can be incorporated into an appropriate goal-ordering that is executed with a depth first strategy. Other considerations are left to the open (fairness required) domain-dependent choice of proof strategies.

We allow the deliberation processes to make decisions about the (de-)activation and parameterisation of subordinated reactive processes. This is a cooperative, structured interaction. Unless the fixed influence of the meta-level, control is not to be found in ALP agents due to rather co-existing, competing deliberation and reactivity.

3.4.4 Implementation

Davila's agents are implemented by specifying the abductive meta-interpreter and object-level rules in depth-first Prolog. Similarly, InteRRaP relies on an implementation base with a logical background. However, since this constraint-based platform is related to abductive reasoning, there is no expensive re-implementation of a meta-interpreter, rather a translation into a constraint program and search engine[3]. Instead, more effort is put into realising the operational parts of the computational model.

3.4.5 Bottom Line

The migration of the ideas from ALP into InteRRaP has been one of the corner stones that inspired our methodology. As such, it brings unified, logical agents and hybrid agents closer together. We have discussed that, to a certain extent, the hybrid setting if InteRRaP makes operational considerations of ALP explicit in the computational model. On the other hand, layering facilities goes beyond a simple object-level integration and co-existence of computational styles. It rather combines with the situatedness aspect of resource-adaptive architectures. The lack of goal-oriented planning in Davila's work has also been recognised by [32]. We agree with his argumentation that the additional annotation of macro-actions with conditional effects allows to integrate the best of both the reactive planning and the backwards planning worlds.

[3]For specific simple processes in InteRRaP, it is even possible to omit any search at all

3.5 Reactive, Unified Agent Models: Bottom Line

A common problem for all reactive agent models that we have discussed in the present chapter is their lack of future-oriented, goal-based planning facilities. In general, goals are an important means to enforce persistence in behaviour. A flexible reorientation of long-term intentions additionally requires the analysis of dependencies in their structure. This is important for complex domains that cannot be fully covered by predesigned plan hierarchies. Especially, social, non-benevolent settings exhibit this property. Therefore, this aspect is either turned down or solved with ad-hoc measures by the presented reactive models.

The unified agent perspective also has to face a perhaps more general problem as it sacrifices many computational aspects for the sake of declarativity and conciseness:

Broad and situated agents are agents that exhibit a variety of cognitive abilities, including navigation, tasking, and social interaction. They are faced with a wide range of goals on various levels of abstraction to be represented and to be mediated in between. In this context, unified agent descriptions tend to overload by relying on a single state and a corresponding, rational computation. It is an overload for the agent's designer or instructor that has the burden of fussily modelling interactions based on the most primitive level of abstraction. Subsequently, it is a particular overload for any (not only reactive) applied reasoning procedure bombarded with a vast amount of heterogeneous information.

Here, Agent0 is one extreme that does not care about these interactions and thus results into floundering behaviour. dMARS provides the other pole by trying to be as persistent as possible, thus to dive into a particular goal. ALP agents, although in principle capable, do not speak out the operational needs to mediate this trade-off. We think that this mediation should be a form of layering (deliberative) facilities in a cooperative, supervising setting.

Chapter 4

InteRRaP versus Deliberative, Cognitive AI

Euphoria of early AI research, such as Shakey [11], had soon been replaced by the frustration of the inherent complexity in the applied symbolic reasoning, especially in planning. In the last chapter, we have presented approaches that for that purpose either abolished representation or focussed on restricted, reactive forms of reasoning. Basic research in understanding the reasons for intractability, formulating alternatives, and especially in controlling expensive computations has also been continued in various, separate fields of AI. Along the separate maturing of these foundations such as in planning and in machine learning, it has soon become evident that a long-term AI programme should envisage their re-integration into possibly complete architectures for intelligence, such as PRODIGY [7].

Adaptive control of computation is also a topic in the complementary enterprise of cognitive science. The different evaluation criteria of agents from software engineering and from cognitive architectures, efficiency versus adequacy, naturally lead to diverging designs. Nevertheless, when it comes to the investigation of cognitive abilities that perform efficiently in a given environment, a common methodology appears. This allows software engineers to take over solutions worked out as adequate models of human cognition as well as it lets cognitive scientists employ the latest technological advances in order to explain their observations. Thus, a comparison of InteRRaP with established cognitive architectures, such as SOAR [29] and ACT-R [1] is useful.

4.1 Planning and Machine Learning: PRODIGY

The PRODIGY [7] architecture has been the first approach to integrate various machine learning techniques with a state-of-the-art planning approach. Although not primarily thought of as a situated agent model, later work on believable agents in virtual environments [3] use the PRODIGY architecture as its deliberative back-bone. Since [3] slightly switches the focus of research, we have decided to rather compare PRODIGY and InteRRaP immediately. Since PRODIGY's various learning facilities have wide-spread roots, our theoretical comparison focusses on the deliberative, planning part.

4.1.1 Architecture

Due to the quite orthogonal research goals of PRODIGY and InteRRaP , the applied modularisation follows a totally different scheme at the first sight. PRODIGY separates the planning component from various learning modules each of which operates on specific aspects (domain axiomatisation, search control, quality control) of the planners data structures and parameterisations. Because these aspects are quite independent, an interaction between the learning modules happens completely over the planner.

Hereby, the interplay of planning and learning can be seen as a meta-object relationship, or even as a layered approach. The learning component monitors the computation and the data structures of the planner and reconfigures them according to meta-inferences. This reconfiguration, however, has a different flavour than in InteRRaP: domain knowledge, action representations, etc. are not subject of influence. Furthermore the control of computation via deliberation is an on-line, resource-adaptive trade-off between efficiency and quality. PRODIGY's learning components do not deliberate (adapt) on-line, but compile their decisions down into the planner which is thus turned into a resource-adapted system.

4.1.2 Theory, Proof Procedure, Computational Model

PRODIGY's algorithmic planning approach bases on a partial-order, hierarchical representation of plans annotated with preconditions and effects. As planning in InteRRaP, a strong criterion for solution plans is applied, i.e., all linearisations of a partially ordered plan have to be solutions in the linear sense. Where PRODIGY insists on fully instantiated operators, HEC can treat partially instantiated, but constrained parameters. Both use a backward chaining technique to connect the ultimate goal to the current situation. Explicit time is only available in InteRRaP while being implicit in PRODIGY's plans.

In addition, PRODIGY's planner keeps a second possibility of reasoning: simulating the execution of the linear head-plan derives a new initial situation and is a form of present-oriented linear planning. This is not immediately representable in (purely plan-based) HEC. It resembles however the ability of executing partial (abstract) solutions in the LPL. Reconfigurations of the BBL in turn influence the the state of the on-line search machine. There is however no backtracking available, if the environment is not reversible.

