Distributed Modal Theorem Proving with KE

Jeremy Pitt[1] and Jim Cunningham[2]

[1] Intelligent Communication Systems, Department of Electrical & Electronic Engineering, Imperial College of Science, Technology, and Medicine, Exhibition Road, London, SW7 2BT, UK.
Email: j.pitt@ic.ac.uk URL: http://www.ics.ee.ic.ac.uk/~jvp/

[2] Department of Computing, Imperial College of Science, Technology, and Medicine, Huxley Building, 180, Queen's Gate, London, SW7 2BZ, UK.
Email: rjc@doc.ic.ac.uk URL: http://medlar.doc.ic.ac.uk/~rjc/

Abstract. This paper describes an approach to distributed modal theorem proving by bringing together and exploiting two software packages. The first is the implementation of a theorem prover for normal modal logics based on **KE** and a generalization of Fitting's prefixed tableaux. The second is a library for implementing brokered inter-process communication over the internet. We describe three demonstrators which combine these implementations and illustrate potential applications of the new technology, enabling theorem provers connected to the internet to cooperate, compete, or be used by third parties.

1 Introduction

This paper describes an approach to distributed modal theorem proving. This is based on two software implementations. The first is $\Box$**KE**, an extension of the first-order theorem prover lean**KE** [23] to include modal rules for theorem proving in normal modal logics. The second is the CSF (Cooperation Services Framework) [21], a library for providing and invoking services between autonomous software processes executing in a distributed open computing environment.

lean**KE** is a sound and complete theorem prover for first-order classical logic, which is based on the calculus **KE** [11] and free variable rules for eliminating quantifiers. It has been implemented in ICL/ECRC's ECL^{i}PS^e Prolog [12], and in [23] its performance is evaluated on some standard problems and compared to the tableau theorem prover leanT^AP [3]. As part of the CEC Medlar II project (Esprit 6471) we have investigated extending lean**KE** to handle normal modal logics.

The CSF was first investigated in the CEC GOAL project (Esprit 6283), and was designed to provide distributed services to autonomous software processes running in a federated information system, based on normalized, brokered, interactions between them. It has been further developed in the Medlar II project, where it was proposed as a means to integrate and combine specialised inference engines [22], and to exploit a common recent practice of making automated reasoning systems available through the WWW, e.g. SCAN [19] and the Logics Workbench [18].

The Logics Workbench is an integrated, interactive system which provides general reasoning facilities in a wide variety of propositional logics. Indeed, considerable research effort has been devoted to developing general proof systems for *families* of non-classical logics: see, for example, [10] on substructural logics, [14] on modal logics, and [15] on "any" logic. We have investigated a generalization of Fitting's prefixed tableaux of [14] for the family of normal modal logics, and have implemented the system $\Box$**KE**, an extension of the propositional core of leanKE to include rules for the modal operators $\Box$ and $\Diamond$. For each normal modal logic in $\Box$**KE**, the form of the modal rules remains the same but the side-conditions on each rule (and on closure) can vary.

In this paper, we bring together the themes of distributed computing and modal theorem proving. We describe a series of demonstrative applications in which it is possible to launch separate $\Box$**KE** processes, each configured to handle a certain modal logic or logics. Using the CSF, these processes can then be coupled in such a way that they can cooperate or compete, can be sited centrally or distributedly, and can be accessed by other users or applications via the internet.

In section 2 we motivate and summarize our approach to prefixed **KE** for normal modal logics, give a specification of $\Box$**KE**, describe its implementation, and present some preliminary results. In section 3 we focus on the important features of the CSF for inter-process communication and cooperation. We then draw the two themes of the paper together in section 4, where we implement three demonstrators which illustrate the potential of $\Box$**KE** and the CSF for distributed modal theorem proving. Some final comments and concluding remarks are made in section 5.

2 Modal KE

2.1 Fitting's Prefixed Tableaux

In automated reasoning with modal logics, the idea of using a prefix, or label, to denote either a world and/or a path to a world, can be found in the works of for example [24, 6, 15], amongst others. The research reported here builds primarily on that of [14], but seeks to redress certain inefficiencies.

In Fitting's prefixed tableaux for modal logics [14], the standard tableau machinery is extended with names for possible worlds, which are prefixes, and which are used in such a way that the accessibility relation between possible worlds is reflected in the syntactic features of those names. If σ is a prefix and X is a formula, then by the prefixed formula $\sigma : X$ we mean that X is true in the world named by σ. A prefix in [14] is simply a non-empty finite sequence of positive integers. This choice of representation allows syntactic recognition of whether one world is accessible from another or not, given the accessibility relation between worlds defined in a frame for a particular logic. For example, in **K**, the world named by the prefix $\langle 1,2,1,4\rangle$ is accessible from $\langle 1,2,1\rangle$. (In fact, in any normal modal logic, given any prefix σ and any integer n, the prefix formed by appending n to σ, σn, is always accessible from σ.)

Other conditions on accessibility between prefixes are given below, which is taken from [14], extended with condition 5:

1. the *general* condition if σi is accessible from σ for every i;
2. the *reverse* (or *symmetric*) condition if σ is accessible from σi for every i;
3. the *identity* (or *reflexive*) condition if σ is accessible from itself;
4. the *transitive* condition if τ is accessible from σ whenever σ is a proper initial segment of τ;
5. the *euclidean* condition if $\sigma i\tau$ is accessible from σj for every i except j;
6. the *universal* condition if every prefix is accessible from any prefix.

The modal rules used to extend the usual tableau rules, for the logics that Fitting considered, can be stated as follows:

Rule $\neg\Box$

$$\frac{\sigma : \neg\Box a}{\sigma i : \neg a}$$ provided σi is a simple unrestricted extension of σ, for any i, for any logic

Rule $\Box$

$$\frac{\sigma : \Box a}{\tau : a}$$ provided either (a) in **K**, **KB**, and **K4**, τ is accessible and available; or (b) in **D**, **KT**, **KDB**, **B**, **KD4**, **S4** and **S5**, τ is available, accessible, or is a simple unrestricted extension of σ.

In Fitting's system, *accessible* means that the prefix satisfies the accessibility conditions for the logic (so accessibility varies with the logic), *available* means that the prefix has been used on the branch of the tableau, and a *simple unrestricted extension* of a prefix is one where a single integer has been added, and the result is not a sub-sequence of any prefix that is available on the branch.

2.2 Generalized Prefixes

The Fitting notation is elegant and general, and our first experiment was to extend leanKE with Fitting's rules. However, a number of problems were encountered:

- the search space could become very large quite quickly. Using Fitting's rules, the number of options of which world to 'name', whenever the $\Box$ rule was applied, could increase rapidly. Not all choices would lead to a closed branch, so the alternatives had to be explored by backtracking;
- the proof was dependent on the order of application of rules. In some cases, it was necessary to use a $\neg\Box$ rule to 'name' a specific world for a subsequent application of the $\Box$ rule a world to be successful. This $\neg\Box$ rule had to be applied first, otherwise the program would need to backtrack, and then redo work that had already been done once;
- the local and global rules opened up the possibility of infinite loops unless their use was controlled (e.g. by user interaction) or some mechanism for loop checking was introduced;
- to be able to use **KE**'s β rule and closure rule, it was necessary to have the identical prefix on both formulas. Again, this could lead to unnecessarily reduplicated effort;
- different prefixes could actually name the same world. If we wanted to build a model, we then had the problem of collecting the formulas in the world that constitutes the model. This could be difficult if these formulas were scattered across several partial descriptions of the model, each of which had a different name.

