gefördert vom
Bundesministerium
für Forschung
und Technologie

Verbmobil
Verbundvorhaben

Distributed Control in Verbmobil

Marcus Kesseler

IMMD VIII/Universität Erlangen-Nürnberg

Report 24
August 1994

August 1994

Marcus Kesseler

IMMD VIII – Künstliche Intelligenz
Universität Erlangen-Nürnberg
Am Weichselgarten 9
D-91058 Erlangen

Tel.: (09131) 699 118 - 118
e-mail: `kesseler@immd8.informatik.uni-erlangen.de`

Gehört zum Antragsabschnitt: 15.7: Architektur integrierter Parser für gesprochene Sprache

Das diesem Bericht zugrundeliegende Forschungsvorhaben wurde mit Mitteln des Bundesministers für Forschung und Technologie unter dem Förderkennzeichen 01 IV 101 H9 gefördert. Die Verantwortung für den Inhalt dieser Arbeit liegt bei dem Autor.

Contents

1 Introduction

> Because of the complexity gap between the systems being modeled formally and the applications that are being studied, CDPS [Cooperative Distributed Problem Solving] research has yet to adequately define rigorous approaches that work in real-world applications.
>
> Durfee, Lesser and Corkill in [4, page 145]

Verbmobil differs in one crucial aspect from all previous projects, large and small, in Natural Language & Speech Processing (NLSP): The degree of distributedness in the Verbmobil implementation effort is unparalleled. The groups implementing the different VM-modules have almost no constraints on their intra-modular design decisions. Interfaces are defined largely by bilateral agreements, which seldom make any references to the control issues arising from their usage. No group in VM is *explicitly* responsible for the definition of an overall control paradigm for VM. This, as we will see below, raises some serious problems. Nevertheless, we will argue that these constraints can also be perceived as an opportunity to break new ground in the area of control of complex distributed systems.

2 Architectures for NLSP Systems

The question of control in VM is tightly knit with the architecture of the VM system. So that we first need to clarify what the architecture of the VM system looks like.

The Verbmobil TP15 Group "Architecture" is responsible for research and development in the area of architectures for NLSP systems. To this end, in TP15 we have been working with a system that is a functional subset of Verbmobil. While Verbmobil aims at understanding, translating and generating spoken language, in TP15 we are content with a smaller, exemplary system that produces semantic representations from spoken language[1] input.

Note that the concept of *architecture* is often applicable to more than one layer within any complex system. In [19] Marr introduced the following three description levels for complex systems in computer science:

The **computational theory** is a reconstruction of the target domain in terms of a specification of what is to be computed and what constraints can be assumed to be valid.

[1]With the usual, state-of-the-art, constraints: Restricted vocabulary, narrow scenario, etc.

Representations that reflect the relevant properties of the objects in the domain must be found together with **algorithms** to manipulate them.

On the **implementation** level, the representations and the algorithms are mapped onto data structures and instruction sequences on a real machine.

Given this framework, an architecture for an NLSP system is a computational theory that tries to reflect human cognitive abilities in speech processing. Using Briscoe [5] as a starting point, Görz [13] gave the following characteristics of a general computational theory for NLSP:

Modularity: There is a widespread consensus that NLSP systems should be layered into several more or less independent modules (also referred to as knowledge sources, processes, agents, etc). Arguments in favor of modularity have been derived from various research fields: evolution, linguistics, cognitive science, software engineering, etc (see [5, page 23]).

Nowadays no one seriously questions the validity of the modularity assumption. Indeed, a project like Verbmobil, a research effort distributed across many universities and companies, would hardly be feasible without an intrinsically modular approach.

Incremental interpretation: As soon as a module finds a partial solution that spans parts of the received input hypotheses, it sends this solution to all other potential consumers of such output hypotheses. Or in the words of Briscoe [5, page 30]:

> ... every component of the human speech comprehension system will proceed with a minimum of delay and will convey the results of its analysis to the next stage of the system as rapidly as possible.

Interactivity: Information also flows top-down, that is, constraints coming from higher cognitive levels may be passed to lower levels for disambiguation.

Graceful degradation: When the quality of the input signal degrades one would not like to reach a sudden *incompetence threshold* under which the system is unable to provide any interpretation whatsoever. Rather, one would like to have a system that is able to degrade gracefully under worsening input signal quality.

Note that some of the concepts above are orthogonal to one another, that is, the use of one does not imply the presence of another. Especially modularity is well applicable in a simple NLSP system without the use of incrementality or interactivity. Such an architecture is precisely what is being implemented as the "Verbmobil Mini-Demonstrator".

We wish to emphasize that modularization is but one dimension of the architecture of an NLSP system. A further, and equally important dimension is the interaction mode used in the architecture.

As yet, the concept of architecture in Verbmobil has been used mostly to describe the overall modularization and the interfaces implied by the data flow between the modules. This architecture, called *domain architecture*[2], is incomplete in the sense that it does not specify any interaction strategies that go beyond the crude sequential interactions of the Mini-Demonstrator.

2.1 The TP15 INTARC Architecture

The TP15 "INTARC 1.2" architecture, as presented by Pyka in [20, 21], is a distributed software system that allows for the interconnection of NLSP modules under the principles of incrementality and interactivity. Figure 1 shows the modularization used in the newest INTARC demonstrator, version 1.3. In INTARC 1.3 there is a main, broad channel connecting all modules in the bottom-up direction, that is, from signal to interpretation. Furthermore, one can see the smaller channels connecting several modules, which are used for the top-down interactive disambiguation data flow. Incrementality, though not graphically represented in Figure 1, is required from all modules embedded in the INTARC 1.3 system.

INTARC 1.2 implemented the principle of limited working memory by introducing a time-based garbage collection (TBGC, see section 5.3) scheme for hypotheses. This required local hypotheses management to be under control of the local INTARC 1.2 demon attached to each module.

INTARC 1.2 has undergone a major revision on the implementation level in the first half of 1994. The revised architecture (ICE – INTARC 1.3 Communication Environment) is presented by Amtrup in [2]. We will not dwell on the technicalities of ICE, suffice to say that ICE also abides to the principles of modularity, incrementality and interactivity. In this first reimplementation the support of time-based garbage collection has been dropped. The feature turned out to be too coarse grained for the rather heterogeneous, fine-tuned hypotheses management strategies used in the different modules. For the same reasons the hypotheses management under control of the local INTARC 1.2 demon has been abandoned in ICE.

Given this transference of hypotheses management responsibilities back to the module implementor, ICE turns out to be much simpler than INTARC 1.2: It

[2] "Fachliche Architektur" in German Verbmobil parlance.