Dynamic facilities, besides efficient planning, are not the focus of PRODIGY. Thus reactive computations, concurrency, control of communication, signalling and on-line facilities do not play any role. However, incomplete knowledge with respect to the domain and action axiomatisation does. For that purpose, we could describe the inference capabilities of the learning modules as an inductive inference being able to construct rules from monitoring the planner. These are rules that are based on a transparent access to all the planner's data (*glass-box representation*), thus an introspective kind of auto-epistemic knowledge. In InteRRaP, goals and plans of the LPL, for example, are accessible from the SPL.

4.1.3 Implementation

PRODIGY has been implemented in LISP. Its modularisation made it possible to reuse much of the learning and planning techniques in totally different settings, such as in the TOK agent model of [3]. The focus of learning and planning, but, are not on-line, reactive settings.

4.1.4 Bottom Line

The orthogonal aims of PRODIGY and InteRRaP demonstrate a further, yet far away milestone: the integrated, situated agent with on-line learning capabilities. For example, this could include explanation-based learning that enhances InteRRaP for compiling deliberative plans into purely reactive behaviour patterns. This could include learning the effects of actions and the domain concepts while being embedded into a new environment. This could also include the learning of utility of computations in order to optimise resource-allocations in the envisaged resource-adapting InteRRaP. Vice versa, using PRODIGY as a module for implementing reactive agents, in contrast, seems to be the wrong way.

4.2 Heuristic Search: SOAR

The SOAR (State, Operator, Result) [29] architecture is a cognitive architecture that is widely used in cognitive modelling as well as in sophisticated agent applications, such as combat simulations. A reason for its success is certainly the combination of heuristic problem solving with impasse-driven learning that adapts its in-principle intelligence into a reactive and efficient system.

4.2.1 Architecture

A first, horizontal modularisation can be stated in SOAR by separating the interface to the environment from the actual deliberative problem solving. The interface consists of independent perception and motor modules that interact with the problem solving part. In InteRRaP some of these functionalities are to be found in the world interface. It is, however, proposed to move as much as possible of procedural knowledge up to the BBL in order to be controllable by the LPL.

Secondly, the problem solving part of SOAR can again be horizontally structured according to nested problem spaces. Each problem space is elaborated as a search process that is driven by a production system. This rule-based module generates options and preferences for the next decision based on current situation and goal. The gathered options are then resolved in a subsequent decision and execution step.

Whenever resolution is not possible, a new sub-problem space is generated on top of the current one to find a solution to the conflict. This solution is afterwards incorporated as a learned data structure into the memory. Thus we could speak of additional layers in the Subsumption sense: the functionality of the higher problem space subsumes the lower problem space; the higher layer is called as a service and its decisions are afterwards integrated, much as in the original, activation-commitment design of InteRRaP. Because the impasse-driven problem solution is automatically compiled into chunks, or learned facts, we also find a control aspect in SOAR. Because the subordinate problem is however not meat-reasoned about, we regard this as a resource-adaptive form of control. Since SOAR is not a multi-agent framework, social reasoning is not addressed.

4.2.2 Computational Model

Perception, action and problem solving in SOAR run concurrently and communicate over shared memory. Although SOAR applies a monolithic cycle, there is nevertheless a concurrency between competing production rules in the generation of options. There is no traditional conflict resolution of active rules. Nevertheless, their concurrency is synchronous being synchronised by subsequent decision cycles. Communication between rules happens over instantiation of variables, thus a shared memory.

The production system module as a part of the decision cycle is already a (not-bounded) complex computation that is nevertheless implemented by simple rules. This is due to decisions waiting until this deductive process is stable. Serialisation of computation is also to be found in nesting problem spaces: until the nested space has not solved its problem, its ancestor is halted.

Control of computation is encoded into explicit control rules (heuristics, preferences) that are compiled from the nested search spaces into the object-level knowledge.

4.2.3 Theory, Proof Calculus

The deliberative capabilities of SOAR are derived from present-based planning approaches of traditional AI. Thus it relies on a situational, symbolic representation of facts. Primitive actions are encoded with preconditions and effects to instantaneously change a situation. Time is not explicitly represented.

However, it is not a blind situation-based search that builds the decision making of SOAR. The selection of possible actions is driven by the deductive production rule process which constrains the number of possibilities to explore. Although the present-oriented scheme provides some reactive account, this is not speculatively used to trigger external actions as by the interplay of LPL and BBL of InteRRaP. Only the primary problem space, while active, is able to perform uniquely determined interactions with perception and motor modules. Thus on-line reactivity in SOAR comes completely out of learned knowledge. Incomplete knowledge is not a topic.

4.2.4 Implementation

SOAR has been implemented on various platforms, from which the most recent C version has gained a lot attention due to its performance. The applied, sophisticated algorithmics to, e.g., implement the pattern matching in production rules, managed to handle huge rule databases. The simulation of combat pilots using SOAR also supports the claim that this is one of the most efficiently implemented platforms in agent design.

4.2.5 Bottom Line

The SOAR approach to adaptive deliberative intelligence is close to the PRODIGY claim: control knowledge and heuristics render a complex problem solving tractable for situated intelligence. While the specific form of learning in SOAR is certainly of interest on symbolic levels, such as the SPL and LPL in InteRRaP, we doubt whether this approach will also succeed in the fine-grained control of robots. This is because rapid learning has to be supported by an appropriate initial knowledge provided by the agent designer. Without the right abstractions, such as sub-symbolic patterns of behaviour, complex plan structures, and social representations this is not easy to encode into a single object-level system, especially into rules that mix action proposals with preferences.

4.3 The Adaptive Control of Thought: ACT-R

ACT-R (Adaptive Control of Thought with Rational Analysis) [1] started out as an integrating framework of a model of human memory (the Human Associative Memory HAM Theory) and a production system as a procedural, cognitive problem solving machinery. Those theories have then been complemented by the rational analysis approach that adapts the computation of memory and production system to a specific setting. Beneath a cognitive modelling tool, i.e., a basis for making predictions on specific axiomatisations of agents, it is used as a user modelling tool, e.g., for predicting failures in tutoring students.

4.3.1 Architecture

The ACT architecture can be separated into three building blocks. There is the associative memory model containing and activating the declarative knowledge of the agent. There follows (vertically) the production system computing upon the procedural knowledge of the agent, constructing goals, accessing the memory, and making decisions. And on top of both, the rational analysis level uses parameterisations of the underlying representations in order to influence their computational behaviour.

