SOCS

A COMPUTATIONAL LOGIC MODEL FOR THE DESCRIPTION, ANALYSIS AND VERIFICATION OF GLOBAL AND OPEN SOCIETIES OF HETEROGENEOUS COMPUTEES

IST-2001-32530

Deliverable D14: Experiments with animated societies of computees

Project number:	IST-2001-32530
Project acronym:	SOCS
Document type:	D (deliverable)
Document distribution:	PP (restricted to the GC Programme)
CEC Document number:	IST32530/DIFERRARA/500/D/PP/b0
File name:	6 500-b0[d14].ps.gz
Editor:	E. Lamma
Contributing partners:	ALL
Contributing workpackages:	WP06
Estimated person months:	30
Date of completion:	10 March 2005
Date of delivery to the EC:	18 March 2005
Number of pages:	75

ABSTRACT
In this deliverable, we report on our work within WP6, which is devoted to testing the proposed SOCS models for individual computees and societies of computees, and their implementation. To this purpose, work-package WP6 aims at testing the models defined in WP1, WP2 and WP3, by exploiting PROSOCS, the outcome of WP4. As a by-product, this work is also a test of PROSOCS.

This task has been addressed by animating PROSOCS (and its components, namely SOCS-iC and SOCS-SI) with computees and their societies, and experimenting with existing scenarios identified in the first two phases of the project.

A demo will be given at the next review meeting, on April 4th. Examples to be demonstrated are available at the Web site `http://www.lia.deis.unibo.it/Research/Projects/SOCS/partners/experimentation/`.

Deliverable D14: Experiments with animated societies of computees

M. Alberti,[6] A. Bracciali,[2] F. Chesani,[5] A. Ciampolini,[5] U. Endriss,[1] M. Gavanelli,[6] A. Guerri,[5] A. Kakas,[4] E. Lamma,[6] W. Lu,[3] P. Mancarella,[2] P. Mello,[5] M. Milano,[5] F. Riguzzi,[6] F. Sadri,[1] K. Stathis,[3] G. Terreni,[2] F. Toni,[1] P. Torroni[5] and A.Yip,[4]

[1]Department of Computing, Imperial College London
Email: {ue,fs,ft}@doc.ic.ac.uk

[2]Dipartimento di Informatica, Università di Pisa
Email: {braccia,paolo,terreni}@di.unipi.it

[3]Department of Computing, City University, London
Email: {stathis,lue}@soi.city.ac.uk

[4]Department of Computer Science, University of Cyprus
Email: antonis@cs.ucy.ac.cy,ay1@doc.ic.ac.uk

[5]DEIS, Università di Bologna
Email: {aciampolini,fchesani,aguerri,pmello,mmilano,ptorroni}@deis.unibo.it

[6]Dipartimento di Ingegneria, Università di Ferrara
Email: {malberti,mgavanelli,elamma,friguzzi}@ing.unife.it

ABSTRACT
In this deliverable, we report on our work within WP6, which is devoted to testing the proposed SOCS models for individual computees and societies of computees, and their implementation. To this purpose, work-package WP6 aims at testing the models defined in WP1, WP2 and WP3, by exploiting PROSOCS, the outcome of WP4. As a by-product, this work is also a test of PROSOCS.

This task has been addressed by animating PROSOCS (and its components, namely SOCS-iC and SOCS-SI) with computees and their societies, and experimenting with existing scenarios identified in the first two phases of the project.

A demo will be given at the next review meeting, on April 4th. Examples to be demonstrated are available at the Web site http://www.lia.deis.unibo.it/Research/Projects/SOCS/partners/experimentation/.

Contents

Part I
WP6: aims, methodology, task allocation

1 Introduction

The purpose of this deliverable is to provide a report of work done on workpackage WP6 of SOCS project, focussing on experimentation. Together with work-package WP5, WP6 aims at achieving Milestone 3 of the project:

> **Milestone 3:** Verifiable properties of (societies of) computees;
> Experimentation.