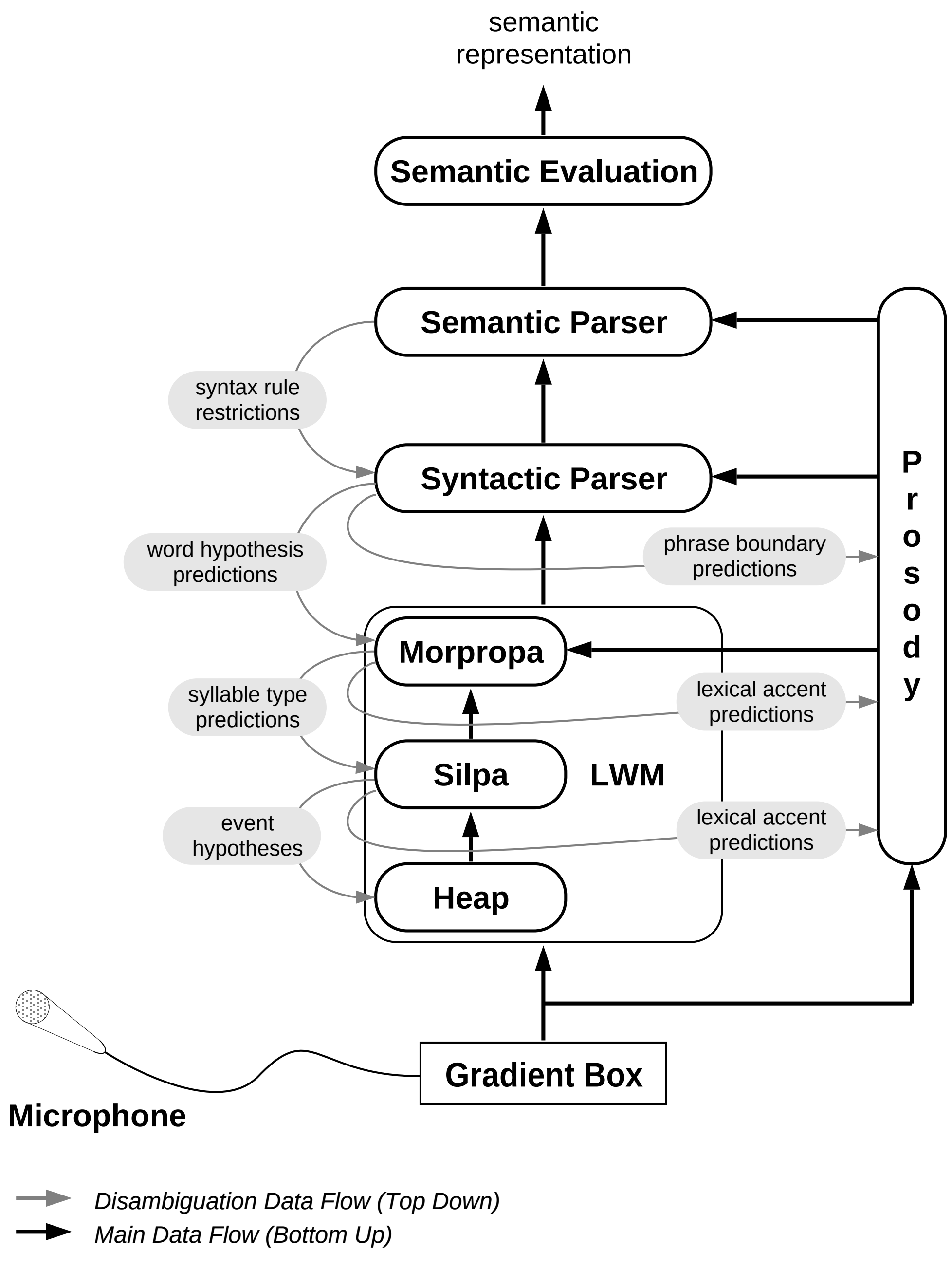
semantic representation
Semantic Evaluation
Semantic Parser
syntax rule restrictions
Syntactic Parser
word hypothesis predictions
phrase boundary predictions
Morpropa
syllable type predictions
lexical accent predictions
Silpa
LWM
event hypotheses
lexical accent predictions
Heap
Prosody
Gradient Box
Microphone
Disambiguation Data Flow (Top Down)
Main Data Flow (Bottom Up)

Figure 1: The interactive, incremental INTARC 1.3 architecture

is a communication architecture with an application programmer interface (API) tuned to the necessities of NLSP in general and Verbmobil in particular. ICE assumes that each module has a local memory that is not directly accessible to other modules. Modules communicate explicitly with one another via messages sent over bidirectional channels.

This kind of communication architecture is hardly new and confronts us directly with a large number of unresolved issues in distributed problem solving (see Durfee, Lesser and Corkill in [4]). In the last 20 years there have been numerous architecture proposals for distributed problem solving among computing entities that exchange information explicitly via message passing: *Actors* by Hewitt and Agha [1]; *Communicating Sequential Processes* by Hoare [16], which was used as the base for the development of the *OCCAM* programming language and its hardware companion, the INMOS *Transputer* [17]; *Linda* by Gelernter [11]; and more recently *Agents* (see Shoham [22]).

None of these models include explicit strategies or paradigms to tackle the problem of distributed control. Nevertheless we opted for a model along these same lines for two reasons: Firstly, such an architecture is a perfect one-to-one mapping of our chosen NLSP system design principles, and second, as we will argue in section 3.3, given the principle of modularity, there really is no other alternative.

3 Distributed Control in Verbmobil: The Problems

3.1 Centralized Control & Blackboards

Traditionally, NLSP systems have been centrally controlled. Since Hearsay-II [10] the use of blackboards, or variations thereof, has been the most popular control paradigm for NLSP systems. For a good overview of blackboard architectures see Engelmore & Morgan [9] and more recently Carver & Lesser [6].

In [6] Carver & Lesser point out the following two main advantages of centralized blackboard-based approaches over distributed ones, like for example, Craig's Cassandra architecture [7]:

Integrated view of hypotheses. Simply by chasing pointers it is possible for a control algorithm to inspect what lower level hypotheses led to the derivation of a more abstract hypothesis. The controller has "the whole picture". It is therefore possible to analyze potentially conflicting goals on much finer scale without incurring huge communication costs.

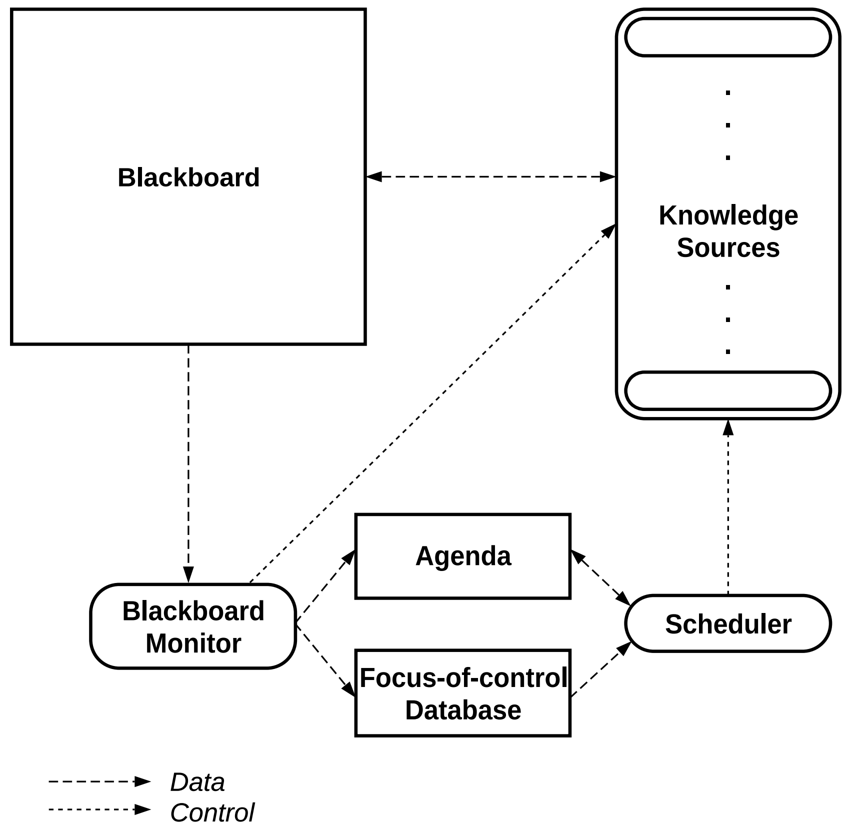

Figure 2: The classic blackboard model (from [6, page 9])