In particular, the need to use $\neg\Box$ rules before $\Box$ rules is similar to the problem in theorem proving in classical logic, where, as far as possible, one applies the existential rules before the universal ones. The mechanical solution is to introduce free variables and unification, and we adopt a similar strategy for modal logic. We generalize the notion of a prefix from a non-empty sequence of positive integers to a possibly empty sequence of positive integers and *variables*. Each variable is constrained to take a value from a particular domain, which can be one of:

$$N^1 = \{1, 2, 3, \ldots\}$$
$$N^\bullet = \{\bullet, 1, 2, 3, \ldots\}$$
$$N^+ = \text{sequences of one or more elements of } N^1$$
$$N^* = \text{sequences of zero or more elements of } N^1$$

We use i and j as metavariables over the domains $N^\bullet$ and N^1, and σ and τ as metavariables over the domains N^* and N^+. Then, for example, $\bullet$, $\langle 1, 1\rangle$, $\langle 1, 2, 3, 4, i : N^\bullet\rangle$, and $\langle i : N^1, 1, j : N^+\rangle$ are all valid prefixes; in particular $\bullet$ denotes the empty sequence and on its own is the prefix which names the 'real world'. Note that $\langle 1, \bullet, 2\rangle$ is the same prefix as $\langle 1, 2\rangle$.

This generalization allows us to be non-committal over suitable world(s) that we are naming when we apply the $\Box$-rule for a particular logic, because we introduce a variable whose domain is determined by that logic. Then the order of the application of $\Box$ and $\neg\Box$ rules is no longer important, and the number of alternative worlds to go to is substantially reduced. We can also show that ostensibly different prefixes – i.e. prefixes which contain variables – are, in fact, the same world if these prefixes can be unified.

Our prefix unification algorithm is completely specified in [9], but full details are omitted here because of space constraints. However, this is essentially a specification for a procedure υ that takes as input two prefixes, matches common sub-sequences, and returns a substitution θ if the unification of the non-common sub-sequences *succeeds*, and *fails* otherwise, i.e.:

$$\upsilon(\sigma, \tau) = \begin{cases} \theta & \text{if } \sigma \text{ and } \tau \text{ unify} \\ \textbf{fail} & \text{otherwise} \end{cases}$$

The usual notions of a substitution being an assignment of values to variables, and a unifier being such a substitution which makes two prefixes 'equal', are assumed.

Note also that the procedure may succeed more than once, and return a different value for θ. For example, $\langle 1, i : N^*\rangle$ and $\langle j : N^*, 1\rangle$ returns as unifiers $\langle 1\rangle$ (with substitution $i = \bullet, j = \bullet$) and $\langle 1, k : N^*, 1\rangle$ (with substitution $i = \langle k : N^*, 1\rangle$, $j = \langle 1, k : N^*\rangle$). Thus there may be no most general unifier, as $\langle 1\rangle$ and $\langle 1, k : N^*, 1\rangle$ do *not* unify.

2.3 Modal Rules for KE

For each of the distinct normal modal logics, we can identify the conditions satisfied by the accessibility relation on prefixes, according to the properties of that logic's accessibility relation in its models. This is summarised in Table 1, which is an extension of the table given in Fitting [14] to include the normal modal logics **K5**, **K45**, **KD5**, **KD45** and **KB4**. (Note that the **KE** calculus includes bivalence in the rule *PB*.)

Table 1. Accessibility Conditions For Normal Modal Logics

Condition	**K**	**D**	**KT**	**KB**	**K4**	**K5**	**S4**	**B**	**KDB**	**KD4**	**KD5**	**K45**	**KD45**	**KB4**	**S5**
General	•	•	•	•	•	•	•	•	•	•	•	•	•	•	
Reverse				•				•	•					•	
Identity			•				•	•							
Transitive					•		•			•		•	•	•	
Euclidean						•					•	•	•	•	
Universal															•

Using Table 1 and the different domains for variables, we can now state the $\Box$ rule for each modal logic listed in that table. These are shown in Table 2[3].

[3] We assume that the effect of certain operations on prefixes, like *head* and *last*, are obvious.

The $\Diamond$-rule (i.e. the $\neg\Box$-rule) and closure rule are respectively:

$$\frac{\sigma : \neg\Box a}{\sigma x : \neg a}\,[L] \quad \begin{array}{l}\text{if } L \text{ is not } \mathbf{K4} \text{ or } \mathbf{K45}\text{, then } x \text{ is a new ground integer,}\\ \text{otherwise, if } \mathit{last}(\sigma) \text{ is } \bullet \text{ or a ground integer, then } x \text{ is a new ground integer;}\\ \text{otherwise, } x = \langle i, y : N^* \rangle \text{ and } i \text{ is a new ground integer}\end{array}$$

$$\frac{\sigma : a \quad \tau : \neg a}{\times}\,[L] \quad \begin{array}{l}\text{if } L \text{ is } \mathbf{K}, \mathbf{KB}, \mathbf{K4}, \mathbf{K5}, \mathbf{K45}, \text{ or } \mathbf{KB4}\text{, then } \upsilon(\sigma,\tau) = \theta,\\ \qquad\qquad\qquad\text{(provided } \sigma(\theta) \text{ can be ground on the branch),}\\ \text{otherwise, } \upsilon(\sigma,\tau) = \theta\end{array}$$

The extra condition on the unifier grounding the prefix on those logics without the D axiom is to ensure that the named world actually exists in the model.

Note on K4 and K45. The logics **K4** and **K45** cause a certain amount of problems because of the 4 axiom allowing boxed formulas to be copied across to the newly accessible world, where they might be needed to 'generate' new worlds. This does not appear to affect other logics which have the 4 axiom but not the D axiom if we loosen the restriction on unifying common sub-paths.

For the two logics **K4** and **K45**, however, we have currently adopted a solution motivated more from an implementation perspective rather than a logical one, and this is to make the extension of the prefix dependent on the value of the last element of the prefix. The motivation is that, with a formula of the form $\sigma : \Box\Diamond p$, we can expand a modal **KE** tree as follows, where i is a single ground integer:

$$\begin{array}{ll}\sigma : \Box\Diamond p & \\ \sigma\tau : \Diamond p & (\Box \text{ rule})\\ \sigma\tau i : p & (\Diamond \text{ rule})\end{array}$$

However, allowing for the 4 axiom, we could also extend the branch above by adding formulas shown below, where j is a single ground integer different from i:

$$\begin{array}{ll}\sigma : \Box\Box\Diamond p & (4 \text{ axiom})\\ \sigma\tau : \Box\Diamond & (\Box \text{ rule})\\ \sigma\tau i : \Diamond p & (\Box \text{ rule})\\ \sigma\tau i j : p & (\Diamond \text{ rule})\end{array}$$

Therefore having got a ground world after a box operator one can always 'slide along' to another ground world because the boxed formula can always be copied over to the first ground world and used to name another world.