The two approaches followed in WP5 and WP6 are complementary: during the last year of the project, these workpackages are dedicated to the formal verification of properties and to the observation of characteristics of (societies of) computees, as implemented in the PROSOCS system, respectively.

We distinguish between (formal) *properties* and (empirically observable) *characteristics* of (societies of) computees. Formal properties are those that can be proved (or disproved) by means of formal verification; characteristics refer to properties which can only be observed empirically. WP5 concentrates on properties of the first class, whereas WP6 addresses characteristics emerging from the animation of (societies of) computees.

Concretely, work-package WP6 seeks to:

- evaluate and validate the framework (computee and society models, PROSOCS platform and its two main components, SOCS-iC and SOCS-SI) developed by the SOCS project through a series of controlled experiments;

- experiment with the models and their implementation in the chosen scenarios, in order to test their potential and limitations in a Global Computing setting;

- possibly confirm/disprove the properties of computees and their societies investigated within WP5, and identify (emerging) novel characteristics of the SOCS models.

Deliverable overview. This deliverable is structured as follows. In this part (Part I), after introducing the aims of WP6, we briefly recall the architecture of PROSOCS, the prototype developed along the third year of the project in the context of workpackage WP4, and its extensions. We, then, identify the classes of experiments conducted, and provide the task allocation amongst the partners for the work carried out along the third year. In Part II, we report about experiments done, pertaining to the PROSOCS platform itself, its application to chosen concrete scenarios and its application to confirm or disconfirm specific properties respectively. Part III summarizes achieved results, and it is structured into three parts. It presents a catalogue of protocols available on the Web, is devoted to evaluation and self-assessment, and summarizes publications and given demos contributing to the aims of WP6. We then conclude in Part IV.

Annexed documents. The following documents, which discuss, in more detail, various aspects of work done in the context of WP6 are annexed to this deliverable and they are also listed, with abstracts, in Appendix A:

- *Experiments on Combinatorial Auctions* [6];
- *Computee Profiles in PROSOCS* [1];

At the next review meeting, we will show a demo pertaining the Combinatorial Auction scenario and experiments with different behaviour profiles. Examples to be demonstrated are available at `http://www.lia.deis.unibo.it/Research/Projects/SOCS/partners/experimentation/`.

Furthermore, most relevant papers to work done in the context of WP6 are collected at the Web page: `http://www-lia.deis.unibo.it/Research/Projects/SOCS/guests/publications/D14.html`. Some of them are also common to work done in the context of WP5 (see Deliverable D13).

As concerns comparisons, two issues are relevant to experimentation. The first concerns the kind of platform we are experimenting upon. The second is more concerned with the experiments performed and the proposed approach for automatically proving/disproving properties, developed in the context of Workpackage WP5, and experimented within Workpackage WP6.

An exhaustive survey of related platforms is already presented in deliverable D9 [4], and we refer to Section 7.3 of that deliverable for this issue.

Comparisons with other related techniques are reported in the various sections, for each of the experiments, when needed. Comparisons with model checking techniques [18], adopted to automatically prove or disprove properties, are reported as well at the end of the appropriate experiments.

In the following, we briefly summarize the PROSOCS architecture, its main components and extensions provided along the third year of the project. Then, we identify our *classes of experiments*, and present the WP6 task allocation as defined along the third year of the project.

2 Recap of PROSOCS

In this section, we provide a brief recap of the PROSOCS platform (for a complete description, the reader can refer to the project deliverables D4 [40], D5 [50], D8 [34] concerning SOCS individual computee and society models, and underlying operational frameworks, respectively, and D9 [4] concerning their implementation in PROSOCS). The updated version of PROSOCS can be downloaded at the same site devoted to experimentation: `http://www.lia.deis.unibo.it/Research/Projects/SOCS/partners/experimentation/`.