Opportunistic control decisions are easier to implement. The emergence of a promising partial solution on any level can be used for a more or less immediate reallocation of resources on all other levels.

Furthermore, a distributed control mechanism might need an arbitration protocol in cases where the controlling instances disagree over what is the best path to pursue. Centralized control obviously needs no such arbitration of priorities with itself.

At this point an obvious idea pops up: Why not use the advantages of centralized blackboard control and combine them with the high performance of modules running in parallel on a shared memory parallel processor (UMA-Architecture[3]). We see two problems:

1. On a lower level we have the problem of memory, or more precisely, bus contention. The *sustained memory bandwidth* of modern busses is up to one order of magnitude smaller than the data consumption/production rate of

[3]UMA – Uniform Memory Access

the respective processors. This is valid even for single processor systems, which is why computer architectures use memory caches. The connection of more processors onto the same bus of course only exacerbates the problem. The n processors only work truly in parallel when $n-1$ of them have all the necessary code and data in the their local caches. All shared memory architectures suffer from this fundamental drawback.

The use of languages like Lisp and Prolog in the higher levels of Verbmobil aggravates the problem further, since they exhibit very low memory locality, that is, the cache hit rate of Lisp and Prolog programs tends to be low.

2. Even disregarding bus contention, we might still have to struggle with blackboard contention. To maintain consistency a single-write/multiple-read synchronization protocol has to be enforced for all blackboard accesses. Such protocols can lead to the *serialization* of the modules, thereby loosing all advantages of a parallel implementation.

The solution to the first problem is obviously to use a distributed architecture, where each processor has its own bus and sustained memory bandwidth therefore increases linearly with the number of processors used. To circumvent the second problem, we would also have to distribute the blackboard.

3.2 Distributed Blackboards

In accordance with Marr's layered view of complex systems, we might see a blackboard as an abstract model in the sense of a computational theory. It is therefore more or less straightforward to implement it as a *distributed blackboard*. The problem with this approach is the strategy used to keep the distributed blackboard consistent. There are two possibilities:

1. All local blackboards contain the same information and are kept consistent by a corresponding update protocol which uses the communication network.

   What has been said above about serialization due to synchronization of blackboard accesses of course is also valid here. Much more so, since communication networks tend to be about one order of magnitude slower than busses, assuming a dedicated communication subsystem, and 2–3 orders of magnitude slower on LAN-based communication.

2. The local blackboards contain only hypotheses pertaining to their own needs. This would obviously solve the problem of blackboard contention, yet we would loose the main advantages of the blackboard approach (integrated view, opportunistic control decisions).

Furthermore, the modules can no longer simply dump all their findings into the blackboard and rely on the monitor/scheduler to make the best of it. Modules now have to *decide* what hypotheses to send to which module. In other words, we have to deal with all the problems of distributed control. We will analyze these problems in more detail in next section.

For a thorough discussion of these issues see the four papers in section II of Jaganathan [18, pp 77–178]

3.3 The Structural Constraints of Verbmobil

As is perfectly obvious from the project definition of Wahlster et al [23], the principle of modularity of NLSP systems is a fundamental assumption in Verbmobil. Given the large number of organizations distributed all over Germany, the implementation of a final Verbmobil system would be almost impossible without a modular implementation paradigm.

But modularity does of course still leave us with two problems: First, modules have to communicate with one another, and second, their local behaviors have to be somehow coordinated into a coherent global, possibly optimal, behavior.

As a first tentative solution to both problems we might suggest the creation of a special group that is explicitly responsible for these global integration issues, which is more or less the declared task of the Verbmobil 16 Group. Yet, we feel that there are tight limits to the degree of integration that is achievable by such a group. These limitations, which we call the *structural constraints* of Verbmobil, are mostly pragmatic in nature:

- Some of the modules are very complex software systems in themselves. Highly parameterizable and with control subtly spread over many interacting submodules, understanding and then integrating such systems into a Verbmobil control strategy can be a very daunting task, even disregarding software integration issues like different development platforms and/or languages.

- Control issues are often very tightly knit with the domain the module is aimed at, ie, it is very difficult to understand the control strategies used without sound knowledge of the underlying domain. For example, to fine-tune the performance of a parser a good deal of knowledge of parsing, syntax, unification, etc is highly recommended. The problem only gets worse if what is to be fine-tuned is the *interaction* between several complex modules.

These two arguments are similar in nature, but differ in the levels, in Marr's sense, that they apply to. The former is implementation related, the latter algorithm and theory related.

3.4 Layers of Control

If we accept the primacy of modularity and the unfeasability of centralized control, we are left with highly distributed architecture much along the lines of the TP15 INTARC 1.3 proposal. In such highly distributed systems we will generally find the following levels of control:

System Control: The minimal set of operating system related actions that each participating module has to be able to perform. This set will typically include the means to start up, reset, monitor, trace and terminate individual modules or the system as a whole (See section 5.1 for a minimal set of system control features).

This kind of control will be exercised by the user through some kind of central monitoring tool. Typically a module of its own, with a graphical user interface (GUI) to the system control functions. Figure 3 shows the INTARC 1.3 architecture augmented by such a *master console*.

Note that system control is on a very general and coarse level which does not address the finer issues of domain related control.

Isolated Local Control: The control strategies used within the module disregarding any interactions beyond initial input of data and final output of solutions.

Figure 4 gives a rough idea of such a module. On startup the module enters a waiting loop (`i`), that may be implemented either as a busy wait (polling) or by some interrupt driven scheduler provided by the operating system (eg sockets).

On data entry the module does any number of iterations over `a`, `b` and `c` until some criterion is satisfied in `b`, upon which the results are put out and the module returns to the waiting loop `i`. There is only one thread of control active at any time.

Note that even if local control is very complex this fact is irrelevant to the user of such a module since it has only a single entry and a single exit. The use of isolated local control only makes sense in a distributed system that has a purely sequential architecture, like the Verbmobil Mini-Demonstrator.