Memory and production system build up a single layer where from the perceptually updated (activated) belief of the agent, goals are activated on a stack in order to trigger production activity. This activity results in producing sub-goals, changing the internal status, and performing external actions. This computation is however reflected and controlled by a (learning) meta-level that (on-line) assigns activities, rule weights, match times, and finally computation time to the different underlying parts. Because rational analysis is in a steady flow, we obtain a resource-adaptive system.

4.3.2 Computational Model

ACT-R is described as a complete cycle of the architecture, ranging from propagation of activity in the memory, matching of production rules, selection of a production rule, and execution of the action-part of the rule. The possibly complex processes of InteRRaP could thus be compared to the independent, simple production rules that are, in a first step, instantiated concurrently from each other. Shared memory is thus an important form of communication, while the action part of the rules could also contain internal events, such as subgoal activation, that trigger computations afterwards. The following conflict resolution, however, synchronises and chooses exactly one instantiated rule to influence the state of the system.

An outstanding property of ACT-R is this resolution being dependent on the matching time that is simulated from the parameters of the already done instantiations. Thus by changing parameters, a direct control of computation is possible. Nevertheless, this control happens *posthum* after instantiations have been made. InteRRaP and its upcoming resource-adaptive descendants use the influence from the upper layer to prune the amount of computation in advance based on predictions.

4.3.3 Theory, Proof Procedure

Frame-based knowledge structures and production rules comprise such as those in the ACT-R model a special form of deductive, situation-based reasoning. While the knowledge structures in ACT omit any time or epistemic representation, the production rules implicitly encode goal-oriented action selection by annotating goals with actions and subgoal structures. Complex actions are not provided in ACT. Nevertheless, primitive actions are executed while building up the goal stack. This contains some deliberative elements, but results in an overall myopic decision quite similar to behaviour patterns in InteRRaP's BBL. Incomplete knowledge and dynamic, on-line facilities are not present in ACT.

Analogy-driven learning, a special method to derive new object-level rules, and the decision-theoretic representations and inferences in the rational analysis are not immediately comparable to InteRRaP. Rational analysis turns out to have a reasonably backed foundation in terms of utility and probability. We have already discussed that there is some work on integrating InteRRaP's long-term, symbolic reasoning with utilitarian. Unless a resource-adapting scheme is developed, the 'predictions' used in the LPL and the SPL however base on absolute, fixed knowledge, not adjustable likeliness.

4.3.4 Implementation

The ACT-R implementation is available in Lisp, thus on a wide-spread platform. For specific purposes, Lisp functions can be integrated to build an appropriate interface to an application. In contrast to SOAR, reactive applications are not envisaged.

4.3.5 Bottom Line

Although the currently research resource-adapting focus of InteRRaP fits quite nicely with the adaptive control of thought paradigm, InteRRaP extends the two-level picture of simple meta-reasoning and complex object reasoning into its three-layered social-local-reactive distinction. We have to learn from ACT-R that control of computation is strictly coupled with treatment of uncertainty.

4.4 Deliberative, Cognitive AI: Bottom Line

The common centre of interest of all these presented deliberative or cognitive architectures is the adaptivity of the agent to its environment and its computational device. The claim of InteRRaP is here that dynamic environments require a steady adaption, thus explicit on-line representation and reasoning in the form of planning and social reflection. The distinction into object-level (external) decision making and meta-level off-line precompilation of computational control is thus not made in favour of the three-layered on-line scheme. It will be one of the future milestones to integrate both perspectives into a single architecture; the ACT-R rational analysis seems to be the most promising representation to be reconciled with symbolic reasoning for that purpose.

Chapter 5

InteRRaP versus other Hybrid Models

As a mainly pragmatic attempt to reconcile the separate advances made in sub-symbolic, reactive systems, such the Subsumption architecture [5], and in deliberative planning systems, such as PRODIGY [7], hybrid architectures employ the principle of integration by modularisation. Similar to InteRRaP, the 3T [4] architecture, for example, builds on a three-layered scheme of reactive skills, task networks, and planning to provide robots with high-level, goal-oriented behaviour. We have also discussed that recent efforts in formalising hybrid agents, such as our computational model for InteRRaP, allow a deeper understanding of their respective foundations and possibilities. Thus, the general, compositional agent specification toolkit DESIRE [10] also has seemingly a close relation to our framework.

5.1 Deliberation and Reactivity: 3T

Layered, hybrid architectures have been especially a product of advanced robotics research. Although sub-symbolic methods [5] provide a robust, low-level control of autonomous, situated agents, it has been soon recognised that they are unable to express higher-level, goal-oriented behaviour beyond navigation and manipulation. On the other hand, purely deliberative planning approaches, such as PRODIGY [7], are not suitable to perform on-line tasks in dynamic, unpredictable environments. The three-layered (or rather tiered) 3T architecture [4] introduces a common frame for reactive skills, task networks, and planning that has been especially designed for various multi-robot platforms. We will see, how this background imposes some differences to InteRRaP.

5.1.1 Architecture

As its name suggests, 3T is also a three-levelled architecture. However, the three horizontal levels of deliberative planning (the Adversarial Planner [12]), of sequencing (Reactive Action Packages [14]), and of reactive skills [39] do not fit exactly with the InteRRaP categories. The deliberative 3T functionality of adversarial planning can be found on the InteRRaP SPL and LPL. The roles of RAP's and skills are however distributed to both the deliberative and reactive layers in InteRRaP.

This is due to 3T's separation into layers being purely based on time, task, bandwidth and modifiability scales. For example, the reactive skills normally operate immediately on-board of robots while sequencing is located remotely on a host computer. Therefore, 3T does not introduce a coherent horizontal interaction: RAP's are indeed a reconfiguring force by controlling activation and deactivation of the reactive skills. On the other hand, planning and sequencing layer follow a subsumption scheme in which the decisions of the planner simply decompose into RAP's.

In InteRRaP, both planning and the decomposition (execution) of plans is located in the LPL. The basic operators herein control the computation and communication of the BBL in which patterns of behaviour are both procedural knowledge as a kind of low-level RAP's and reactive skills. This design decision flexibilises goal-oriented influence that is herein to reason about sequencing facilities. This decision also flexibilises reactive modelling as being more expressive and controllable by the deliberative facilities. We have already discussed in Section 3.1 that although InteRRaP is a coherent, shared-memory model, there are consistent scheme for distributing its processing over several computing devices.