We are seeking a more general expression of this fact in keeping with the spirit of the overall exposition rather than 'testing' the last element of the prefix, but this solution seems efficacious.

2.4 Implementation

In [23], we give a formal specification of first-order free variable **KE** which was used for the implementation of leanKE. In table 3 we give a similar but simpler specification[4] of propositional modal **KE** which is used for the current implementation of $\Box$**KE**. Here a branch is represented as a 2-tuple (Δ,Λ), where Δ is the set of unanalysed formulas, and

[4] We only give *instances* of the α, β, and η rules, and of *PB*: we assume that the other instances are easily recognized. A more general and more readily mechanisable expression is of course possible (see [23]), but needs more notation which is not required for present purposes.

Table 2. $\Box$-rule for each Logic of Table 1

$\dfrac{\sigma : \Box a}{\sigma x : a}$ [**K**]	$x : N^1$
$\dfrac{\sigma : \Box a}{\sigma x : a}$ [**D**]	$x : N^1$
$\dfrac{\sigma : \Box a}{\sigma x : a}$ [**KT**]	$x : N^0$
$\dfrac{\sigma i : \Box a}{\sigma x : a}$ [**KB**]	$i \in N^1 \wedge x = \bullet$, or $i = \bullet \wedge x : N^1$
$\dfrac{\sigma : \Box a}{\sigma x : a}$ [**K4**]	$x : N^+$, if $\mathit{last}(\sigma) = \bullet, \in N^1$, or $\in N^+$ $x = \langle i, y : N^* \rangle$, for new ground integer i, otherwise
$\dfrac{\sigma i : \Box a}{\sigma x : a}$ [**K5**]	$i \in N^1 \wedge x : N^+ \wedge i \neq \mathit{head}(x)$
$\dfrac{\sigma : \Box a}{\sigma x : a}$ [**S4**]	$x : N^*$
$\dfrac{\sigma i : \Box a}{\sigma x : a}$ [**B**]	$i \in N^1 \wedge x = \bullet$, or $i = \bullet \wedge x : N^0$
$\dfrac{\sigma i : \Box a}{\sigma x : a}$ [**KDB**]	$i \in N^1 \wedge x = \bullet$, or $i = \bullet \wedge x : N^1$
$\dfrac{\sigma : \Box a}{\sigma x : a}$ [**KD4**]	$x : N^+$
$\dfrac{\sigma i : \Box a}{\sigma x : a}$ [**KD5**]	$i \in N^1 \wedge x : N^+ \wedge i \neq \mathit{head}(x)$, or $i = \bullet \wedge x : N^1$
$\dfrac{\sigma i : \Box a}{\sigma x : a}$ [**K45**]	$i = \bullet \wedge (x : N^+$, if $\mathit{last}(\sigma) = \bullet, \in N^1$, or $\in N^+$ $x = \langle i, y : N^* \rangle$, for new ground integer i, otherwise$)$, or $i \in N^1 \wedge x : N^+ \wedge i \neq \mathit{head}(x)$
$\dfrac{\sigma i : \Box a}{\sigma x : a}$ [**KD45**]	$i \in N^1 \wedge x : N^+ \wedge i \neq \mathit{head}(x)$, or $i = \bullet \wedge x : N^+$
$\dfrac{\sigma i : \Box a}{\sigma x : a}$ [**KB4**]	$i = \bullet \wedge x : N^+$, or $i \in N^1 \wedge x : N^+ \wedge i \neq \mathit{head}(x)$, or $(i \in N^1 \vee i : N^1) \wedge x = \bullet$
$\dfrac{\sigma : \Box a}{x : a}$ [**S5**]	$x : N^*$

Λ is the set of analysed formulas. Each of the rules is parameterised for a normal modal logic L, although this is only significant in three rules ($\Box$, $\Diamond$, and closure); note also that the β and η rules can only be applied if the labels on the formulas can be unified. The substitution assigning values to prefix variables is 'global' to a branch, is assumed to be automatically updated by a successful unification, and is denoted by ϕ. In a Prolog implementation, of course, we get this for 'free'.

The algorithm given in [23] is simply adapted by the addition of the $\Box$ and $\Diamond$ rules, so that the existing implementation of lean**KE** is barely perturbed: the first order rules are commented out and the $\Box$ and $\Diamond$ rules added in. The prefix unification algorithm is implemented in ECL^{i}PS^e, and this gives the complete implementation of $\Box$**KE**.

We do not have formal soundness and completeness arguments as yet (this is being worked on), but there is empirical evidence which indicates that the system works. For example, each logic only proves those axioms that it should (i.e. **S4** proves K, D, T, and 4; **DB** proves K, D and B; **S5** proves them all, etc.); no logic proves the negation of any axiom; and some of the textbook problems in [7] have been solved. More rigorously, we have also tested the theorem prover on the 31 examples of Catach in each normal modal logic. Where the logics are the same as Catach implemented (**K**, **D**, **K4**, **KD4**, **DB**, **KD5**, **KT**, **S4**, **KTB**, **KD45** and **S5**) the output correlates. Although one cannot really compare runtimes as the two systems are implemented on different hardware and software platforms, the raw comparison is favourable to $\Box$**KE**, namely ~23 seconds for 465 attempted proofs, as opposed to Catach's reported ~42 seconds for 496 proofs. These are simple examples, but there is also some indication of where there may be problems in proof search.

A fuller exposition of all the work on $\Box$**KE** can be found in [9].

Table 3. Specification for $\Box$**KE**

move literal

$$\frac{\Delta \cup \{\sigma : l\}, \Lambda}{\Delta, \Lambda \cup \{\sigma : l\}}$$

α **rule**

$$\frac{\Delta \cup \{\sigma : p \wedge q\}, \Lambda}{\Delta \cup \{\sigma : p, \sigma : q\}, \Lambda \cup \{\sigma : p \wedge q\}}$$

β **rule**

$$\frac{\Delta \cup \{\sigma : p \rightarrow q\}, \Lambda}{\Delta \cup \{\sigma : q\}, \Lambda \cup \{\sigma : p \rightarrow q\}}$$

if $\tau : p \in \Delta \cup \Lambda$, and $\upsilon(\sigma, \tau) = \theta$

η **rule**

$$\frac{\Delta \cup \{\sigma : p \leftrightarrow q\}, \Lambda}{\Delta \cup \{\sigma : q\}, \Lambda \cup \{\sigma : p \leftrightarrow q\}}$$

if $\tau : p \in \Delta \cup \Lambda$, and $\upsilon(\sigma, \tau) = \theta$

closure

$$\frac{\Delta, \Lambda \cup \{\sigma : p, \tau : \neg p\}}{\times}\ L$$

if $\upsilon(\sigma, \tau) = \theta$, and $\sigma(\phi)$ can be ground on the branch, if L is **K**, **K4**, **KB**, **K5**, **K45** or **KB4**

