Using a Distributed Frame System to Implement Distributed Problem Solvers

Ora Lassila & Seppo Törmä

Laboratory of Information Processing Science
Helsinki University of Technology
Otakaari 1, SF-02150 Espoo, FINLAND

Presented at the International Working Conference on Cooperating Knowledge Based Systems (CKBS-90), University of Keele, Keele (Great Britain), October 1990.

Abstract

This paper presents some preliminary results and ideas from a research project dealing with the implementation of distributed problem-solving systems. The paper discusses the possibility of implementing these systems using a distributed object system. The basic approach is to use the object-oriented paradigm to hide the physical environment: the universe of objects is distributed across the network of machines involved in the problem-solving process, allowing different problem solvers to access the same underlying knowledge about the problem without actually having to know about the distribution itself. The problem-solving agents, however, may still communicate with each other by sending explicit messages, acknowledging the distributed nature of the environment if necessary.

The software tools created and used in the research are presented: they are BEEF, a compact and powerful frame system, and BONE, its extension which allows frame-to-frame communication between frames in separate physical machines. The implementations of these systems are described.

TKO-B68
ISBN 951-22-0871-7
ISSN 0785-6601

TKK OFFSET 1991

1. Introduction*

The object-oriented paradigm has proved useful in the design and implementation of systems which perform concurrent computation or are otherwise distributed in nature. The concept of message passing is well suited to interprocess and intermachine communication. In recent years, many distributed object-oriented programming systems have been designed and implemented. Very often the object-oriented paradigm is used to hide the fact that processing has been distributed to many processes or many machines. Examples of such systems are Emerald [Black et al 87] and Guardian [Liskov & Scheifler 82].

Since their introduction in the 1970's, *frame systems* [Minsky 75, Fikes & Kehler 85] have gained ground as basic knowledge representation tools. The fundamental idea of a frame system is rather simple: a *frame* represents an object or a concept. The frame has a collection of attributes which may initially have default *values*. The values of attributes can later be altered to make the frame correspond to the particular situation at hand. Pragmatically frames are an extension of conventional record structures. However, frame systems also fit the common characterizations of object-oriented systems [e.g. Meyer 88, Pascoe 86]. A frame system allows the definition of *types* and the creation of *instances*. It also allows types to *inherit* descriptions of structure and behaviour from other types.

BOSS [Hynynen & Lassila 88, Hynynen 88, Hynynen & Lassila 89] is a knowledge-based distributed production scheduler employing a hierarchical framework for distributed problem solving. In BOSS, the problem-solving agents are organized into a hierarchy which offers possibilities to solve problems of varying granularity (in the particular case of BOSS, to create schedules of varying precision). The agents are threefold entities with components for problem solving, knowledge management and communication. This conceptualization is a fairly straight-forward way to represent a purposeful agent in a decentralized environment. A similar approach may be found in several modeling frameworks, e.g. [Sprague & Carlson 82, De Suranjan et al 85].

2. Organizing Problem Solvers

The basic idea of the BOSS scheduler is to organize the problem-solving (i.e. production scheduling) agents hierarchically to reflect the structure of the manufacturing organization. The agents are connected through superior-subordinate links – mediating authoritative communication of goal-setting and success-report messages – and through peer-to-peer links mediating negotiations.

The basic operation consists of problem decomposition by a superior agent, and the assignment of goals to subordinates. The subordinate agents, after having solved their respective sub-problems,

---

* This research has been sponsored in part by the Laboratory of Information Processing Science of the Computer Science Department at Helsinki University of Technology, the Technology Development Centre in Finland (TEKES), and the Rautaruukki corporation.

report to the superior. In case of minor conflicts, the subordinates can negotiate and solve the conflict by themselves. These principles have been illustrated in figure 1.