Interactive Local Control: In a first approximation this form of control can be seen as isolated local control extended with interaction capabilities. Figure 5

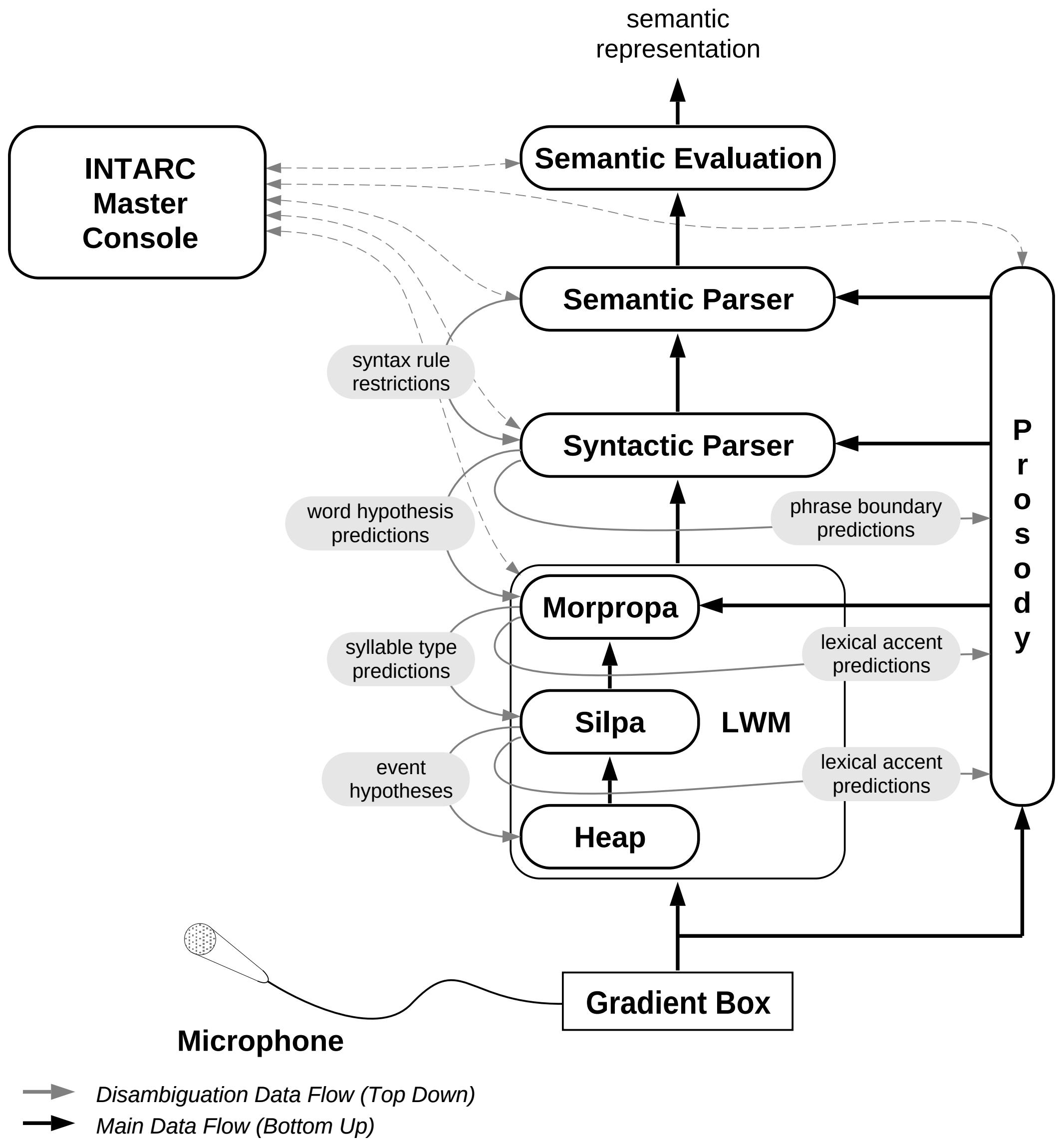
semantic representation
INTARC Master Console
Semantic Evaluation
Semantic Parser
syntax rule restrictions
Syntactic Parser
word hypothesis predictions
phrase boundary predictions
Prosody
Morpropa
syllable type predictions
lexical accent predictions
Silpa
LWM
event hypotheses
lexical accent predictions
Heap
Gradient Box
Microphone
Disambiguation Data Flow (Top Down)
Main Data Flow (Bottom Up)

Figure 3: The interactive, incremental architecture with a master console

shows the isolated local control of figure 4 (in gray) with extensions to deal with incrementality and interactivity. **Incrementality** is given by the possibility of control flowing back to `b` after an output operation, so that output can be finer grained. Higher **interactivity** is made possible by entering `i` more often from various points within the module and by adding a new waiting loop `j` to check for any top down requests. To process top down requests the module will usually have to be extended with further control loops (`d`). To allow for fast response times `i`