$\Box$ **rule**

$$\frac{\Delta \cup \{\sigma : \Box p\}, \Lambda}{\Delta \cup \{\tau : p\}, \Lambda \cup \{\sigma : \Box p\}}\ L$$

if τ comes from σ according to the constraints on the $\Box$ rule for logic L

$\Diamond$ **rule**

$$\frac{\Delta \cup \{\sigma : \Diamond p\}, \Lambda}{\Delta \cup \{\tau : p\}, \Lambda \cup \{\sigma : \Diamond p\}}\ L$$

if τ comes from σ according to the constraints on the $\neg\Box$ rule for logic L

PB

$$\frac{\Delta, \Lambda}{\Delta \cup \{\sigma : p\}, \Lambda \mid \Delta \cup \{\sigma : \neg p\}, \Lambda}$$

if $\sigma : p \vee q \in \Delta$

3 The Cooperation Services Framework

We now turn our attention from the implementation of a theorem prover for normal modal logics, to the implementation of a framework which enables (copies of) this theorem prover, executing as separate processes, possibly on separate machines, to interact.

This framework is based on the CSF, which was inspired by both CORBA (Common Object Request Broker Architecture), the OMG's (Object Management Group) specification and architecture for distributed object-oriented computing [20], and the ANSA model for Trading and Federation [2]. CORBA is designed to address the problems of getting networked applications to work together and to share information, i.e. to bind together different applications running on different computers connected by one or more types of network. This is achieved by a separation and generalization of clients and servers in the standard client–server model: separation because operations are defined by a common interface, communication is restricted to requests, and requests and results are routed through an *object request broker*; and generalization because clients can also be servers, clients and servers can come and go dynamically, and communication can be asynchronous as well as synchronous. In the ANSA model, there is a further generalization, as a client can use a *trader* to locate a server by function rather than by name.

The CSF was originally developed as part of the CEC GOAL project (Esprit 6471), whose aim was to develop project management tools for enterprises in which participating partners were distributed, heterogeneous and autonomous. The CSF was proposed in order to provide the functionality of CORBA, but also extend it so that clients and servers shared a common ontology so that services offered and services requested could be matched correctly (cf. [13]), and clients could get the most favourable service by offering work for tender or negotiating over the terms and conditions for satisfying service requests. It was also desirable for users to be able to monitor the interactions of (otherwise autonomous) clients and servers. Thus three of the more important features of the CSF are: *conversations*, which are managed by a broker, which enable a client process to locate and invoke services provided by servers registered with the broker; *protocols*, which allow two processes to negotiate over the terms and conditions of providing a service, for example; and *visualization*, which allows a user to review or monitor a conversation being conducted by two processes. This section reviews each of these features in turn.

3.1 Brokered Conversations

The CSF uses brokers to enable clients to find servers. A broker is assigned a 'dedicated' socket (which should be reserved by the operating system, in the same way that port numbers are standardised for particular internet protocols, e.g. http uses port 80, gopher uses port 70, etc.). A server implementation then only needs only know this one address in order to 'publicise' its services; a client need only know this address in order to use the broker to find a server which might be running on any machine connected to the internet.

The CSF allows two types of brokered conversation to be established. The first is via indirect communication, where all messages from client to server (and vice versa) are sent to the broker and which then forwards them to the appropriate destination. The second type is direct communication, where the client uses the broker to find a server, but then in the resulting conversation all messages are passed directly between the client and server.

Each type of communication, either direct or indirect, can be conducted synchronously or asynchronously. This enables a software developer to use the CSF to implement four styles of communication. *Synchronous* communication is when one process sends a message to another and then waits, doing nothing, until a reply is received. *Synchronous deferred* communication is when one process sends a message to another, and then continues working, checking for the reply at some later time. *Asynchronous* communication is when one process sends a message to another, continues working, but *interrupts* whatever it is doing whenever a new message arrives – this may or may not be the reply to the message sent out. Finally, *asynchronous deferred* communication is when a process sends a message, continues working, checking for new messages at some later time. The reply may or may not be included amongst whatever new messages have been received.

3.2 Cooperation Protocols

Communication between processes in the CSF is not just restricted to requests. In fact, communication between a client and server can go through three phases: service request, negotiation over terms and conditions using *cooperation protocols* [5], and service invocation. Thus the CSF can provide three enhancements to so-called object-brokering capabilities, namely standard interface operation for clients and servers through which negotiation and service invocation is conducted; expected behaviour for clients and servers, i.e. protocols for inter-process cooperation; and common services which enable clients, servers and brokers to manage conversations and verify the compliance of other processes with an agreed cooperation protocol. More concretely, we define:

Message: Processes communicate by message passing. A message is therefore the basic unit of interaction between two processes. Message passing is performed as an action (i.e. a "speech act"), which determines the type of message, its contents, and between whom the message is passing. We refer to a speech act as a *message action* to distinguish between message passing and other actions that a process can perform;

Protocol: A protocol is the formal definition of a structured exchange of messages which constitute a declarative agreement for communicative behaviour. For each type of message, its *protocol specification* determines the obliged actions, including the space of possible responses. More complex protocols can be defined using the given protocol specifications. A protocol is refined at a lower level of specification to define a mutually understood procedure for performing a particular service, task or activity.

Conversation: A conversation is a protocol in execution, and is composed of a sequence of messages which comply with the protocol definition. The current state and history of a conversation is maintained independently by all participating processes, including the broker.

Full details can be found in [21]. The message space of a simple protocol for negotiating over the terms and conditions for performing some task is illustrated in Fig. 1. Note that after the 'client' (C) accepts the 'server's' (S) conditions the server has an obligation both to perform the task and to inform the client of the result, at which point the conversation is successfully terminated. Either party can exit the negotiation 'loop' by issuing a reject message.

3.3 Visualisation

We are developing the CSF in the expectation that it will be used in some cases to delegate tasks to processes or 'agents'. This raises questions of trust and control, which we aim to

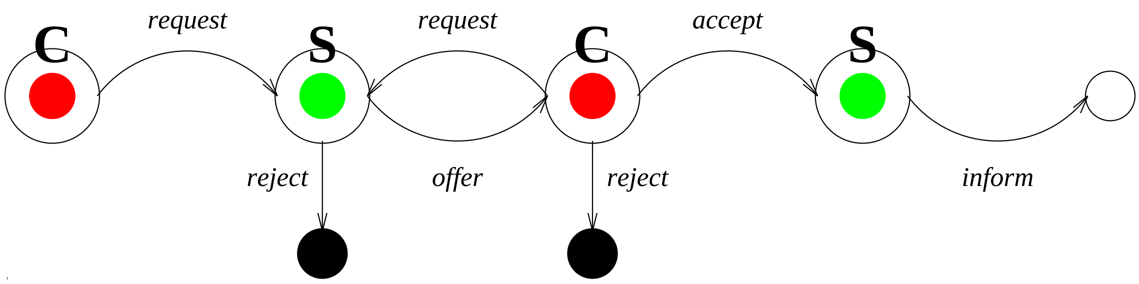

Fig. 1. Cooperation protocol for negotiation

address (in part) by including in the CSF library mechanisms to enable a user to monitor and review conversations.