2.1 The Original PROSOCS Prototype

In the development of PROSOCS described in [4] we focused mainly on integrating the two components of the PROSOCS platform, viz. SOCS-iC and SOCS-SI. These components have been the subjects of experimentation done till now, including their internal modules.

The conceptual organization of PROSOCS presented in [4] is depicted in Figure 1. The figure shows a *Computee*, an agent equipped with reasoning capabilities allowing it to interact with its *Environment*, which is composed of all the other computees, a *Social Infrastructure* to provide verification of compliance to social rules of the computee interaction, and a *Medium* to support

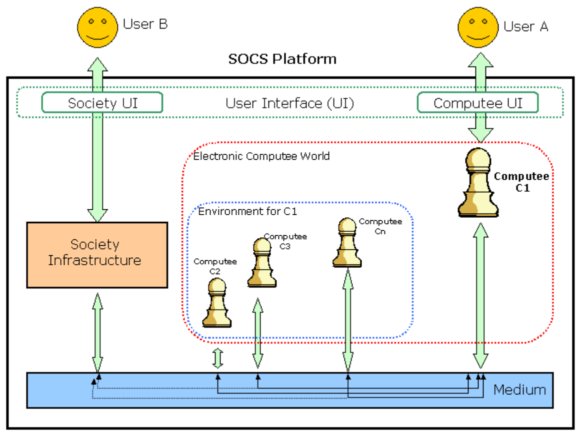


Figure 1: The conceptual organization of PROSOCS

inter-computee communication. Both individual computees and the social infrastructure are equipped with a Graphical User Interface (GUI).

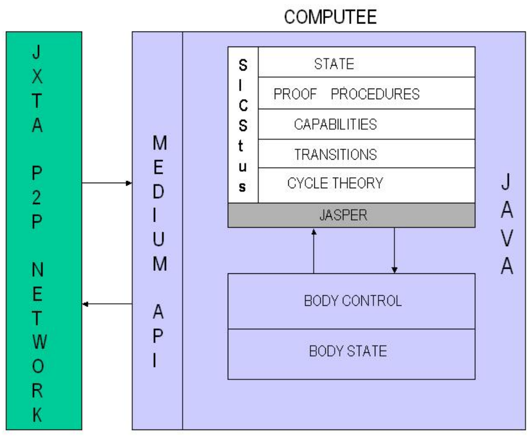


Figure 2: Implementation architecture of a PROSOCS computee (SOCS-iC)

SOCS-iC (see Figure 2) implements the KGP (Knowledge, Goals, Plan) model (described in deliverable D4 [40]). As shown in Figure 2,

- a computee is based on a set of *reasoning capabilities*, which allow the computee to perform *planning*, *temporal reasoning*, *identification of preconditions*, *reactivity* and *goal decision*;
- capabilities are supported by the *proof procedures* CIFF (abductive) and Gorgias (based on LPwNF, see deliverable D8 [34]).
- a *cycle theory* is used to regulate how transitions (basically employing previous capabilities) are sequenced for the computee to behave in a particular way. The cycle theory

is implemented in Gorgias as well, and it can implement one of the identified profiles of behaviour (see Section 2.3);

- a computee is equipped with an *internal state* on which its various capabilities operate. The internal state is composed of a KB (representing the computee's knowledge about itself and of the environment, and consisting of separate modules each one concerning the different reasoning capabilities), a set of *Goals* (properties that the computee has decided that it wants to achieve, possibly constrained by temporal constraints), and a set of *Plans* (partially ordered sets of concrete *actions*, by means of which the computee intends to achieve its goals).

Both the underlying proof procedures, CIFF and Gorgias, are implemented in SICStus Prolog [63]; the SOCS-iC GUI is implemented in Java.