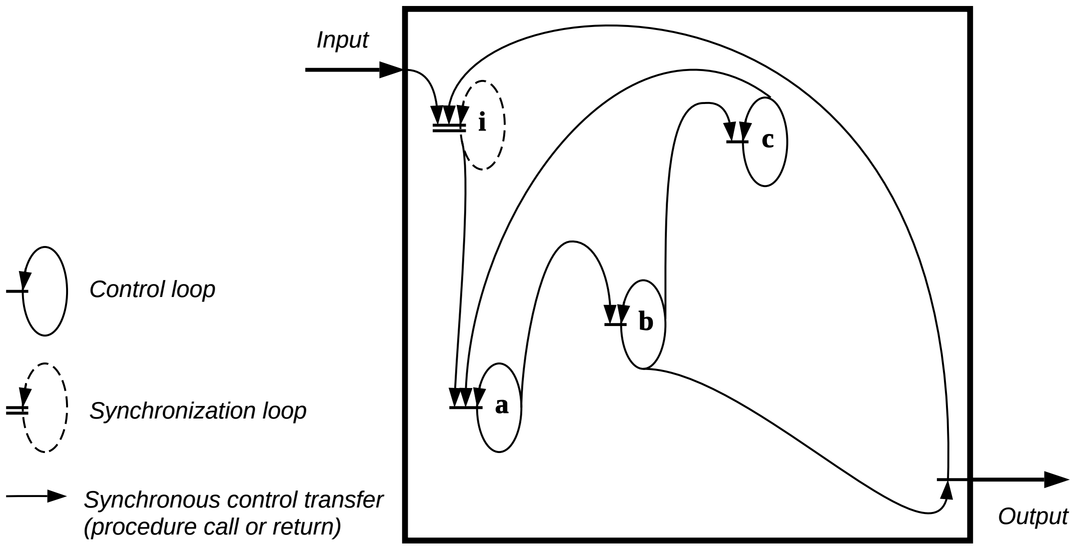

Figure 4: A module with isolated local control

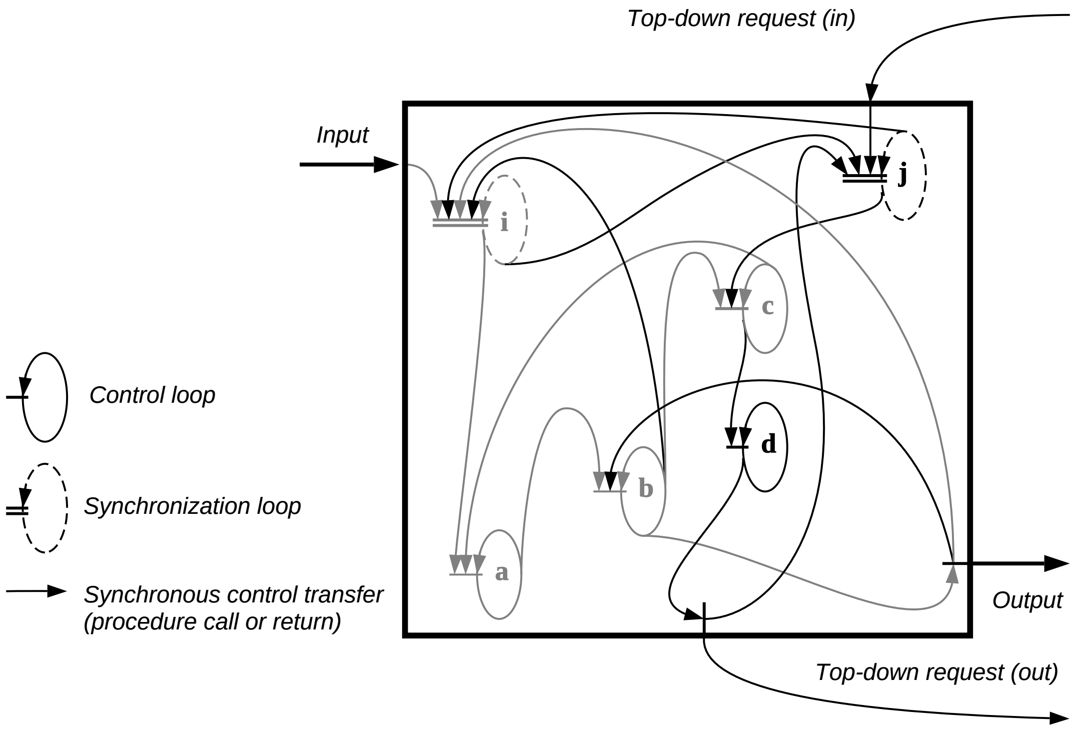

Figure 5: The same module with interactive local control

and `j` are non-blocking in the interactive version. That is, they just check if there are requests or data available and return immediately if nothing is available.

Despite the ad-hoc character of figure 5 (please don't bother following every connection in it), in our experience the change of control flow when going from isolated local control to interactive control will tremendously increase the complexity of the resulting code, which is essentially the idea figure 5 is trying to convey.

Notwithstanding the increased complexity of figure 5, we are still making several simplifying assumptions:

1. That the algorithm represented by `a`, `b` and `c` can be used incrementally. There are algorithms which are not usable in an incremental fashion, or at least, not without incurring performance losses. As an example the fairly popular $A*$ search technique works backwards in time, which obviously makes its use in an incremental way difficult.

   Incrementality can lead to a demand for complete redesign of a module. By complete redesign we mean to question if the computational theory and/or the algorithms employed are right for the task at hand.

2. That simply by exchanging data and doing simple extensions in the control flow everything will balance out nicely on the system scale.

The second assumption is, of course, enormously naive. Just to keep on par with the sequential architecture implied by our starting module in figure 4 we have to solve a whole plethora of new problems that come along with interactivity:

- Modules can now *deadlock* one another by, for example, answering to a top-down request with a request to the requesting module.
- Modules can now *live-lock*[4] one another, by sending so much requests that little or no useful computation is done.
- There may be race conditions (missing synchronization), which in the worst case lead to spurious, undeterministic errors, which tend to be extremely difficult to locate.
- There may be over-synchronization, which leads to serialization of the modules, so that we dot not gain anything from the higher parallelization that our architecture seemed to imply.

[4]For a discussion of deadlock and live-lock in the context of Distributed Artificial Intelligence (DAI) see Craig [7, page 103].

In short, we have all the usual problems of distributed systems to deal with. Unfortunately there is no way to shield the module implementors from these pitfalls, usually associated with operating systems theory. Even if the the lower levels of the VM communication system are reliable and deadlock-free, this in itself constitutes no guarantee whatsoever against deadlocks. Two (or more) processes can *decide to wait* for results of one another on their highest control levels. Deadlock problems can potentially arise on every abstraction layer in a complex, multi-level system.

Sophisticated Local Control: In spite of all the complexity implied by interactive local control we did not yet present a way to handle the finer issues of domain related control. That is, find *local* control strategies that make the modules behave cooperatively on a system-wide scale.

This is a classic problem in a great many sciences. Management gurus demand their followers to "Think globally, act locally". In DAI Durfee et al [8, page 147] ask "How to get agents to make intelligent communication decisions that influence each other to their [mutual] advantage?". In Physics of Nonlinear Phenomena, also called Synergetics, (see Haken [14]) one says that cooperation should be an *emergent property* of the interactions between the local control strategies.

As yet, there is no proven methodology to deal with such problems in a generally applicable manner (See Whitehair & Lesser [25] for an interesting, if fairly complex, proposal). Even if we had such a methodology, we would still run against the structural constraints in Verbmobil, since its application would have to be decided centrally and agreed to by all participating groups, which is unrealistic in the current phase of the project. The description of our proposed way out of this dilemma is deferred until section 4.

Dialogue Control: In NLSP systems capable of dialogue, like Verbmobil, there *is* a module that comes close to possessing the "integrated view" of a centralized blackboard control. The *dialogue module* has a good abstract description of what is happening in the system on a level that is quite close to a human's idea of a dialogue[5]. Therefore the dialogue module seems the right place to handle some of the strategic global control issues, like:

Domain error handling: When a module is unable to cope with the input data (eg: user was not speaking loud enough), it will typically send a failure notification[6] to the dialogue module. The dialogue will then trigger the generation module to issue a polite request to the user to repeat the last utter-

[5]Lately the transfer module has also been suggested as a potential high-level controller.

[6]We are supposing the module is able to recognize its limitations and act accordingly.

ance and then reset all afflicted modules into states from where computation can restart or proceed.

Observe timeout constraints: If the system does not start the utterance of a translation, say, 6 seconds[7] after the user stopped speaking, then enter *I-beg-your-pardon* mode and trigger a restart.

Resolve external ambiguities/unknowns: By "external" we mean an ambiguity than is not resolvable within the system so that there is no alternative than to ask the user. Again the dialogue module would trigger semantics and generation to formulate a question to this end.

The tricky question with external ambiguities is, of course, how to identify them. In its normal mode of operation the system will already be dealing with lots of ambiguities that will simply be passed on to transfer and then to generation.

The fact that the dialogue module exercises a kind of *global control* does not invalidate what has been said about the unfeasability of central control in Verbmobil. The control exercised by the dialogue module is very coarse grained. To handle finer grained control issues in any module would take us back to memory and/or communication system contention.