We take the view that visual representations, like that shown in Fig. 1, are data structures for expressing knowledge about messages, protocols and conversations. As such, they can facilitate problem solving by providing an efficient structure for representing the data. By enabling users to observe the behaviour of their processes through an *animation* of their conversations using network diagrams, the CSF GUI can be used to verify, monitor and debug the decision-making capabilities of processes.

This is currently the least well-developed part of the CSF library, but the general idea is illustrated in the second demonstrator of the next section.

4 Demonstrators

The main module of the CSF used in these demonstrators is implemented in ECL^{i}PS^e Prolog and runs under BSD Unix. It supports inter-process communication using sockets and datagram connection with the internet domain, so multiple processes on any machine connected to a network that can can reach the internet can communicate with each other.

Sockets in the internet domain have an *address*. The address of a socket is identified by the host name and a port number: the host name is the IP address of the host (e.g. `medlar.doc.ic.ac.uk`) and the port number identifies the channel on the host which is used for the communication. This is an integer which is usually allocated by the operating system rather than specified by the programmer. To receive messages, a process has to *open* a socket which then has an address identified by host name and port number; to send messages to this socket, another process has to *connect* to the socket using that address.

Full technical details, the procedures for opening and engaging in conversations (of whatever type), and a description of the library of Prolog predicates, can be found in [22].

4.1 Cooperating Theorem Provers

The first demonstrator involves a simple distribution of tasks. We have a client process which has a set of problems to solve in a number of modal logics. To assist it, we also have a number of $\Box$**KE** server processes, possibly running on different machines. Finally there is a broker, which maintains a register of available servers (see Fig. 2).

Using the CSF, the client is initialized in *asynchronous* mode, and queries the broker for the addresses of $\Box$**KE** server processes which it can use (i.e. servers which are registered

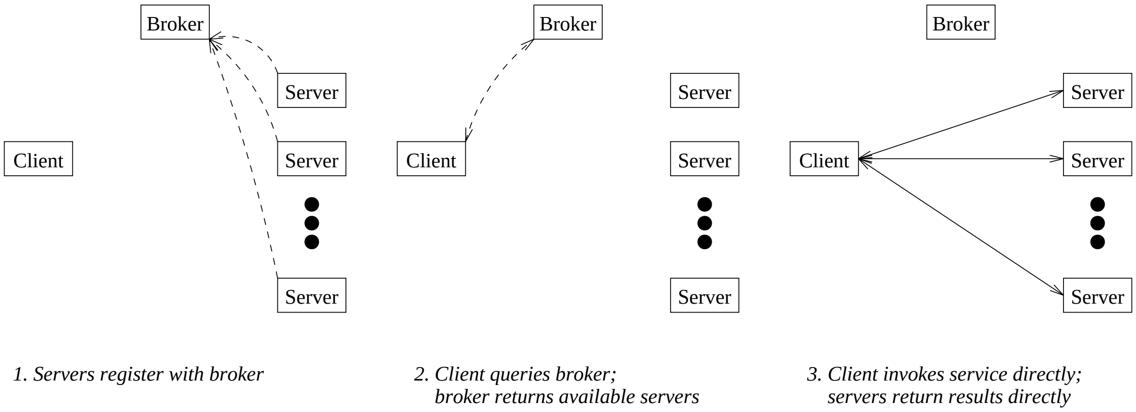

Fig. 2. Distributing tasks using $\Box$**KE** and the CSF

with the broker). When it receives a copy of the register, it opens a *direct* conversation with each server on the register. The client then parcels out 'tasks' to each server; when a server replies with the result of one task, it is allocated another, and so on until all the tasks have been completed.

We have tested this arrangement on the 31 modal formulas of [6] in each of the 15 distinct normal modal logics, making a total of 465 tasks for which the 'client' wants to determine if they are theorems of each logic or not. We made use of the 4 Sun™ workstations in our local cluster: Medlar (Sun SparcStation 2), Poplar (Sun SparcStation IPX), Arion (Sun SparcStation IPX), and Fern (Sun SparcStation 1+), and run a separate $\Box$**KE** process on each of them. The results of the experiment are threefold.

Firstly, asynchronous communication with multiple processes allows tasks to overlap. There are certain formulas which are relatively 'tough' for particular logics to try to prove, for example the formula number 11 in [6], $\Box(p \rightarrow \Box(q \rightarrow r)) \rightarrow \Diamond(q \rightarrow (\Box p \rightarrow \Diamond r))$, takes 10 times as long try to prove in **K45** (and fail) than it takes to try to prove in **S5** (and succeed). With one process, this is a source of delay: with several processes, the other processes can continue with useful work while just one is occupied with the difficult problem. The following extract from the output from a typical run with one process on each machine shows this. Here, each table entry indicates whether or not the formula is a theorem (`y` or `n`), which process worked on the task (in this case `1`=Fern, `2`=Arion, `3`=Poplar and `4`=Medlar), and the order in which replies were received. Note that formula number 11 in **K45** is the 160th task to be assigned (here to Medlar), and was the 205th reply received. The other 3 processes have continued to receive and reply to other tasks.

`Formula No.`	...	**KD5**	**K45**	**KD45**	**KB4**	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮
10	...	`y:1:143`	`n:3:144`	`y:2:145`	`n:1:147`	...
11	...	`y:1:158`	`n:4:205`	`y:2:159`	`n:1:182`	...
12	...	`n:3:172`	`y:2:173`	`y:3:174`	`y:2:175`	...
13	...	`y:1:188`	`y:2:189`	`y:3:190`	`y:1:192`	...
14	...	`y:3:201`	`y:3:204`	`y:2:203`	`y:2:206`	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Secondly, as a result of this asynchronous interleaving of task allocation and reply, task distribution to multiple processes reduces 'real' runtime, if there are enough processes. Even allowing for the overheads to start up the conversations and communication between them during execution, the expected improvement in the real time elapsed (i.e. the actual time rather than the total cputime, which of course remains more or less constant) is achieved. This is illustrated by the following results (all times in seconds):

Time	Control Medlar	1 Process Medlar	2 Processes Medlar, Poplar	3 Processes Medlar, Poplar, Arion	4 Processes Medlar, Poplar, Arion, Fern
Start Up	n/a	3	5	7	9
Execution	28	43	23	17	13
Total	28	46	28	24	22

The speed-up is noticeable rather than dramatic because the need to transmit and receive 930 messages is quite costly: fewer, harder tasks should have seen a more considerable improvement. The runtimes also indicate how the time is inversely proportional to the number of servers: the runtime for N servers multiplied by N is approximately that of the runtime for 1 server. The distortion is largely caused by one process finishing off a 'tough' task long after the other processes have finished.