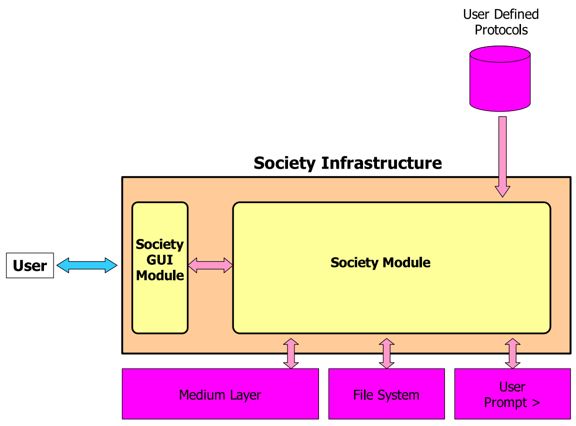


Figure 3: Overview of the Society Infrastructure architecture (SOCS-SI)

As shown in Figure 3, SOCS-SI implements the *Society Infrastructure* (see Deliverable D8 [34] and D9 [4]) of PROSOCS, and it is committed to the verification of compliance to protocols of the computees' interactions.

The verification is performed by means of the $\mathcal{S}$CIFF abductive proof procedure (see deliverable D8 [34]).

As defined in deliverable D5 [50], the knowledge about the interaction protocols in the society is represented by an abductive logic program (composed of a logic program called *knowledge base*, a set of *Social Integrity Constraints* and a set of *abducibles*), while the knowledge about the computee interaction is represented as a *history* of events. Given a (partial) history of the computee interaction, $\mathcal{S}$CIFF generates a set of abducibles (*expectations*) which represent a course of action that would satisfy the interaction protocols; checking the expectations against the events, the society module is able to give an answer of *fulfillment* (the interaction protocols have been respected) or *violation* (the interaction protocols have *not* been respected).

The $\mathcal{S}$CIFF proof procedure is implemented in SICStus Prolog [63]; the SOCS-SI GUI is implemented in Java.

2.2 Extending PROSOCS with Objects

The first implementation of the PROSOCS platform integrated SOCS-iC and SOCS-SI using a specific reference model which was presented in two companion papers, [10] and [66]. For the purposes of this experimentation, however, the original model has been extended in order to allow the interaction of computees with objects, i.e., entities other than computees which differentiate themselves from computees and the society in that they do not require a reasoning component [46].

Figure 4 shows the new/extended PROSOCS reference model, as presented in [46]. This new model keeps from the original PROSOCS model a *discovery component*, which enables parts of a computee, such as its sensors, to discover other computees and objects, when these are created for an application. The functionality of SOCS-iC is provided via the *computee template*, which allows us to create instances of computees with a mind and body as discussed in [66], including how these components interact and communicate with the environment via sensors and effectors. Similarly, the functionality of SOCS-SI is provided via a *social infrastructure*, which provides us with mechanisms that check for conformance of the interactions according to well-defined "social rules".

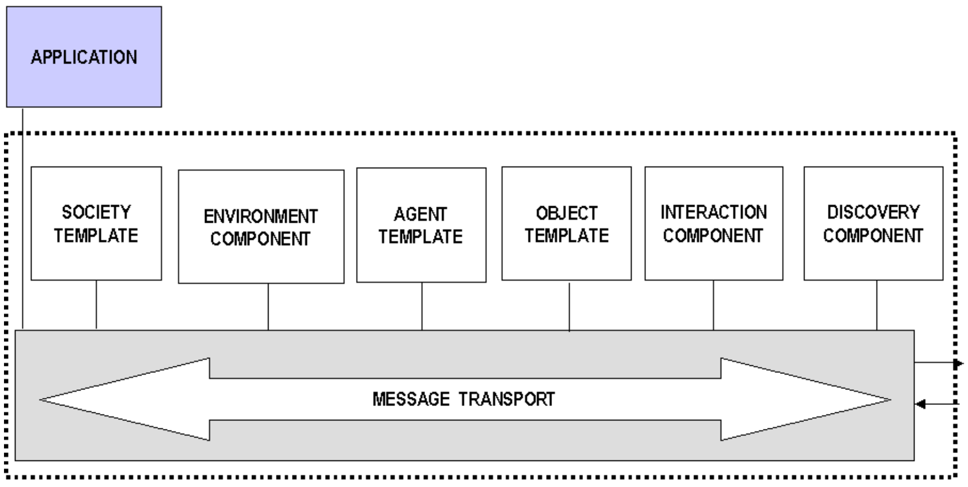


Figure 4: The PROSOCS Reference Model.