4 Distributed Control in Verbmobil: A Solution?

The outlook on distributed control in Verbmobil presented so far is gloomy at best. The enormous complexities already present in the many different control levels are further exacerbated by the "geopolitical" distributedness of Verbmobil itself. As already mentioned above, currently there is no known method to tackle such problems reliably.

Given all the presented constraints, the only feasible way out seems to be an *evolutionary* approach:

1. Start from what is given, that is:

   (a) A heterogeneous set of highly specialized software development groups.

   (b) A set of complex modules. Each implemented as an autonomous software system (process).

[7]In non real-time research prototypes the timeout can be set to $6k$ seconds, where k is a factor that expresses how much slower than real-time the current implementation is.

   (c) A message passing communication system that serves as the glue between the modules.

2. Bilaterally agree on a set interactions between the modules. For the Mini-Demonstrator this set was trivially implied by the modules used in it and the fact that information flows only from lower to higher cognitive levels. Top-down interactions for disambiguation are more complex, but can be added one by one to the base system.

3. Specify the protocols for the interactions in bilateral agreement rounds. There are two sides to this: First, there is the problem of what data to exchange in which format represented by what data types. This is already taking place in Verbmobil quite routinely.

   Second, we need precise, step-by-step specifications for the exchanges of data or control information between two or more modules, with regard to handshake modalities (after a reset), synchronization, timeouts, etc. Please keep in mind that the controllers needed to support highly interactive protocols in a sophisticated architecture are bound to be quite complex.

   Note that here we encounter an important structural barrier in Verbmobil: Typically, implementors cannot simply chat about interface issues over coffee break twice a week. Unless two groups are in the same organization (and building!) any bilateral agreement of protocols will demand considerable efforts of the parties involved.

4. Distribute complete system implementations as widely as possible within Verbmobil. Given good visualization tools, observations based on experiments will likely lead to fruitful insights into the nature of bottlenecks, new interaction opportunities and strategies, meta-information or constraints worth transmitting to neighboring modules, etc.

Given this procedure, it is likely that control will *evolve* from the initial stage of a rather simple sequential architecture to a partially[8] incremental architecture, on to a globally unoptimized but interactive architecture. The final stage in Verbmobil evolution would then be a system that is able to achieve optimized, globally cooperative behavior based on local controllers with limited information.

[8]Some modules would have to be completely reimplemented to be usable in an incremental fashion.

5 Towards Distributed Control

This section presents some building blocks needed on the way to distributed control in Verbmobil.

5.1 A Protocol for System Control

The following list gives a minimal set of operations that each module has to implement in order to conform to the requirements of global system control:

Trace or Monitor: Send information describing the internal state of the module to some central monitoring module. Such actions might be implemented as a switch that makes the module send status reports in regular intervals or simply as one-time requests that are answered with a corresponding report.

Reset: Go into a well defined starting state from which operation can then proceed. Note that in a system as complex as Verbmobil there are different *reset granularities*. In a parser, for example, the following levels of reset make sense (with falling granularity):

> **Integral Reset:** Essentially equivalent to killing the process and restarting it all anew. Parameter files, grammars, lexicons, etc are reloaded and the communication system is restarted in an integral reset.
>
> **State Reset:** Clear the chart but keeps all parameters, grammars, lexicons, etc.
>
> **Communication Reset:** Break off and then re-establish communication with the respective partners. Probably meaningful only in conjunction with a state reset.
>
> **Utterance Reset:** Clear the chart up to the last recognized utterance. Here things start to get quite complicated. A reset on this granularity only makes sense if all the other partners are reset in a consistent manner.

Other modules might have even finer meaningful reset points. Notice that finer reset granularities will usually be more difficult to implement, since such resets will work only if the correct contexts are kept across modules.

Exit: Obvious.

5.2 Using Channels

The ICE communication system allows module implementors to set up an arbitrary number of channels between the modules in the system. ICE channels are similar to those presented by Hoare in CSP [16] and implemented in the OCCAM 2 programming language [17].

Channels are objects that implement point-to-point bidirectional communication between two processes[9]. A channel transports *messages* between the processes it connects and will basically support four operations:

`Send`: Dispatch a message to the receiving process. A `Send` operation returns immediately, ie, it does not matter if the receiving process is explicitly waiting for a message on this channel[10]. This has two consequences: First, communication is *asynchronous*, second, communication is *buffered*, since the sent messages have to be kept somewhere until the receiver is prepared to retrieve them from the corresponding input buffer.

Asynchronous, buffered communication generally has performance advantages over synchronous, unbuffered communication, since it *decouples* processes, thereby allowing a greater degree of parallelism. We do, of course, incur some costs: Buffer space is not infinite and precautions may have to be taken not to exhaust available buffer resources (memory). This means switching to synchronous (blocking) communication if buffers get scarce.

`Recv`: A *blocking receive*, that is the calling procedure *waits* until a message gets available on the channel. Useful for modules operating in lock-step mode.

`nRecv`: A *non-blocking receive*, that is the call to this procedure returns immediately, either with the oldest message in the channel input queue or the void message value.

`tRecv`: A *blocking receive with timeout*, ie, after expiration of a timeout value given as parameter the call returns with a value denoting a *void message* (in C typically a NULL pointer) if none was available.

One of the main advantages of using channels is that they impose structure on the information flow: Typically a channel will be used for each type of data flowing between any two modules. This allows for specialization of the channels according to the processing needs of the data involved: Buffered, low-priority channels for

[9]The two processes may be running on the same or on different computers connected by some communication system.

[10]In CSP and OCCAM communication only takes place if *both* partners are ready to communicate. Every communication in such a model is therefore also a synchronization.

bottom-up hypotheses; unbuffered, medium-priority channels for top-down predictions; buffered, high-priority channels for system control (trace, reset, exit), etc.

5.3 Quality-based Buffering and Time-based Garbage Collection of Hypotheses

In a system without global control modules are prone to *live-locking* one another. Live-lock is a condition in parallel system where modules are so busy sending and receiving data, that little or no time is left for useful work.

An obvious way to tackle such problems is the development of local control strategies that try to strike a good balance between communication and computation. Of course such an approach would confront us with the complexities of sophisticated local control.

As an alternative, we propose a mechanism that tries to avoid the pitfalls of sophisticated local control, and yet manages to balance communication and computation based on limited local information.

The mechanism has two requirements: First, that hypotheses traveling over a channel contain a quality score that reflects the certainty of the respective hypothesis. Second, that every hypothesis also includes a time frame for which it is considered to be valid or applicable.

Using this information we extend channels by the following features:

1. A **channel quality threshold** against which all hypotheses waiting to be dispatched over the channel are set. Hypotheses whose score is below the channels current threshold are buffered in a *local* send queue instead of being sent to the destination process immediately.

   The threshold should ideally only be set-able by the receiver, so that it has a means of damping (or increasing) message transmission rates according to its own processing needs or capabilities.

2. A **time limit** after which hypotheses with scores below threshold are removed from the send queue. This avoids an overflow of buffer space with low score hypotheses.

   We call this time-based garbage collection (TBGC). Garbage collection[11] is a technique used in dynamic memory management schemes to reclaim used but provably inaccessible storage space and recycle it into the unused memory pool. For an overview of such techniques see [3].