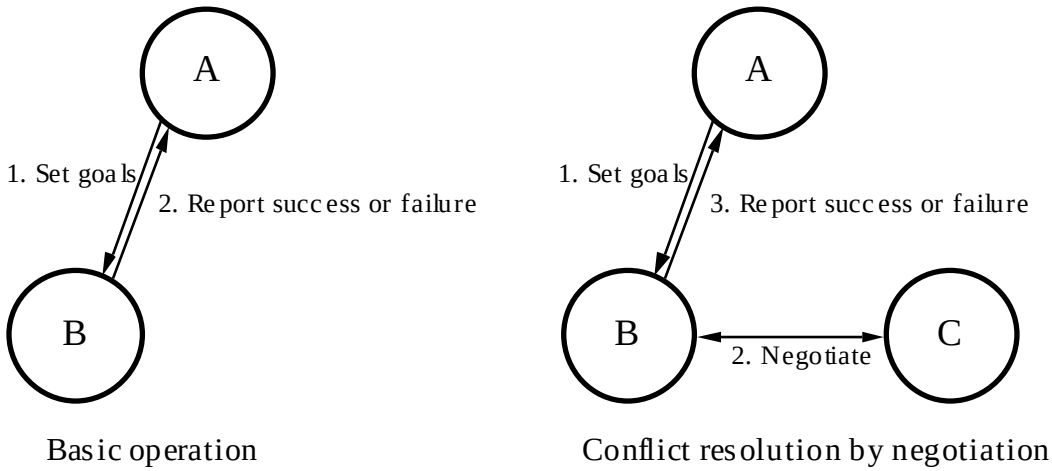

Figure 1: Basic problem-solving mechanisms in BOSS.

Each of the problem-solving agents is a separate computational process, typically in its own computer, and has its own universe of frames which represent the world as the agent sees it. In our case this is a model of a manufacturing system. To be able to collaborate, the agents also have to know something about other agents and their respective models. At least, a superior agent has to have knowledge about its subordinates. The problem of knowing about other agents and other models could be solved, if the different frame universes had an interface through which frames could be seen, with the communication between processes being handled automatically.

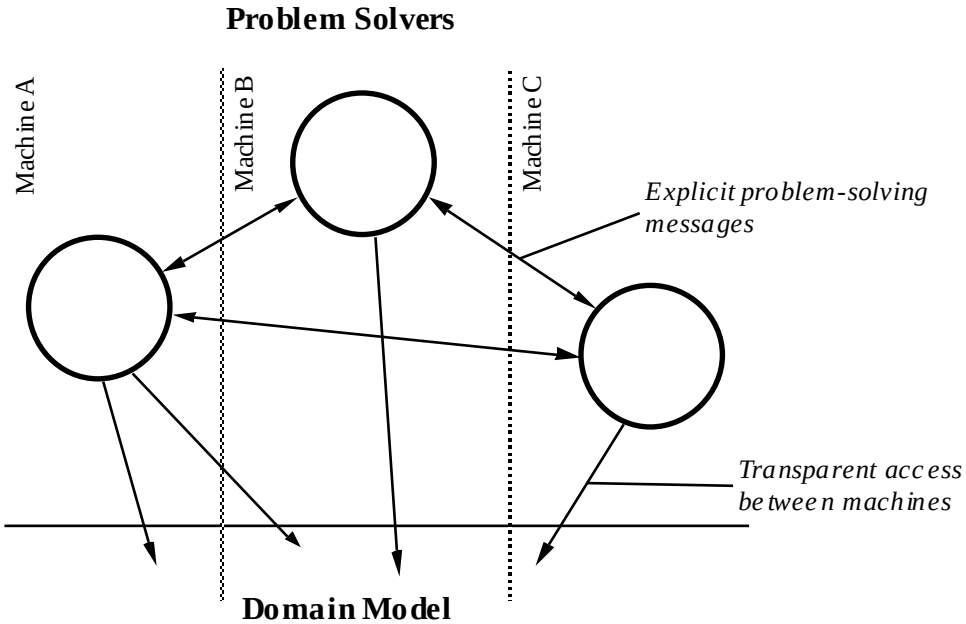

Figure 2: Problem-solving agents and communication between them.