A new component, which we call the *object template*, is introduced to allow instances of PROSOCS objects to be created, so that computees can interact with objects, or objects can interact between them. To accommodate this, we have rationalized the *communication component* in the old version of the reference model (see [66] and deliverable D9 [4]) with what we call the *interaction component* in the new version. This component is essentially a component that allows us to enable interaction between computees and objects (or between objects), and communicative interactions in terms of exchanges of different types of messages. Finally, we have rationalized the *computee management* component in the old version, with an *environment component* in the new version, which keeps a record of the existence of computees and objects, and allows a computee to perform physical actions on objects or other computees.

We show in later sections how the implementation of the new reference model of PROSOCS will allow the platform to deploy a set of *computees* and a set of *object* in an artificial world and, when necessary, a *society* that will allow the experimentation process to check interactions between computees and objects for an application.

2.3 A Library of Behaviour Profiles

Along the third year, PROSOCS was also equipped with a library of behaviour profiles. As presented in Deliverable D12 [5], we identified a collection of profiles of behaviour. They are:

- Cautious - A cautious computee does not (or prefer not to) execute actions which have preconditions that it cannot currently show to be true.
- Focussed - A focussed computee remains committed to the same goal amongst its top-level goals until it has either successfully achieved this goal or this goal has become infeasible to complete or is not preferred by the Goal Decision capability before it moves on to other goals.
- Punctual - A punctual computee is committed to execute actions and plan for goals when they become urgent, in the sense that they are close to their deadlines.
- Actively Cautious - An actively cautious computee behaves like a cautious computee but if it does not know whether a precondition is true of false, the computee will also actively observe its environment to find out the truth value of these preconditions.
- Careful - A careful computee will revise its current commitments frequently so that any new information obtained can be taken into account at a timely fashion.
- Objective - An objective computee always attempt to check immediately after the execution of an action if its desired effect has been achieved.
- Impatient - An impatient computee will abandon an action when it realises that its execution did not produce the desired effect, e.g., failed. It will only try the action again if there is no other action to be planned, and the original action has not timed out.

When a computee employs a profile, the operational pattern of the computee will exhibit certain desired characteristic. For example, a punctual computee will try the best to complete a goal before the deadline; while a cautious computee will make sure the pre-condition of an action is satisfied before attempting execution, hence saving valuable time and resources. Each profile is a specialized cycle theory. A profile may specialize in any or all of following components:

- τBasic
- τBehaviour
- Selection Functions

The aim of the profile library is to make these profiles available as standard modules that can be selected and utilized to create computees with predefined operational behaviour. In order to achieve this, we have to extend the implementation of the PROSOCS platform with supporting mechanism for profiles. Please refer to the annexed document titled *Computee Profiles in PROSOCS* [1] for the implementation methodology and technical details. At this moment, we have implemented most of the profiles, please refer to Section 7 for the experimental results of some of these profiles.