[11]Ecologically speaking, the term "garbage collection" is a misnomer, since the garbage is not only collected but also *recycled* with a 100% efficiency.

This mechanism is an extension of the time-frame based filtering proposed by Pyka in [20]. Adaptive behavior of channels can be implemented with ideas such as automatic threshold decay to default values or auto-regulation based on hypotheses throughput statistics. The quality threshold and the time limit can be seen as parameters that define the *viscosity* of the hypotheses flowing through the channel. Experiments with INTARC 1.3 will show if the mechanism succeeds in avoiding live-lock without damping the flow of information so much as to hinder successful operation of the system.

5.4 Incrementality, Interactivity, Synchronization, Control and Algorithms

Using the ambiguity resolution mechanisms presented by Hauenstein & Weber in [15] as an example, we would like to show in more detail the range of complications arising from the use of interactivity.

We briefly describe the two different disambiguation strategies used by Hauenstein & Weber between an HMM-based word recognizer and a chart-based parser:

Verify Mode: After sending all word hypotheses terminating at a time frame F_t, the word recognizer (WR) waits for feedback from the chart parser (CP). Based on the current state of the chart and the the applicable grammar rules, the parser will accept some of the word hypotheses and reject the ones that do not fit any of the current valid paths. The accepted ones are sorted according to the scores derived from the paths that they belong to, and the best of them are sent back to the WR. The WR in turn revises its scores based on this *verification* and then enters the next cycle by extending its models to frame F_{t+1}. The basic idea is to pick winning paths in the WR much earlier than it would be possible without the verifications derived from syntactic constraints coming from the CP, thereby restricting search space.

Predict Mode: Before extending word hypotheses from frame F_t to frame F_{t+1} the WR waits until it gets a *prediction* for what words seem plausible given the current parse state from CP. All paths not compatible with the set of received word hypothesis predictions are discontinued by the WR, again restricting search space.

From the point of view of control it is interesting to note that Hauenstein and Weber used a strict *lock-step synchronization*: The CP would wait to receive the next bag of word hypotheses and then produce either a verification or a prediction. Meanwhile the WR would be blocked until the arrival of the verifications/predictions.

The gain in quality through top-down disambiguation was partially offset by the loss of parallelization in the WR–CP system. There were no more overlapping computations.

Given this loss, what would be the price, in terms of increased controller complexity, to get parallel processing back into the game? Superficially we would have to *decouple* the two modules. That is, the CP would simply produce verifications or predictions without bothering with the actual time-frame position of the WR. Both modules would then run in parallel again.

A question that immediately comes to mind is, if "prediction" is an apt name for data that arrives late and therefore concerns a time that is already in the past from the point of view of the WR? A simple strategy to deal with late arriving predictions would be to ignore them, which seems a wasteful loss of potentially useful syntactic restrictions. A better approach might be to downgrade the scores of the predictions and use them as mere *hints*[12].

If such hints are taken into account by the WR, they may lead to a *major revision* of the state of the WR from the time frame the prediction is about (say F_t) to the current time frame (say F_{t+4}). The new hypotheses for the time frame $[t, t+3]$ may even contradict the previously computed ones, which, to complicate matters further, may already have been sent to the CP. At this point, the right thing to do is to notify the CP that a revision of hypotheses has taken place and that a back-up to frame F_t is necessary. This rather simple sounding requirement violates one of the fundamental assumptions of chart parsing: that the chart grows monotonically. In order to do backtracking in a chart parser, the local control algorithms have to be completely redesigned.

Note also that chart backtracking can trigger a new cycle of predict/revise/backtrack. Interactive control has to make provisions against infinite feedback loops occuring in the system.

The purpose of this section is not to argue against interactivity in Verbmobil. Rather, we want to point out the complexities involved, so that algorithms fit for interactive architectures are chosen early enough in the respective development phases.

See Weber [24] for a control algorithm for a loosely coupled and therefore parallel WR-CP system.

[12]The reason to downgrade the value of the information is that the parser, having received further word hypotheses in the meantime, might also have changed directions.

6 Conclusion

The aim of this paper was to call attention to the enormous complexities associated with the problem of controlling a distributed software system as large as Verbmobil. To this end, we introduced two insights that are of crucial importance for the discussion of control in Verbmobil:

1. The concept of *architecture* is applicable to many layers within Verbmobil.

2. Likewise, *control* has many layers that interact with one another and is tightly intertwined with the architectural choices.

   Developers in Verbmobil need to be aware that any system architecture that goes beyond the simple sequential model, implies forms of control that may force them into complete revisions of their implementations or even algorithms.

Unfortunately there is no simple, ready-made solution to these problems. Computer science is still a long way from offering a sound theory of distributed control[13]. Furthermore, the sheer distributedness of Verbmobil, in terms of geography and its status as a joint research project, exacerbate the problems of reaching consensus on any issues spanning more than one architectural level in the overall system.

The Verbmobil TP15 Group is currently assembling the version 1.3 of INTARC. As depicted in figure 3 (page 11), INTARC 1.3 is highly interactive and consequently has very complex data flow paths. We expect the control problem to be very challenging indeed. All the involved modules are by now ready and stable. ICE, our communication environment, is in final beta testing. INTARC 1.3 is scheduled to be operational by April 95. We will produce a follow-up report on our experiences with distributed control in INTARC 1.3 shortly thereafter.

[13]Witness the enormous technical (and financial) problems the new airport in Denver is having with the distributed control of its luggage distribution system (see Gibbs [12]).

A A Formal Description of a Modular Architecture

In this section we present a formal description of a distributed system. The terminology is adapted Durfee et al. [8, page 95].

Let the distributed system be built of set of modules $\mathbf{M} = \{M_1, \ldots, M_m\}$. Each module is a 6-tuple $M_i = (\mathbf{S}, \mathbf{I}, \mathbf{D}, \mathbf{N}, \mathbf{O}, \mathbf{P})$ where:

$\mathbf{S} = \{S_1, \ldots, S_s\}$ is a set of states that the module can be in.

$\mathbf{I} = \{I_1, \ldots, I_i\}$ is a set of data inputs that have been sent from other modules to this module. The I_i may have arbitrary content, eg, control information, signal data, constraints from higher modules, etc.

$\mathbf{D} = \{D_1, \ldots, D_d\}$ is a set of data that reside in the module's local memory.

$\mathbf{N} = \{N_1, \ldots, N_n\}$ is a set of ternary input operators that map elements of the set $\mathbf{S} \times \mathbf{I} \times \mathbf{D}$ onto itself. Application of an operator N_k will typically transport elements of $\mathbf{I}$ to $\mathbf{D}$.

$\mathbf{O} = \{O_1, \ldots, O_o\}$ is a set of binary operators that map elements of the set $\mathbf{S} \times \mathbf{D}$ onto itself. The O_o represent the algorithms that the module implements.

$\mathbf{P} = \{P_1, \ldots, P_p\}$ is a set of binary projection operators that map elements of the set $\mathbf{S} \times \mathbf{D}$ in a module M_i onto itself thereby copying or moving one or more elements of $\mathbf{D}$ into the set $\mathbf{I}$ of one or more other modules. That is, the P_p transport hypotheses from $\mathbf{D}$-Space into other modules' $\mathbf{I}$-Spaces.