Particular features of the AP planner cater for multi-agent settings, such as synchronisation of joint plans and reasoning about how to prevent other agents' interference. In the light of a concise theory of planning, these features are indeed higher-level features merged in a fixed manner into the deliberative object-level. In a general setting, a complete social level to mediate and talk about the local deliberation is necessary.

Turning to the vertical modularisation, we find that the lower two tiers of 3T do not distinguish between functions such as knowledge base, goal activation, decision making, and execution. Only the planning level switches between a planning and execution mode.

5.1.2 Computational Model

3T integrates three, rather separately developed systems (AP, RAP's, and reactive skills) thus three different, though related computational models for each of the tiers. The analog to the encapsulated processes in InteRRaP are the planner, each RAP, and each reactive skill. Communication is done by exchanging data structures within each tier and signalling between the tiers. Between arbitrary InteRRaP processes, both types of communication are available according to the different needs of communication (input-output versus exceptional situation).

Accordingly, concurrency varies with tiers. The skill tier is customisable to exhibit either synchronous skill functions (steadily triggered one-shot functions) or asynchronous skill processes (continuously running processes). The communication structure is, similar to the Subsumption architecture, fixed, and control is devoted purely to (de-)activation of skills.

On the independent RAP tier, we find RAP's to encode concurrent activity in a similar, dynamic manner than the InteRRaP processes do. They are simply invoked by the third, independent layer in its execution mode that is mutually switched with the planning mode.

5.1.3 Theory, Proof Calculus

Adversarial planning in 3T and the InteRRaP HEC-based planner share a hierarchical, non-linear planning approach. This planning operates on precondition and effect representations of the underlying reactive facilities to form long-term decisions. In contrast to the concurrent planning and execution scheme of InteRRaP, AP switches between both modes, thus planning is a rather off-line procedure under complete and static knowledge. AP introduces however some features that are typically interesting in multi-agent domains.

The synchronised execution of actions is a construct that implicitly covers negotiation and coordination actions. In InteRRaP, these actions are explicitly inserted and reasoned about by the SPL as being highly dependent on the social setting.

AP also features the so-called counterplanning mode to plan for actions that support the integrity of persistence links and the additional annotation of actions with during conditions. We do also argue here that this is certainly a higher-order feature that requires reasoning about the top-level plan on the SPL. Furthermore, protocol reasoning and execution is not easily to mix with object-level decision making.

5.1.4 Implementation

3T is intended to run on various robotic platforms. The two upper tiers are implemented in LISP and run on a host computer whereas the skill manager and its reactive skills are to be ported to the different on-board facilities of robots.

InteRRaP focusses on an integrated, transformable implementation from the formal specification and heavily relies on a homogeneous, logical implementation base completely running on a host. Experiments with InteRRaP have shown, that its reactive core exhibits a latency of ca. 100 ms and is able to implement navigation and tasking facilities in an automated loading dock scenario with Khepera robots.

Bonasso and Kortenkamp report on their applications requiring a time scale in the millisecond range in order to guarantee functioning. Furthermore, certain applications like robot vision require a high bandwidth of data to be handled by the skills. This would be certainly not transmittable to a host computer.

An alternative for InteRRaP is to set on the power of distributed computing in order to export certain reactive features onto the on-board computer without loosing a shared-memory model. Tests with extending the role of the world-interface accordingly have proven successful, but a clean, layered perspective requires deliberative access to these functionalities.

5.1.5 Bottom Line

Different applications enforce different design decisions and notions, especially in layered architectures. We nevertheless think that the meta-object separation of deliberative and reactive facilities is a new, and coherent extension to hybrid agent design. For demanding robotic applications, we should however investigate possibilities of distributed computing in order to make the clean specification able to cope with, e.g., robot vision. 3T's adversarial planning incorporates yet unique multi-agent features that we propose to be better described by a separate meta-deliberative, or social layer.

5.2 An Agent Toolkit: DESIRE

The last station of our comparative tour through agent design does not investigate a specific model, but the generic agent construction toolkit, DESIRE [10]. DESIRE has been developed out of a compositional software engineering perspective, thus shares a similar methodological background as we have proposed for InteRRaP. Building blocks of a DESIRE computational model are logical components or modules to be concatenated via semantic links and to be grouped to higher-level structures. This process is supported by a graphical user interface and an automated animation facility. A straightforward way of comparing both approaches is thus to map InteRRaP onto the DESIRE concepts which uncovers subtle incompatibilities.

5.2.1 Architecture

Being a generic framework for compositional agents, DESIRE does not come with a specific architecture, but provides basic horizontal and vertical modularisation facilities. Vertically, semantic links connect the state of object-level components in a directed manner. These semantic links consist of transmitting information, requesting information, or setting up computations. They are also available in a horizontal form transporting meta-data, such as information about particular properties of a component's state or reconfigurations to a subordinate computation.

Thus the InteRRaP architecture, i.e, the external interface of the agent, the inner-layer processing (processes=modules), the composition of processes to a layer (layer=higher-level module), and the meta-object relationship between layers is expressible by component structures and specific forms of semantic links.

DESIRE distinguishes 9 different types of semantic links with particular impact onto the receiving module. This possibility is however not closely linked to the deductive, monotonic inference in components: strategic and operational ingredients are not subject of change, only the state of the component (changing knowledge or goals). An advanced, resource-adapting perspective such as InteRRaP proposes is thus not straightforward to model.

5.2.2 Computational Model

Similar to InteRRaP, the basic inference processes, thus the primitive components of DESIRE, encapsulate logical inference engines. Their dynamic creation and interplay, however, poses a problem for DESIRE that assumes a fixed set of components and semantic links during the lifetime of the agent. Therefore all possible InteRRaP process instances and their interactions would have to be specified in advance. (De-)activation would have to be encoded into the state of components and the type of semantic links; this turns out to be a complicated enterprise.

Similarly, semantical links are the only medium of communication to explicitly transmit snapshots of the logical state of one component into the state of another component. They are thus signals. A complete update of component states based on these signals is a (possibly) concurrent, but synchronised step in the transition of the agent. This update is followed by a (possibly) concurrent, synchronous inference within each component.

InteRRaP's communication facilities go beyond that scheme. Signals are asynchronously concurrent also to the independent computation of processes. Furthermore, immediate and implicit information exchange comes through the shared memory model and the shared logical data structures applied. This form of communication is the standard input-output scheme for computations. InteRRaP signals rather indicate exceptional changes in state and are transmitted in a controlled (via the layer) manner.