The approach selected is based on the following principles:

Principle 1: The problem-solving agents communicate with each other by sending explicit *problem-solving messages*; to hide this communication would confuse the designer of the distributed problem solving system. Distributed and therefore parallel processing is fundamentally different from the traditional sequential way of programming, therefore it is unwise to forget it and pretend to be working in a non-distributed environment.

Principle 2: The object-oriented paradigm is used to hide the physical environment – the universe of objects is distributed across the network of machines involved in the problem-solving process, allowing different problem solvers to access the same underlying knowledge about the problem without actually having to know about the distribution itself.

Figure 2 symbolizes this approach: the problem solvers communicate using explicit messages, but see a common model of the problem domain.

3. Modeling and Distribution Tools

The object-oriented approach to the implementation of distributed problem solvers suggested in this paper has been experimented with using a frame system and its extension which supports distributed frame universes on a local-area network.

BEEF (BOSS's Extremely Elegant Frame System) [Lassila 90] is a frame system implemented using Common Lisp [Steele 84]. It was designed with the intention of bringing object-oriented and frame-based programming together. Initially, BEEF was used in porting the BOSS system from a Lisp Machine to a microcomputer. BOSS and its predecessor OPIS [Smith et al 86] were originally implemented using Common Lisp and CRL [Carnegie 86].

Object-oriented knowledge representation is often available in heavy knowledge-engineering tools. KEE [IntelliCorp 85] and Knowledge Craft [Carnegie 86] offer the programmer a rich selection of features, but their completeness can sometimes be viewed as their weakness: it takes a lot of memory and CPU cycles to run these tools, and this often degrades their expressiveness and power. BEEF's design was influenced by constrained computational resources of the target environment. As a tool for the adaptation of a large AI program into a microcomputer, BEEF itself couldn't be too large or use too much dynamic memory. BEEF's possible use as a portable object-oriented programming system also guided the design and implementation.

BEEF is based on a small set of concepts, the basic ones being *frames*, *slots* and *values*. Modeling and programming is quite possible with Common Lisp and these concepts alone. Since BEEF is an extension of Common Lisp, it was made "culturally compatible" with CL.

From the modeling point of view, *frames* are used as equivalents of real-world objects and concepts. They can be given named attributes, called *slots*. A slot can have any number of Common Lisp data objects as *values*. The notion of slots is extended by *relations*, which are expressions describing how different data are related with each other. The behaviour of relations is best understood if we view the

frame universe as a graph consisting of vertices (frames) and arcs (connections through slots). Relation expressions are used as patterns which match fragments of the graph.

A *world* is a snapshot of the frame universe. Initially, all actions take place in a so-called root world. A new world can be created as a child of an old world. It is initially similar to its parent, but changes made in it cannot be seen in the parent. When a child is deleted, all changes made in it are undone. Changes can also be copied upward to the parent.

Programs can be organized by making frames communicate through *message passing*. A frame receiving a message needs to be able to handle it. This is done by *message handlers* attached to the frames. Handlers are Lisp functions and correspond to what is usually called *methods*. Our experience with CRL showed that a frame system offering a rudimentary provision for frame methods forces the programmer to use a very tedious programming style. CRL only allows separately defined functions to be assigned as slot values, and a clumsy way of calling these.

BEEF provides a simple efficient way of writing frame methods, facilitating object-oriented programming in practice, and not just theoretically. A special method definition form (á la defun) invisibly enforces the parameter protocol and also makes it easy to access the slots of the method's receiver. Inside the method's body, the receiver frame's slots can be accessed as local variables. The method definition mechanism resembles that of CLOS [Bobrow et al 88].