Hypotheses A piece of data received from another module describing part of the signal, or higher abstractions thereof, is called an *input hypothesis*. Note that input hypotheses are located in $\mathbf{I}$-Space upon receipt but may be transfered to $\mathbf{D}$-Space by a N_n operator application. A piece of data representing a (partial) solution produced by the algorithms implemented in the module is accordingly called an *output hypothesis*.

A hypothesis H can be represented by a triple $(T, d, \vec{v})$ where $T = [t_1, t_2)$ is a time interval over which the statement d applies. $\vec{v}$ is a vector of confidence values for d. Time intervals are likely to be more useful in the lower levels of an NLSP system, where there still is a strong correlation between signal and interpretation.

Time Let time be a local discrete quantity, ie, the time reference used in module M_a is not necessarily the same as in M_b and time proceeds in *ticks*. Furthermore, let $S(t)$ be a mapping from time t onto **S** and $I(t)$, $D(t)$, $O(t)$, and $P(t)$ the corresponding mappings for **I**, **D**, **N**, **O** and **P**.

Local Control The *local control problem* can now be formulated as: Choose $N_{n(t)}$, $O_{o(t)}$ and $P_{p(t)}$ such that

```
do
  $N_{n(t_0)}(S(t_0), I(t_0), D(t_0)) \rightarrow (S(t_1), I(t_1), D(t_1))$
  $O_{o(t_1)}(S(t_1), D(t_1)) \rightarrow (S(t_2), D(t_2))$
  $P_{p(t_2)}(S(t_2), D(t_2)) \rightarrow (S(t_3), D(t_3))$
until $X(S(t_3), I(t_3), D(t_3)) = true$
```

are optimal with respect to the aims of the module. Keep doing this until a termination criterion, represented by $X(...)$, is true.

Note that the choices of $N_{n(t)}$, $O_{o(t)}$ and $P_{p(t)}$ are not functions of **S**, **I** and **D**. The values of the indices $n(t)$, $o(t)$ and $p(t)$ will usually be *constrained* by $S(t)$, $I(t)$ and $D(t)$, but not *determined* by them. Therefore we have an explicit operator choice mechanism present in each module.

The VM Mini-Demonstrator uses an architecture that Briscoe [5, page 50] has dubbed *the autonomous, serial model*, which has a purely sequential architecture that uses neither incrementality nor interactivity, can be described as follows within this framework:

N and **P** consist of single element sets: $\mathbf{N} = \{N_1\}$ and $\mathbf{P} = \{P_1\}$. N_1 and P_1 are each used exactly once. N_1 to import the solutions found by the lower level module and P_1 to export any solutions found to the next higher module. In this framework the local control is reduced to: Choose $O_{o(t)}$ such that

```
$N_1(S(t_0), I(t_0), D(t_0)) \rightarrow (S(t_1), I(t_1), D(t_1))$
do
  $O_{o(t_1)}(S(t_1), D(t_1)) \rightarrow (S(t_2), D(t_2))$
until $Y(S(t_2), D(t_2)) = true$
$P_1(S(t_2), D(t_2)) \rightarrow (S(t_3), D(t_3))$
```

is optimal with respect to the aims of the module. Keep doing this until a termination criterion, represented by $Y(...)$, is true.

References

[1] Gul Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* The MIT Press, 1986.

[2] Jan Amtrup. ICE - INTARC Communication Environment: Design und Spezifikation. Verbmobil Memo 48, Universität Hamburg, FBI/UHH, August 1994.

[3] Andrew W. Appel. *Garbage Collection*, pages 89–100. MIT Press, Cambridge, Massachusetts, 1991.

[4] Avron Barr, Paul R. Cohen, and Edward A. Feigenbaum. *The Handbook of Artificial Intelligence*, volume 4. Addison–Wesley Publishing Company, Massachusetts, 1989.

[5] E. J. Briscoe. *Modelling human speech comprehension.* Ellis Horwood Ltd., Chichester, 1987.

[6] Norman Carver and Victor Lesser. The evolution of blackboard control architectures. Technical Report 92–71, CMPSCI, October 1992.

[7] Iain D. Craig. *The CASSANDRA Architecture: Distributed Control in a Blackboard System.* Ellis Horwood Limited, Chichester, 1989.

[8] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. *Cooperative Distributed Problem Solving*, pages 83–147. Volume 4 of Barr et al. [4], 1989.

[9] Robert Engelmore and Tony Morgan. *Blackboard Systems.* Addison–Wesley Publishing Company, Massachusetts, 1988.

[10] L.D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The HEARSAY-II speech understanding system: Integrating knowledge to resolve uncertainty. *ACM Computing Surveys*, 12:213–253, February 1980.

[11] David Gelernter. Getting the job done. *Byte*, 13(12):301–308, December 1988.

[12] W Wayt Gibbs. Software's chronic crisis. *Scientific American*, 271(3):72–81, September 1994.

[13] Günther Görz. Kognitiv orientierte Architekturen für die Sprachverarbeitung. Technical Report ASL–TR–39–92/UER, Universität Erlangen-Nürnberg, 1992.

[14] Hermann Haken. *Synergetics, An Introduction.* Springer Series in Synergetics. Springer Verlag, 3rd edition, 1983.

[15] Andreas Hauenstein and Hans Weber. An investigation of tightly coupled time synchronous speech language interfaces using a unification grammar. Verbmobil Report 9, Universität Hamburg, NATS und Universität Erlangen-Nürnberg, IMMD VIII, February 1994.

[16] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, August 1978.

[17] INMOS Limited. *OCCAM 2 Reference Manual.* Prentice Hall, 1988.

[18] V. Jaganathan, Rajendra Dodhiawala, and Lawrence S. Baum, editors. *Blackboard Architectures and Applications.* Academic Press, Inc., 1989.

[19] D. Marr. *Vision. A Computationl Investigation into the Human Representation and Processing of Visual Information.* Freeman, 1982.

[20] Claudius Pyka. Deterministische, inkrementelle und zeitsynchrone Verarbeitung und die Architektur von ASL-Nord. Technical Report ASL–TR–22–91/UHH, Universität Hamburg, 1992.

[21] Claudius Pyka. Management of hypotheses in an integrated speech-language architecture. In *Proc. ECAI*, pages 558–560, 1992.

[22] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

[23] Wolfgang Wahlster and Judith Engelkamp, editors. *Wissenschatliche Ziele und Netzpläne für das VERBMOBIL-Projekt.* DFKI, Saarbrücken, 1992.

[24] Hans H. Weber. *LR-inkrementelles probabilistisches Chartparsing von Worthypothesenmengen mit Unifikationsgrammatiken: Eine enge Kopplung von Suche und Analyse.* PhD thesis, Submitted to Universität Hamburg, 1994.

[25] Robert C. Whitehair and Victor R. Lesser. A framework for the analysis of sophisticated control in interpretation systems. Technical Report 93–53, University of Massachusetts at Amherst, Amherst, MA, 1993.