To get this mechanism in concordance with DESIRE, we could introduce the signal queues of each InteRRaP process as additional components. Once shared logical data is available, two semantic links could then transmit mutual computation results. What seems to be not possible, however, is that semantic links are added on the fly according to the dynamic process communication structure in InteRRaP. Again, specifying the space of all possible semantic links and controlling their (de-)activation seems to be not feasible.

5.2.3 Theory, Proof Calculus

The overall theory of a compositional specification in DESIRE is rather a descriptive, observer's point of view. The monotonic inferences within the agent, however, each employ their own theoretical framework to be encapsulated. Semantic links therefore have to 'translate' between syntactical units and enforce non-monotonic behaviour. Although this allows a general, on-line agent skeleton and multi-agent structures with (auto-)epistemic representations (due to meta-links), it is the non-trivial task of the concrete specification to insert a reasonable treatment of incomplete and dynamically changing knowledge.

A coherent theory across components that is based on a more flexible inference principle, such as the combination of HEC and abduction in InteRRaP, would flexibilise the expressivity of DESIRE. Much of the designer's work with respect to coping with non-monotonic influence could be consistently dealt with in the semantic links. Furthermore, such notions as fluents, time, and action are central to any reasoning within the agent. As argued before, these considerations are important for any situated agent model and domain, thus should be part of the domain-independent part, here of the agent toolkit.

5.2.4 Implementation

DESIRE supports the implementation of agents by animating the compositional skeletons from the formal specification. This animation is rather thought of as testing the specified models before implementing them manually. In InteRRaP, we have also defined such a transition for the hybrid process shell. In addition, also the inner-process inferences are mapped onto a correct and practical implementation.

5.2.5 Bottom Line

At the beginning, we were quite confident to discover a strong relationship between the generic DESIRE model and our concrete InteRRaP model since relying on a quite similar methodology. We have however shown that due to subtle incompatibilities, the most feasible possibility for modelling InteRRaP agents with DESIRE is to specify large parts of the layered agent in form of single, monolithic components, such as the complete inner-layer processing. On the one hand, DESIRE introduces an interesting categorisation of interactions between modules in the form of semantics links. This could be used to complete the computational model of InteRRaP with respect to different modes of control. On the other hand, we propose the use of a single, coherent theory and an underlying, situated inference principle within components as a highly important extension of DESIRE in order to be more flexible and useful.

5.3 Hybrid Models: Bottom Line

It has been surprising to see that InteRRaP uses quite different concepts to other, more pragmatically oriented hybrid models, such as 3T and DESIRE. While particular aspects of those approaches (counterplanning, categorisation of interactions) promise to be fruitful extensions to InteRRaP, we are convinced in our approach being one of the most matured, hybrid models available. A coherent way of describing layered reasoning, a coherent theoretical background based on situated inference, and a formal specification strongly connected to implementation platforms are the features that we suppose are the benefits of a clear methodology and its instantiation to a specific agent model.

Chapter 6

Conclusion

The present report proposes a full-range methodological approach for specifying (hybrid) agents. The role of computational models has been emphasised as being the missing link between abstract, high-level conceptualisations and concrete implementations, between declarative, theoretical considerations and the operational, architectural ones.

We have demonstrated the power of the proposed methodological attempt by exploring the complete specification space for the practical, hybrid agent design InteRRaP. On the abstract level, we were able to describe modularisation and interaction issues in the context of layering. We did also address representational and inferential issues with respect to the inherent decision making. Using a situated, abductive proof calculus and a computational model highlights concurrency and communication between (sub-)cognitive processes, these considerations are connected to concrete and practical implementation techniques, here a logical host platform.

The particular specification properties attributed to all these stages are identified by the dimensions of abstraction and declarativity. They could also be investigated across models from heterogeneous backgrounds, with different aims, and also different applications. Figure 6 lists the variety of models that we have taken to compare with InteRRaP. A particular result of this investigation is that the dedicated specification properties appear to be rather independent within a particular stage, i.e., such features are to be found across the models. But as we have also seen in this report, there are correlations between the application of particular designs on connected specification stages, such as the coincidence of high modularisation and concurrency. This identifies a collection of associated model features which build a methodological core for future agent design.

In the remainder of the concluding chapter, we would like to comprise the investigated properties across the different models. Hereby, we especially focus on the perspective that InteRRaP takes on the particular specification stages and how the future development of InteRRaP could gain from overtaking results from the compared models.

Model	*Design Goals*	*Background*	*Applications*
InteRRaP (Müller et al, 96)	Integration of Reactivity and (Social) Deliberation	DAI	Robotics, Transport Telematics
Subsumption (Brooks, 86)	Robust and Reactive Robot Control	Robotics	Robotics
dMARS (Georgeff et al, 87)	Reactive, Rational Planning	DAI	i.a., Workflow Management, Combat Simulation, Air Traffic Control
Agent0 (Shoham, 91)	Agent-Oriented Programming	AI	Robotics, Information Management
ALP (Kowalski et al, 96)	Unifying Reactive and Deliberative Inferences	AI	Reactive Databases, Elevator Control
PRODIGY (Carbonell et al, 91)	Integrating Planning and Machine Learning	AI	Synthetic Characters
SOAR (Newell, 91)	Cognitive Architecture with Heuristic Search	Cognitive Science	i.a., Cognitive Modelling, Air Combat Simulation
ACT-R (Anderson, 93)	Resource-Adapting Cognitive Architecture	Cognitive Science	i.a., Cognitive Modelling, Tutoring
3T (Bonasso et al, 95)	Intergration of Reactivity and Deliberation	Robotics	Robotics
DESIRE (Keplicz et al, 94)	Agent Specification Toolkit	Computer Science	-

Figure 6.1: Goals and Backgrounds of the Compared Agent Models

Architecture	*Horizontal Interaction*	*Horizontal Modularisation*	*Vertical Modularisation*	*Vertical Interaction*
InteRRaP	Resource-Adaptive Meta-Control	Reactivity, Deliberation Social Reasoning	K, G, D, E K, G, D, E K, G, D, E	Bidirectional Bidirectional Bidirectional
Subsumption	Subsumption	Domain-Dep. (Reactivity)	Domain-Dep.	Unidirectional (Feedback)
dMARS	No	Reactive Reasoning	K, GD, E	Unidirectional
Agent0	Simple Meta-Control	Application Social Reasoning	No K, D, E	No Unidirectional
ALP	Resource-Adapted Meta-Control	Inference Interface	No No	No No
PRODIGY	Resource-Adapting Meta-Control	Planning Learning Modules	No Domain, Search, Quality	No No
SOAR	Subsumption (Resource-Adaptive Meta-Control)	Interface Problem Spaces	Perception, Motor KG, DE	No Unidirectional
ACT-R	Resource-Adapting Meta-Control	Production System Rational Analysis	K, GDE No	Unidirectional No
3T	Simple Meta Control Subsumption	Reactive Skills Sequencer Planning	No No KGD, E	Unidirectional (Feedback) Bidirectional Bidirectional
DESIRE	Simple Meta-Control	Model-Dep.	Model-Dep.	Unidirectional (Feedback)