Thirdly, this experiment raises the possibility of a route through to parallelising **KE** theorem provers (see also [8]), with each branch spawning a separate process. Note that because of *PB*, different branches in a **KE** tree are mutually inconsistent, so that no parallel process is redundant. It is not so with standard tableaux. For example, consider the following tableau:

$a \vee b$
$a \vee c$

branch 1 a $\quad$ b

branch 2 a $\quad$ c **branch 3**

Here, branch **1** closed clearly implies branch **2** closed, and, conversely, branch **2** open implies branch **1** open, so there is no point in pursuing both branches. Such redundancies can be nested into each other and generate an undue multiplicity of processes. In tableau theorem proving, this is usually avoided by so-called lemma generation or other *ad hoc* adjustments of the standard rules, but the **KE** calculus simply does not give rise to such redundancy and so there is no need to tamper with the rules. For a more detailed discussion of this issue, see [11].

Of course, there is a critical requirement that branches communicate their unifications to one another in lean**KE** and $\Box$**KE**, but the CSF provides the mechanisms for precisely the sort of inter-process communication that would be required.

4.2 Trading & Competing Theorem Provers

The second demonstrator uses a number of $\Box$**KE** processes, each of which is restricted to proving theorems in only a subset of the 15 normal modal logics. Each process registers with a broker its name and the logics it can prove theorems in. Whenever a process needs to prove a theorem in a logic it is not equipped to deal with, it uses asynchronous, indirect communication via the broker to *trade* with another process that does.

The procedure for proving a theorem in a logic unknown to a trading process is illustrated in figure 3.

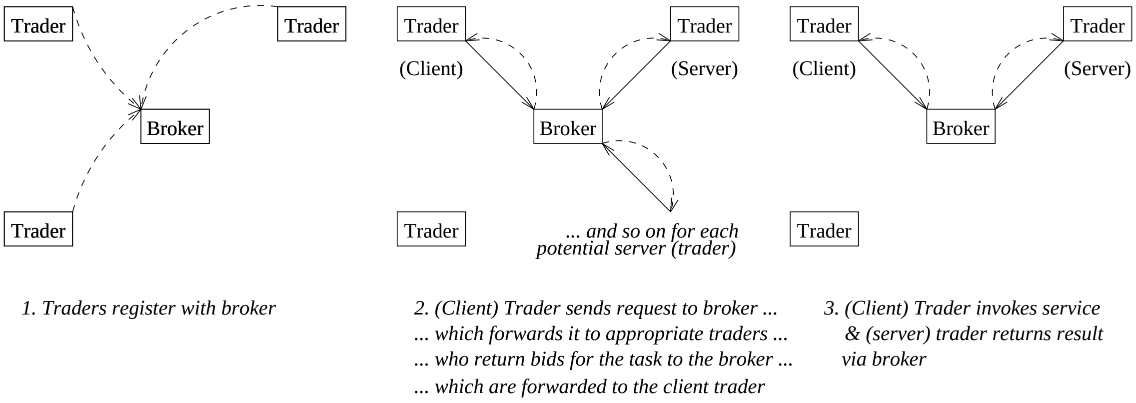

Fig. 3. Trading using $\Box$**KE** and the CSF

First the (client) trader opens a conversation with a broker, detailing which logic is required[5]. Then it sends the formula to the broker, which forwards the request to those traders which are registered to handle the logic. Each (server) trader then makes a bid for the task (i.e. the problem subsumes a version of the contract net protocol – see [5]), the collected bids are forwarded to the (client) trader. It then selects a bid and demands the service. The selected (server) trader executes the command and return the result, plus a 'charge' for cputime. The (client) trader then 'owes' the (server) trader this cputime. This can be taken into consideration whenever the latter in turn needs a service and has to choose between bids[6].

A simple Tcl/Tk graphical representation of the protocol from the broker's point of view is illustrated in Fig. 4. The (client) trader `medlar` needs a theorem proved in a logic which each of the (potential server) traders `poplar`, `fern` and `arion` know about. The ensuing conversation is *animated* in the figure. This shows how the conversation develops, with the 'black dot' indicating which process has to send the next message, and the last line of the scroll box indicating who sent what type of message to whom.

This demonstrator illustrates how the distinction between client and server can be blurred in the CSF. Each of the processes understands the common protocol for both requesting services and responding to service requests. Furthermore, it shows how 'clients' can locate 'servers' at runtime by the functions that they offer rather than by the names that they are given, as is usually the case in distributed object-oriented computing. This

5 Note that we are using the terms 'broker' and 'trader' in a different sense to CORBA and ANSA. Here a 'broker' is more like the ANSA or ISO Trader [4] (i.e. it stores information about servers and services) and the 'trader' is a process that can be both client and server (i.e. it uses and provides services).

6 This is a somewhat contrived situation given the versatility of $\Box$**KE** for proving theorems in all 15 normal modal logics. It can be lifted to situations where the 'trading processes' are offering a variety of different theorem proving services and the applications requiring such services are sharing resources, as in the situation that motivated this demonstrator, US observatories trading telescope time [16].

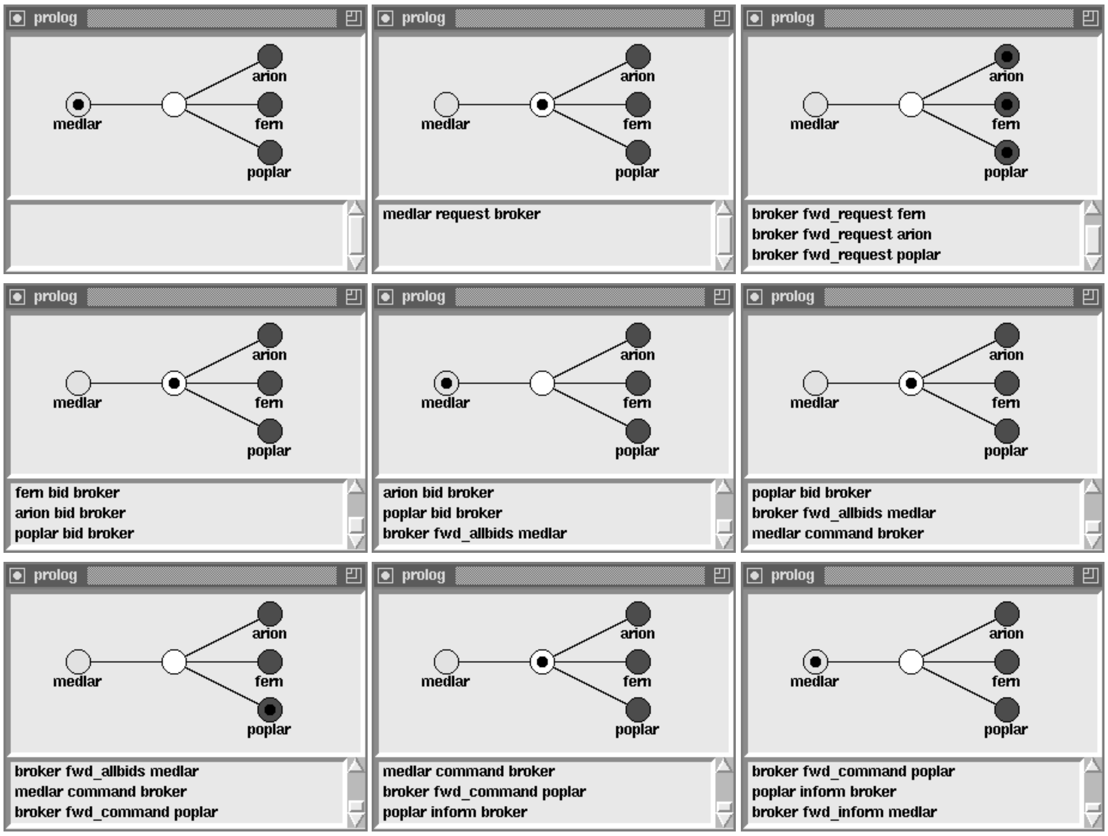

Fig. 4. Animation of a Conversation in the CSF

provides a more dynamic concept of a distributed application, as traders can come and go without disrupting the overall working of the entire system.