Updates and accesses of slot values can cause the firing of *daemons*. These are methods which can, for example, construct the slot's value when accessed, or modify the values being stored.

BONE (BEEF's Official Network Extension) is a system which allows BEEF frames to reside in separate machines, yet communicate by passing *low-level messages*. One should not confuse the concepts *low-level message* and *problem-solving message* – the former is an implementation-level entity, the latter an abstract notion of the problem-solving mechanism. The term *message* is subsequently used to denote low-level messages.

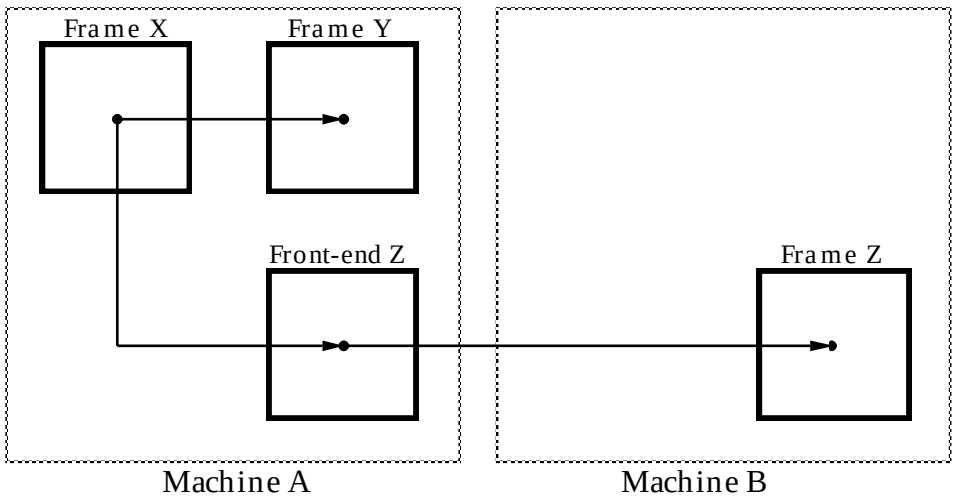

Figure 3: Local and remote message passing in BONE.

3.1. Automatic Communication

A frame wishing to communicate with other frames in a remote machine creates a *front-end* into that machine. Remote frames then communicate with the front-end which mediates all communication to the actual owner of the front-end (also called a *back-end*). In a way, the front-end is a perfect image of its back-end, so the other frames need not know that the actual information is mediated to (and fetched from) another computer. This mechanism is the basic vehicle for implementing the principle 2.

The mechanism of front-end creation is shown in figure 4. A frame – a back-end, that is – sends a *front-end creation message* to another machine. The processing of the message at the receiving end causes a front-end to be instantiated. The front- and back-ends have the same names. Physical machines are modeled using instances of the frame class host-machine which contain all relevant information about communicating (network addresses etc.). The host-machine information is duplicated in each machine. The implementation of the distributed version of BEEF is currently based on the TCP protocol. The implementation is based on the Macintosh Allegro Common Lisp and the MacTCP software tools.