K=Knowledge Base, G=Goal Activation, D=Decision Making, E=Execution

Figure 6.2: Comparison of Architecture

The architectural overview is to be found in Figure 6. As aforementioned, we have categorised the models according to horizontal modularisation and interaction, and for each horizontal module, its vertical structure. Hereby, the letters K, G, D, E stand for the functional roles of Knowledge, Goal Activation, Decision Making, and (Plan) Execution. A module called KG, for example, integrates both functions at once. The direction of interaction indicates whether a vertical structure implements a monolithic cycle with unidirectional control flow, whether feedback is allowed, or whether it exhibits a flexible bidirectional scheme.

Following the argumentation of [6], horizontal modularisation is a technique to enable agent designs for dealing with multiple, heterogeneous goals on various levels of abstraction. Unified, monolithic models, in contrast, have problems to flexibly mediate between these goals the more restrictive the agent is in its vertical computation cycle. Thus modularisation is a means to make the necessary, operational considerations explicit.

We have distinguished horizontal modularisation into subsuming forms, such as in the Subsumption architecture, and into meta-forms, such as in InteRRaP. Subsumption regards a layer as an optional computation path through the agent. Herein, 'final' decisions of layers have basically the same status and are arbitrated in between. In contrast, a lower layer in InteRRaP, such as the BBL, does indeed implement **all** the functionality of the agent. To be guided towards exhibiting a specific, rational function, it is, however, monitored, reasoned about, and reconfigured by its super-layer (the LPL, e.g., by shutting off an avoid-collision reflex). It is thus not subsumed, but supported by its super-layer. Layers do no more stand in competition, but in a structured, cooperative relation with their super-layers. The combination of this notion as a resource-adaptive influence and the relationship of social and local reasoning makes InteRRaP unique.

The lesson to learn from deliberative, or cognitive architectures as presented in Section 4 is to envisage a more elaborate form of resource-adapting intelligence in InteRRaP. Learning representations and control rules is one aspect. Currently we focus on on-line, utilitarian guide of computation to be integrated into our paradigm of layering as control.

Turning to the respective computational models (Figure 6) it is to note that besides InteRRaP, dMARS, ALP, and DESIRE, such a model is not formally defined. Concurrency is available in most models (Exception: PRODIGY) but in different forms. Especially the cycle-oriented unified models dMARS and Agent0 focus this aspect around selecting active plans out of simple decision procedures.

Complex planning, however, needs a more asynchronous decoupling also from other computations to retain responsiveness. We have argued that encapsulation and concurrency are reasonable tools to allow for dynamical reorientation in computation. On the other hand, sequential or depth-first strategies are of course a more persistent, but non-responsive form of scheduling activities. Agent0 and dMARS show that both extremes are a burden for the agent programmer to set up reasonable decisions. A controlled form of concurrency as in InteRRaP

Computational Model	*Concurrency*	*Encapsulation*	*Communication*	*Control*	*Basic Complexity*
InteRRaP	Asynchronous, Within Modules	Within Modules and Modules	Asyn, Safe, Dyn, Signals and Shared Memory	Communication and Computation	Simple and Complex
Subsumption	Implement.-Dep. Per Module	Modules	Asyn, Unsafe, Stat Signals	Communication	Simple
dMARS	Asynchronous Per Plan	No	Syn, Safe, Dyn Events and Shared Memory	(Computation)	Simple
Agent0	Synchronous Per Rule and Commitment	No	Syn, Safe, Dyn Events and Shared Memory	(Computation)	Simple
ALP	Asynchronous Within Modules	Modules	Shared Memory, Dyn	Computation	Simple and Complex
PRODIGY	No	Modules	Shared Memory, Dyn	Computation	Complex
SOAR	Async (Interf.) Synchronous, Per Production	Perc. , Motor Problem Spaces	Shared Memory, Dyn	Computation	Simple and Complex
ACT-R	Synchronous Per Production	No	Syn, Safe, Dyn Events and Shared Memory	Computation	Simple
3T	Asynchronous Within Modules	Modules	Asyn, Safe, Stat Signals and Shared Memory	Computation	Simple and Complex
DESIRE	Model-Dep. Per Module	Modules	Safe, Stat Signals	Computation	Model-Dep.

Figure 6.3: Comparison of Computational Model

should be the conclusion to draw from this discussion.

The application of directed signals or undirected events relates to the vertical interactions as described by the architectural comparison. Signals are reasonable for bidirectional feedback whereas events (a kind of inner perception) are applied in unified, single-level models to influence the belief and goal structure from within the plan execution. We have made clear that there are different needs for communication and that accordingly, different means of information transmission in the form of explicit signals (exceptional situations) and shared memory (input-output) is reasonable.

As already discussed in Section 5.2, the expected similarities between DESIRE and InteRRaP turned into subtle incompatibilities, such as the control of communication that is only to be found in the Subsumption architecture and InteRRaP. Similarly, control of computation is treated in most single-level models by using internal events. On the other hand, DESIRE introduces a clean categorisation of module or process interactions, a feature which could also enhance the InteRRaP computational model.

In Figure 6, we have comprised the theoretical background of the agent models. We have focussed on the deliberative capabilities if they rely on different theories, such as the decision-theoretic rational analysis of ACT-R on top of the production system. Brook's agent model does of course not fit at all into this scheme because denying any such aspect.

Coinciding with their application in multi-agent settings, architectures with a (possible) social module do also apply an (auto-)epistemic level of representing situations. Time, in contrast, is always an important notion for any agent whether implicitly encoded into relations of situations and actions or whether explicitly present in the form of intervals or time-points. Explicit representations of time are subject of reasoning. If we imagine domains in which scheduling decisions is an important facility for the agent, explicit representations turn out to be more flexible.

It is also interesting to find that reactive reasoning does not restrict to situation-based representations. The ALP agents of Section 3.4 are an example for this claim. Nevertheless, among the other models, reactive reasoning coincides with situation-based and deliberative planning with plan-based representations.