4.3 Online Theorem Proving

For the third and final demonstrator, we have implemented a WWW interface to 'distributed middleware', in which a client, which 'understands' the simple cooperation protocol of Fig. 1, uses this for negotiated modal theorem proving with a server via a broker, over the internet using *cgi* to connect the WWW server with the middleware (cf. [1]).

Here the scenario is as illustrated in Fig. 5. A broker maintains a web page which contains details of servers currently registered with it, offers theorem proving services in the 15 normal modal logics, and other possible user requirements.

A user can fill in the HTML form contained on the web page shown in Fig. 6(a), specifying a server, logic(s), logical query, and the terms and conditions (for example, whether the output is required in text or LaTeX). This is submitted to the local HTTP dæmon. This executes a Tcl CGI (Common Gateway Interface) program, i.e. a cgi-script, which boots up an ECL^{i}PS^e client which persists for the duration of the conversation: it understands the protocol and acts as an intermediary for the end user in all dealings with the broker and server.

The client then enters into a brokered conversation with the specified server observing the protocol illustrated in Fig. 1. Once the connection is established, the client submits the

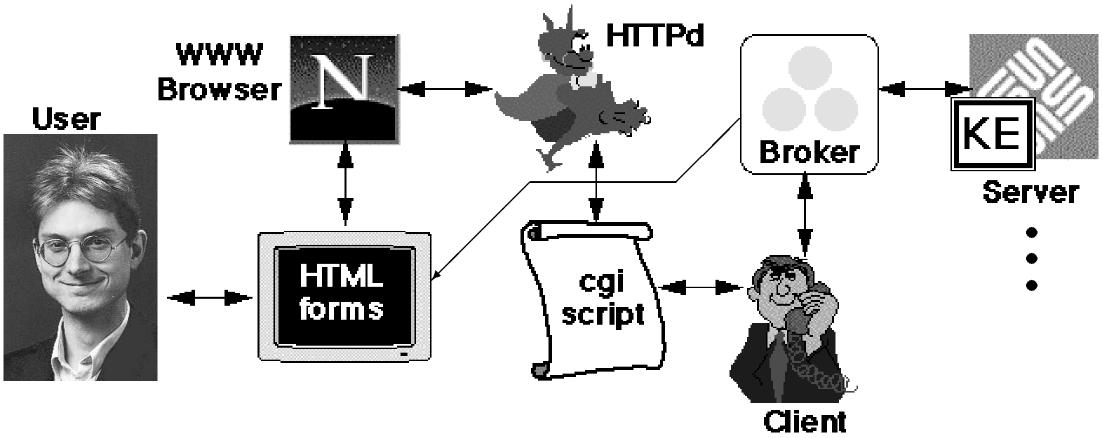

Fig. 5. WWW access to negotiated modal theorem proving

request, the server calculates a 'charge' in some putative electronic currency, and returns this to the client. The result is returned to the cgi-script, which prepares the HTML that is returned to the user as the result of the request. As shown in Fig. 6(b), the user then has several options according to the protocol. (Note that each action by the user initiates a fresh invocation of the cgi-script which communicates the information contained on the form to the client.) The user can reject the offer as too expensive, modify the request using the form, or accept the charge. The first option terminates the conversation, the second brings about a modified charge, and the third results in the request being executed. In the first and last cases the HTML returned to the user via the cgi-script is illustrated in Fig. 6(c) and (d) respectively (in the second case, the HTML looks like (b) but with the data updated to reflect the new request).

5 Summary & Conclusions

We have reviewed the implementation of $\Box$**KE**, a theorem prover for normal modal logics based on **KE** and a generalization of Fitting's prefixed tableaux. In this context, we draw attention to the work of Governatori and his colleagues [17], who use paths with variables, path unification, and also **KE**. However, Governatori uses only one type of path variable (single worlds) while we allow variables over both single and sequences of worlds, and while Governatori has only a single $\Box$ rule and unification is logic-dependent, unification in our system is independent of the logic but we have a different $\Box$ rule for each logic, each having different side conditions or *constraints* on the use of the $\Box$ rule and on closure.

We have also reviewed the CSF, and the implementation of a Prolog library for enabling brokered inter-process communication over the internet. The CSF was originally investigated in the CEC GOAL project (Esprit 6283) as a means of supporting inter-organizational computer-supported cooperative work, and was inspired by both OMG's CORBA and ANSA's Trading and Federation Model. As a result, the CSF was designed for generality, and intended to be applicable to any application which requires inter-process communication, and also to any open computing environment which spans organizational boundaries. Thus further development was conducted on the Medlar II project (Esprit