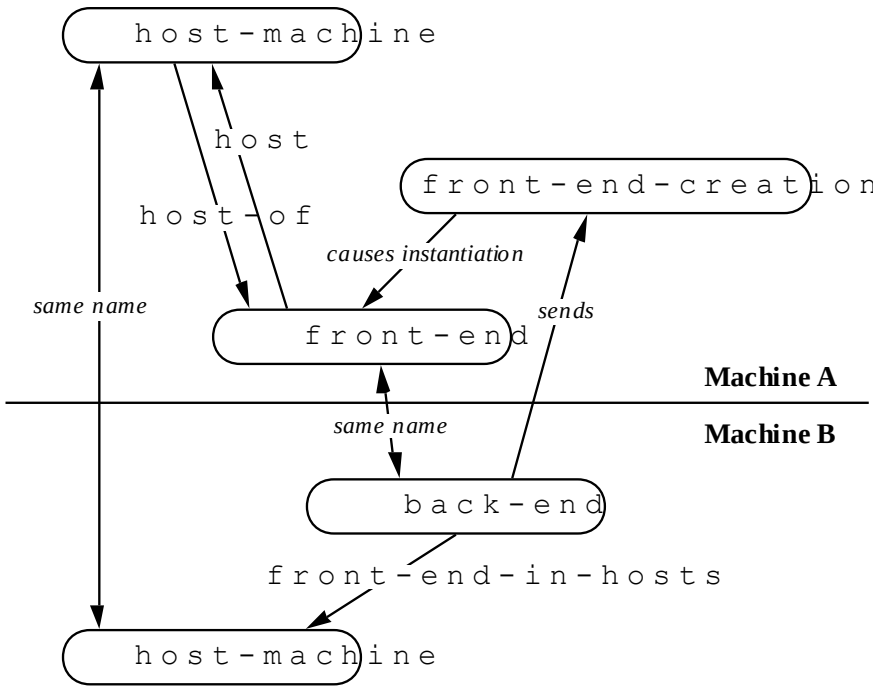

Figure 4: Front-end creation in BONE.

Basically the front-end/back-end mechanism necessitates the use of a well defined method call interface between different frames. In addition, using the daemon mechanism of BEEF, slot values can be updated from a back-end to a front-end. In general, however, the front-ends cannot be made exact copies of their back-ends, because the local frame universe seen by a particular front-end might not have front-ends for all those frames seen by the corresponding back-end.

3.2. Low-Level Implementation

The implementation of low-level messages is based on a remote procedure call facility. In a programming environment based on side-effects, the program has to be aware of the explicit flow of control. It is therefore useful to have different kinds of remote calls. We provide three:

- A procedure call can be sent to a remote machine and executed asynchronously. No results are sent back. The caller is not blocked.
- A function call can be sent to a remote machine, effectively blocking the caller. Results are returned.
- As a combination of the above two, a function call can be sent to a remote machine, but the caller is not blocked. Instead, a *future object* is created [Caromel 88]. If the future's value is accessed, the accessor is blocked if the remote call has not returned.

On the lowest level, the calls are communicated as text strings, which are read back to the receiving Common Lisp system. This is possible, because Common Lisp has a strict equivalence between objects and their printed representation. The BEEF system customizes the Common Lisp reader to handle the printed representation of frame instances.

4. Conclusions

The described approach is currently being applied to the development of DOM, a distributed knowledge-based production scheduler [Lassila et al 90]. The initial experiences support the idea of the dual nature of communication. It has been found out, however, that sometimes one has to acknowledge the distributed nature of the underlying model, and explicitly manipulate front- and back-ends. Another unsolved problem is the possibility of communication optimization through some kind of caching.

Acknowledgements

The authors wish to thank Dr Juha Hynynen and professor Markku Syrjänen for fruitful discussions and invaluable help before and during this research project.

References

[Black et al 87] Black Andrew, Norman Hutchinson, Eric Jul, Henry Levy and Larry Carter, *Distribution and Abstract Types in Emerald*. IEEE Transactions on Software Engineering, pp.65-76, vol SE-13, no.1, January 1987.

[Bobrow et al 88] Bobrow Daniel G, Linda G. DeMichiel, Richard P. Gabriel, Sonya E. Keene, Gregor Kiczales and David A. Moon, *Common Lisp Object System Specification*, X3J13 Document 88-002R, SIGPLAN Notices, volume 23, Association for Computing Machinery, September 1988.

[Carnegie 86] Carnegie Group, *Knowledge Craft User's Manual*. Carnegie Group, Inc., Pittsburgh (Pennsylvania, U.S.A.), 1986.