For doing reactive reasoning it is especially relevant to build up abstract macro-decisions which decompose into more primitive actions. Conditionals hereby play an important role since putting certain decision making features into the execution module. Both macro representations and conditionals are however also important for deliberative planning approaches, such as InteRRaP and PRODIGY, since allowing for more reactive behaviour (abstract connection from goal to current situation) and the treatment bounded indeterminism.

For the present report, we have regarded descriptive and specifying theories to be equivalent. Indeed, the theoretical framework that we yet have established for InteRRaP turns out to cover just inner-process inferences. In the light of a

Theory	*Facts, Fluents*	*Time*	*Action Annotation*	*Macro Actions*	*Reasoning*
InteRRaP	Subsymbolic Symbolic (Auto-)Epistemic	Explicit Interval-Based	Preconditions cond. Effects	S, Co, D, T, L	Plan-Based
Subsumption	Subsymbolic	No	No	No	No
dMARS	Symbolic	Implicit Point-Based	Preconditions (Effects)	S, C, L	Situation-Based
Agent0	Symbolic (Auto-)Epistemic	Explicit Point-Based	Preconditions	C	Situation-Based
ALP	Symbolic	Explicit Interval-Based	Preconditions	S, Co, D, T, L	Plan-Based
PRODIGY	Symbolic Auto-epistemic	Implicit Interval-Based	Preconditions cond. Effects	S, Co, D, L	Plan- and Situation-Based
SOAR	Symbolic	Implicit Point-Based	Preconditions Effects	No	Situation-Based
ACT-R	Symbolic	No	(Effects)	No	Situation-Based
3T	Subsymbolic Symbolic	Explicit Interval-Based	Preconditions During Conditions Effects	S, Co, D, T, L, Sy	Plan-Based and Situation-Based
DESIRE	Symbolic (Auto-)epistemic	Model-Dep.	Model-Dep.	Model-Dep.	Model-Dep.

S=Sequential Composition, Co=Concurrent Comp., D=Disjunctive Comp., Sy=Synchronous Comp., T=Test Actions, L=Loop, C=Conditional

Figure 6.4: Comparison of Theory

theory of hybrid agents, we however need some more advanced framework that also covers modularisation and encapsulation. Where this is already available for single-module, unified models such as dMARS following the BDI [31] theory, we currently try to set up a Hybrid BDI model [19] as a structured conglomerate of traditionally rational layer agents.

The properties of the Proof Calculus (Figure 6) are strongly coupled with the theoretical considerations: where a complex reasoning about plans is envisaged, the inference runs a future-oriented planning procedure.

In the case of PRODIGY, the planner furthermore is able to complement this with a present-oriented 'simulated execution'. In INTERRAP, a similar, but situated effect is introduced by abstract operators solutions being already executed and thus triggering changes in the reactive layer.

Pure present-oriented plan decomposition puts the burden of a reasonable plan library to the agent programmer. The analysis of a complex domain, especially for social settings, is not straightforward. Thus a mixed strategy between using as much designer knowledge in the form of abstract operators as possible and exhibiting as much planning, goal-oriented facilities as necessary favourites future-oriented abstraction planning approaches as in INTERRAP. Resolving dependencies between possibly conflicting intentions is furthermore an important feature of persistent, but flexible agent behaviour.

We see that abduction is especially useful for describing any situated reasoning faced with incomplete knowledge about facts and future decisions. Deductive frameworks have to treat this in a non-monotonic way which often implies non-retractable decisions. This is especially true for the generic toolkit DESIRE. Since inner-module theory and proof calculus are not part of the model-independent framework, the use of an appropriate inference principle is necessary.

A special status can be assigned to the learning facilities of PRODIGY: they can be partially attributed to inductive reasoning on top of the domain and plan representation, thus they deal with incomplete domain and action axiomatisation. This is a yet unique feature.

Finally we turn to the relevant implementation issues which we have identified as the landscape of available language platforms, the relation to the higher-level specifications, and the realisation of concurrency. We see that in particular, reactive implementations for robotics are done in C and ported to or even realised by robot hardware.

For the applicability of INTERRAP to more advanced robotic settings than the automated loading dock, we have seen that distributing computation simulateneously on several computing devices is a necessary enterprise. The latest developments in distributed logical programming provide the transparent tool to nevertheless keep the shared-memory model. More operational considerations, but, have to be made in order to minimise network overhead by that design decision.

We do also recognise that some implementations heavily restrict the expres-

Proof Calculus	*Inference*	*Incomplete Knowledge*	*On-line Facilities*	*Decision/ Planning*
InteRRaP	Abduction	Action Occurence	Yes	Future-oriented Abstraction Planning
Subsumption	-	-	-	-
dMARS	Deduction	No	Yes	Present-Oriented Plan Decomposition
Agent0	Deduction	No	Yes (no retract)	Present-Oriented Action Selection
ALP	Abduction	Action Occurence	Yes	Present-Oriented Plan Decomposition
PRODIGY	- (Induction)	- (Action and Domain Axioms)	No	Future- and Present-Oriented Abstraction Planning
SOAR	Deduction	No	Yes	Present-Oriented Planning
ACT-R	Deduction	No	Yes (no retract)	Future-Oriented Action Selection
3T	-	No	Yes	Future-Oriented Planning
DESIRE	Deduction	Model-Dep.	Model-Dep.	Model-Dep.

Figure 6.5: Comparison of Proof Calculus

Implementation	*Host Platform*	*Relation to Specification*	*Implementation of Concurrency*
InteRRaP	Oz, (Java)	formally derived	fair, asynchronous
Subsumption	Lisp, Hardware	derived	synchronous to parallel
dMARS	C	formally derived	depth-first
Agent0	C	derived	fair, synchronous
ALP	Prolog	formally derived	depth-first
PRODIGY	Lisp	loosely coupled	No
SOAR	C	loosely coupled	fair, synchronous
ACT-R	Lisp	loosely coupled	fair, synchronous
3T	C, Hardware	loosely coupled	fair, asynchronous
DESIRE	Animation	part. formally derived	Model-Dep.

Figure 6.6: Comparison of Implementation

sivity of their computational model by rendering concurrency unfair. As aforementioned, depth-first approaches in selection introduce a persistent behaviour but tend to stick the agent's decisions. A reasonably weighted, fair approach is therefore advisable.

Formal derivations of implementations are tightly related to formal computational models of proof calculi which are only provided by InteRRaP, dMARS, ALP, and DESIRE. Where the algorithmic issues are presented in detail, a loose relation of implementation and specification is to be tolerated, however.

Bibliography

[1] J. R. Anderson. *Rules of the mind.* Lawrence Erlbaum Associates, Hilldsdale, NJ, 1993.