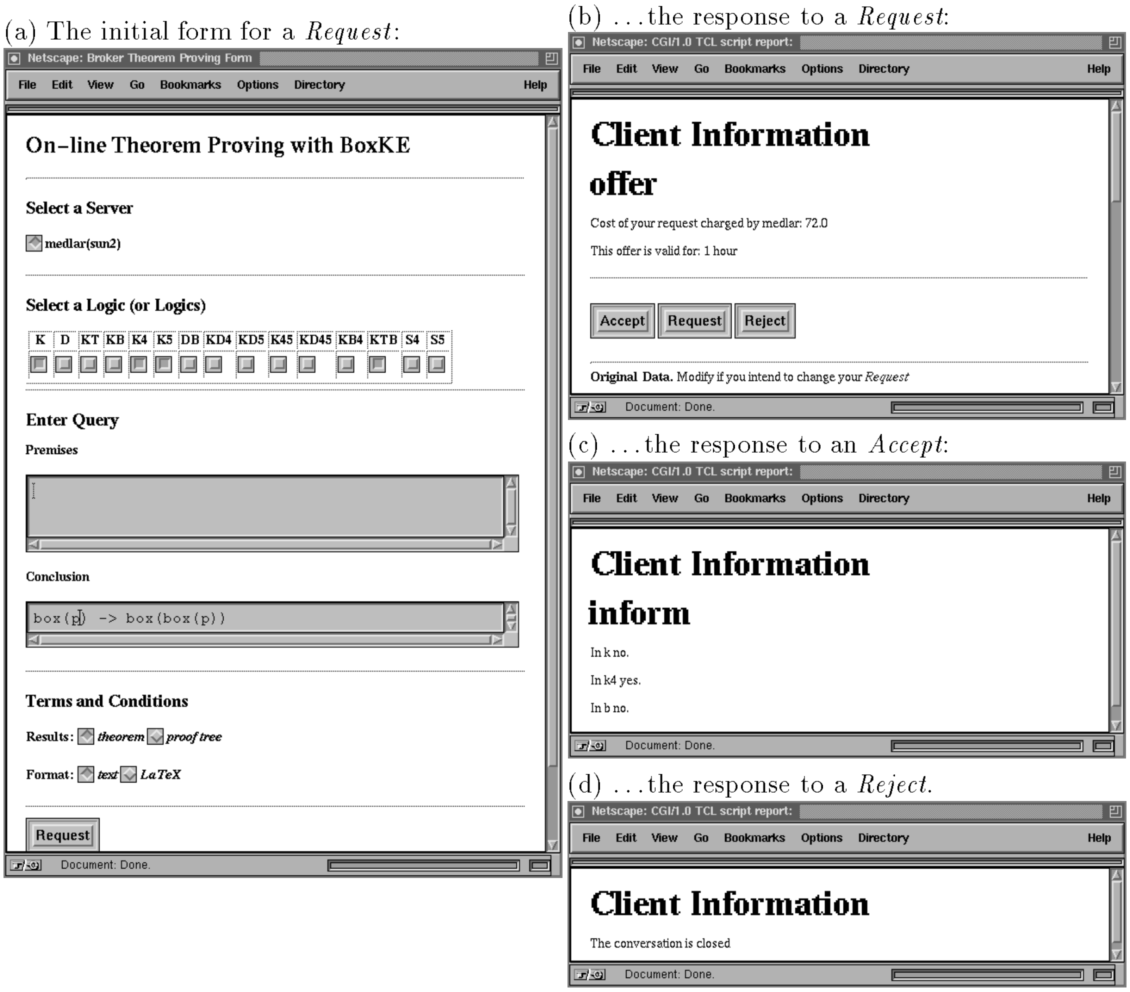

Fig. 6. WWW Interface to □**KE** via the CSF and Netscape v1.1

6283), as the CSF could provide the 'glue' that would enable heterogeneous Medlar practical reasoners to be integrated in a single ostensibly homogeneous system.

The current paper has grown out of this work. While we have not connected different provers, the first two demonstrators show how disparate systems could interact through a mutually understood mechanism for communication. The third demonstrator shows how negotiated theorem proving can be made available through a common user interface, the WWW, thus increasing access to and exploitation of these valuable resources. While there is much further research to be done, we believe that the software packages lay the foundations, and that these demonstrators establish the principles, on which substantial applications in distributed modal theorem proving can be built.

Acknowledgements. We would particularly like thank Marcello D'Agostino for his collaboration on implementing **KE** and for his comments on this paper, and we are grateful to three anonymous reviewers for their useful comments. The first author was supported by the CEC under Esprit Project GOAL (Esprit 6283) and BRA Medlar II (Esprit 6471), and should be contacted if experimental access to the WWW demonstrator is required.

References

1. G. Almasi, A. Suvaaiala, I. Muslea, C. Cascaval, T. Davis, and V. Jagannathan. Web*: A technlogy to make information available on the web. Technical Report, Concurrent Engineering Research Center, West Virginia University, 1995.
2. ANSA. The ANSA Model for Trading and Federation. Architecture Report AR.005.00, 1993.
3. B. Beckert and J. Posegga. $\mathsf{lean}T^AP$: Lean tableau-based thereom proving. *Journal of Automated Reasoning*, 1995.
4. A. Beitz and M. Bearman. An ODP trading service for DCE. In *Proceedings First International Workshop on Services in Distributed and Networked Environments*, pages 42–49. 1994.
5. B. Burmeister, A. Haddadi, and K. Sundermeyer. Generic configurable cooperation protocols for multi-agent systems. In *MAAMAW'93, Neuchatel*, 1993.
6. L. Catach. Tableaux: A general theorem prover for modal logics. *Journal of Automated Reasoning*, 7:489–510, 1991.
7. B. Chellas. *Modal Logic: An Introduction.* CUP, 1980.
8. J. Cunningham. Investigations into modular and concurrent deduction. CEC Esprit Project Medlar (Esprit 6471) Periodic Progress Report 2, Department of Computing, Imperial College of Science, Technology and Medicine, 1994.
9. J. Cunningham and J. Pitt. Towards model building in multi-modal logics using **KE** and constraint satisfaction. CEC Esprit Project Medlar (Esprit 6471) PPR3, Department of Computing, Imperial College of Science, Technology and Medicine, 1995.
10. M. D'Agostino and D. Gabbay. A generalization of analytic deduction via labelled deductive systems. part I: Basic substructural logics. *Journal of Automated Reasoning*, 13:243–281, 1994.
11. M. D'Agostino and M. Mondadori. The Taming of the Cut. *Journal of Logic and Computation*, 4:285–319, 1994.
12. ECRC and ICL. ECL^{i}PS^e user manual release 3.4.5. ECRC, Munich, 1994.
13. T. Finin and R. Fritzson. KQML — a language and protocol for knowledge and information exchange. In *Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence*, pages 126–136, Lake Quinalt, WA, 1994.
14. M. Fitting. Basic modal logic. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 1*, pages 368–448. OUP, 1993.
15. D. Gabbay. *Labelled Deductive Systems.* OUP, to appear (1996).
16. E. Germain. Software's special agents. New Scientist, 9th April, 1994.
17. G. Governatori. Labelled tableaux for multi-modal logics. In P. Baumgartner, R. Hähnle, and J. Posegga, editors, *Proceedings 4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI, pages 79–94. Springer-Verlag, 1995.
18. A. Heuerding, G. Jäger, S. Schwendimann, and M. Seyfried. Propositional logics on the computer. In P. Baumgartner, R. Hähnle, and J. Posegga, editors, *Proceedings 4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI, pages 310–323. Springer-Verlag, 1995.
19. H.-J. Ohlbach. SCAN – elimination of predicate quantifiers system description. CEC Esprit Project Medlar (Esprit 6471) Periodic Progress Report 3, Department of Computing, Imperial College of Science, Technology and Medicine, 1995.
20. OMG. The Common Object Request Broker: Architecture and Specification. OMG Document Number 91.12.1 (Revision 1.1), Wiley & Sons, 1992.
21. J. Pitt, M. Anderton, and J. Cunningham. Normalized interactions between autonomous agents: A case study in inter-organizational project management. *Computer-Supported Cooperative Work*, to appear.
22. J. Pitt and J. Cunningham. Blending the Balkans: One approach to integrating and combining inference engines. CEC Esprit Project Medlar (Esprit 6471) PPR3, Department of Computing, Imperial College of Science, Technology and Medicine, 1995.
23. J. Pitt and J. Cunningham. Theorem proving and model building with the calculus **KE**. *Journal of the IGPL*, 4(1):127–148, 1996.
24. L. Wallen. *Automated Deduction in Non-Classical Logics.* MIT Press, 1990.

This article was processed using the LaTeX macro package with LLNCS style