[Caromel 88] Caromel Denis, *A General Model for Concurrent and Distributed Object-Oriented Programming*. In Gul Agha, Peter Wegner and Akinori Yonezawa (eds.), *Proceedings of the ACM SIGPLAN Workshop on Object-Based Concurrent Programming*. SIGPLAN Notices, volume 24, number 4, Association for Computing Machinery, April 1988.

[De Suranjan et al 85] De Suranjan, Shimon Y. Nof, and Andrew B. Whinston, *Decision Support in Computer-Integrated Manufacturing*. Decision Support Systems 1(1985)1, pp. 37-56.

[Fikes & Kehler 85] Fikes Richard and Tom Kehler, *The Role of Frame-Based Representation in Reasoning*. Communications of the ACM, 28:9, Association for Computing Machinery, September 1985.

[Hynynen & Lassila 88] Hynynen Juha and Ora Lassila, *BOSS: A Knowledge-Based System for Distributed Production Management* (in Finnish). In: Matti Mäkelä, Seppo Linnainmaa, and Esko Ukkonen (eds), Proceedings of the Finnish Artificial Intelligence Symposium (STeP'88), pp. 219-230, Helsinki (Finland), August, 1988.

[Hynynen & Lassila 89] Hynynen Juha and Ora Lassila, *On the Use of Object-Oriented Paradigm in a Distributed Problem Solver*. In: Hannu Jaakkola and Seppo Linnainmaa (eds), Proceedings of the Second Scandinavian Conference on Artificial Intelligence (SCAI'89), pp. 657-669, Tampere (Finland), June 1989.

[Hynynen 88] Hynynen Juha, *A Framework for Coordination in Distributed Production Management*. Doctoral dissertation, Laboratory of Information Processing Science, Helsinki University of Technology, Otaniemi (Finland), November, 1988.

[IntelliCorp 85] IntelliCorp, *KEE™ Software Development System User's Manual*. IntelliCorp, Mountain View, California (U.S.A.), July 1985.

[Lassila 90] Lassila Ora, *The BEEF Reference Manual - A Programmer's Guide to the BEEF Frame System*. Report TKO-C43, Laboratory of Information Processing Science, Helsinki University of Technology, Otaniemi (Finland), February 1990.

[Lassila et al 90] Lassila Ora, Markku Syrjänen and Seppo Törmä, *Coordinating Mutually Dependent Decisions in a Distributed Scheduler*. Presented at the International Conference on Advances in Production Management Systems (APMS'90), Otaniemi (Finland), August 1990.

[Liskov & Scheifler 82] Liskov Barbara and Robert Scheifler, *Guardians and Actions: Linguistic Support for Robust, Distributed Programs*. In: Proceedings of the Ninth ACM Symposium on the Principles of Programming Languages, Association for Computing Machinery, 1982.

[Meyer 88] Meyer Bertrand, *Object-oriented Software Construction*. Prentice Hall International Series in Computer Science, Great Britain, 1988.

[Minsky 75] Minsky Marvin, *A Framework for Representing Knowledge*. In: Patrick Henry Winston (ed.), The Psychology of Computer Vision. McGraw-Hill, New York (U.S.A.), 1975.

[Pascoe 86] Pascoe Geoffrey A, *Elements of Object-Oriented Programming*. BYTE, August 1986, pp.139-144.

[Smith et al 86] Smith Stephen F., Mark S. Fox ja Peng Si Ow, *Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling*, The AI Magazine, Vol. 7, No. 4, ss. 45-61, The American Association for Artificial Intelligence (AAAI), Fall, 1986.

[Sprague & Carlson 82] Sprague Ralph H. Jr and Eric D. Carlson, *Building Effective Decision Support Systems*. Englewood Cliffs (New Jersey, U.S.A.), 1982, Prentice Hall, Inc.

[Steele 84] Steele Guy L. Jr., *Common LISP: The Language* (first edition). Digital Press, Digital Equipment Corporation, U.S.A., 1984.