[2] Joseph Bates, A. Bryan Loyall, and W. Scott Reilly. Broad agents. *SIGART Bulletin*, 2(4), August 1992.

[3] J. Blythe and W. S. Reilly. Integrating reactive and deliberative planning for agents. Technical Report CMU-CS-93-155, Carnegie Mellon University, 1993.

[4] R. P. Bonasso, D. Kortenkamp, D. Miller, and M. Slack. Experiments with an architecture for intelligent, reactive agents. In *Intelligent Agents II*, Lecture Notes in Artificial Intelligence. Springer, 1995.

[5] Rodney A. Brooks. A robust layered control system for a mobile robot. In *IEEE Journal of Robotics and Automation*, volume RA-2 (1), pages 14–23, April 1986.

[6] Rodney A. Brooks. Intelligence without reason. Technical Report 1293, MIT AI Laboratory, April 1991.

[7] J. G. Carbonell, C. A. Knoblock, and S Minton. PRODIGY: An integrated architecture for planning and learning. In *Architectures for Intelligence.* Lawrence Erlbaum Associate, 1991.

[8] J. Davila. Reactive Pascal and the Event Calculus. In U. Siegmund and M. Thielscher, editors, *Proceedings of the FAPR'96 Workshop on Reasoning about Actions and Planning in Complex Environments*, volume 11 of *Technical Report AIDA*, June 1996.

[9] M. d'Inverno, D .Kinny, M. Luck, and M. Wooldridge. A Formal Specification of dmars. In *Intelligent Agents IV*, volume 1365 of *Lecture Notes in Artificial Intelligence*. Springer, 1998.

[10] B. Dunin-Keplicz and J. Treur. Compositional formal specification of multi-agent systems. In *Intelligent Agents*, volume 890 of *Lecture Notes in Artificial Intelligence*, pages 102–117. Springer, 1994.

[11] N. J. Nilsson (ed.). Shakey the robot. Technical report, SRI AI Center, April 1984.

[12] C. Elsaesser and R. MacMillan. Representation and algorithms for multi-agent adversarial planning. Technical report, The MITRE Corporation, 1991.

[13] K. Eshghi. Abductive planning with event calculus. In *Proceedings of the Fifth International Conference on Logic Programming*, pages 562–578, 1988.

[14] R. James Firby. Task networks for controlling continuous processes. In *Proceedings of the 2nd International Conference on Artifical Intelligence Planning Systems*, 1994.

[15] T. H. Fung and R.Kowalski. The IFF Proof Procedure for Abductive Logic Programming. *Journal of Logic Programming*, 1997. in press.

[16] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the 6th National Conference on Artificial Intelligence*, 1987.

[17] C. Gerber and C. G. Jung. Towards the bounded optimal agent society. In C. G. Jung, K. Fischer, and S. Schacht, editors, *Distributed Cognitive Systems*, number D-97-8 in DFKI Document, Saarbrücken, 1997. DFKI GmbH.

[18] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, (12):576—580 and 583, 1969.

[19] C. G. Jung. Emergent mental attitudes in layered agents. In *Proceedings of the 5th International Workshop on Agent Theories, Architectures, and Languages ATAL'98*, 1998. to appear.

[20] C. G. Jung. On the Role of Computational Models for Specificying Hybrid Agents. In *Cybernetics And Systems'98*, Vienna, 1998. Austrian Society for Cybernetic Studies.

[21] C. G. Jung. Situated abstraction planning by abductive temporal reasoning. In *Proceedings of the 13th European Conference on Artificial Intelligence ECAI'98*, 1998. to appear.

[22] C. G. Jung and K. Fischer. A Layered Agent Calculus with Concurrent, Continuous Processes. In *Intelligent Agents IV*, volume 1365 of *Lecture Notes in Artificial Intelligence*. Springer, 1998.

[23] C. G. Jung, K. Fischer, and A. Burt. *Multiagent Planning using an Abductive Event Calculus*. Number RR-96-4 in DFKI Research Report. DFKI GmbH, Saarbrücken, Germany, 1996.

[24] R. Kowalski. *Logic for Problem Solving*, volume 7 of *Artificial Intelligence Series.* Elsevier Science Publisher B.V. (North-Holland), 1979.

[25] R. Kowalski. Using meta-logic to reconcile reactive with rational agents. In K. Apt and F. Turini, editors, *Meta-Logic and Logic Programming*, pages 227–242. MIT Press, 1995.

[26] R. Kowalski and F. Sadri. Towards a unified agent architecture that combines rationality with reactivity. In D. Pedreschi and C. Zaniolo, editors, *Logic in Databases*, volume 1154 of *Lecture Notes in Computer Science.* Springer-Verlag, 1996.

[27] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[28] J. P. Müller. *The Design of Intelligent Agents: A Layered Approach*, volume 1177 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag, December 1996.

[29] A. Newell. *Unified theories of cognition.* Harvard University Press, Cambridge, London, 1990.

[30] D. Poole and K. Kanazawa. A decision-theoretic abductive basis for planning. 1994.

[31] A. S. Rao and M. P. Georgeff. Modeling Agents Within a BDI-Architecture. In R. Fikes and E. Sandewall, editors, *Proc. of the 2rd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484, Cambridge, Mass., April 1991. Morgan Kaufmann.

[32] M. Shanahan. Planning and event calculus revisited. In *Proceedings of European Conference on Planning Systems*, 1997.

[33] Y. Shoham. Agent-oriented programming. Technical report, Stanford University, 1990.

[34] G. Smolka. The Oz Programming Model. In Jan van Leeuwen, editor, *Computer Science Today*, Lecture Notes in Computer Science, vol. 1000, pages 324–343. Springer-Verlag, Berlin, 1995.

[35] M. Spivey. *The Z notation (second edition).* Prentice Hall International, Hempel Hempstead, England, 1992.

[36] G. Wetzel. A unifying framework for abductive and constraint logic programming. In *Proceedings of the 12th Workshop on Logic Programming (WLP'97).* LMU München, September 1997.

[37] M. Wooldridge. Practical Reasoning with Procedural Knowledge: A Logic of BDI Agents with Know-How. In *Proceedings of the International Conference on Formal and Applied Practical Reasoning*. Springer-Verlag, 1996.

[38] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2), 1995.

[39] S. T. Yu, M. Slack, and D. P. Miller. A streamlined software environment for situated skills. In *Proceedings of the AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space (CIRFFSS'94*, 1994.

Methodological Comparison of Agent Models

Christoph G. Jung and Klaus Fischer

RR-98-01

Research Repor