3 Classes of Experiments

For the purposes of WP6, we have identified the following three classes of experiments:

1 Testing the PROSOCS platform (and its components, also independently);

2 Experiment models and their implementation in chosen concrete scenarios;

3 Investigating properties.

We have planned WP6 activity in order to concentrate on the following steps (among them, (i) and (ii) are related with 1 and 2 above; (iii) with 3 above):

(i) Parameter identification. In WP6 we have started to investigate parameters of the framework developed within WP1-WP3 and implemented in PROSOCS within WP4, such as: reasoning capabilities of the individual computees, behavior profile of computees, mode of communication and interaction (society protocols), scale of density of computees in a society, degree of heterogeneity of the composition of societies, frequency of revision of computees' knowledge, goal and plans as new circumstances occur. Some of them are specific to experiments for testing PROSOCS and its components (and they are named $\mathcal{P}_{PROSOCS}$, for short). Others relate more to scenarios (and they are named $\mathcal{P}_{SCENARIOS}$, for short).

(ii) Develop and evaluate specific experiments. Specific experiments on (societies of) computees and scenarios have been identified in order to test and control how the identified parameters impact on the overall system. For each of the experiments set out above, the behavior of the system is then examined.

Further experimentation in WP6 is linked to WP5 in the following way:

(iii) Develop specific experiments for properties. Experimental verification of formally proved properties is necessary because the theoretical framework must make assumptions which may not be satisfied in real world applications. For some of the identified (and formally proved) properties, partners have been asked to identify specific experiments (as simple as possible) on societies of computees and scenarios set out in order to experimentally test the property. Evaluation of experiments for properties has been conducted with the aim to confirm or disprove that the system exhibits the desired features, and possibly for discovering novel, relevant characteristics. We have also experimented a proof theoretic approach to automatically prove or disprove properties.

Along the second and third year of the project, we have already identified two notable scenarios to be considered for experimentation along the project, namely the case of Combinatorial Auctions scenario (see the internal document entitled *Combinatorial Auctions* [49], produced along the first year of the project, and document entitled *Experiments on Combinatorial Auctions* [6] annexed to this deliverable) and the Music for Beer scenario (see document entitled *Further Examples for the functioning of computees* [23], which was annexed to Deliverable D9).

Furthermore, partners were involved in experimenting specific properties of either single computees or societies of computees, and automatically proving protocol properties.

Parameters	**ICSTM**	**DIPISA**	**CITY**	**UCY**	**UNIBO**	**DIFER.**
$\mathcal{P}_{PROSOCS}$, $\mathcal{P}_{SCENARIOS}$ computee side	✓	✓	✓	✓		
$\mathcal{P}_{PROSOCS}$, $\mathcal{P}_{SCENARIOS}$ society side					✓	✓
Testing PROSOCS						
Computee side	✓	✓	✓	✓		
Society side					✓	✓
Medium			✓		✓	
Overall platform	✓	✓	✓	✓	✓	✓
Scenarios						
Combinatorial Auctions	✓		✓		✓	✓
Music for Beer				✓	✓	
Properties						
See WP5 task allocation	✓	✓	✓	✓	✓	✓

Table 1: WP6 Task Allocation

4 WP6 Task Allocation

The task allocation for WP6 is the following. Table 1 shows which partners contributed to identify parameters for experiments and to set and run experiments for the classes identified earlier.

With respect to task allocation, since the beginning of year three, unfortunately UCY was in a difficult position since it has lost its RA, and therefore one of the two scenarios, the Music for Beer, was only partially experimented.

It was also agreed that the WP4 team, involving one person per site, was engaged as support for any tuning of PROSOCS and its main components that the experimentation required.

Part II
Experimentation

5 Testing PROSOCS

In this section, we report on experiments testing the PROSOCS platform. They are aiming to (*i*) tune the components of individual computees (e.g., computee's capabilities), (*ii*) tune the society infrastructure components and underlying proof procedures, and/or also for (*iii*) evaluate overall performance.

The various experiments, when increasing in size, have helped to assess the scalability of the PROSOCS platform resulting from WP4, and extended as discussed in previous Section 2.2.

We first summarize the identified parameters, and then present the various experiments for testing PROSOCS components. Further tests for the whole integrated PROSOCS platform are reported in Section 6, pertaining to the chosen scenarios.

5.1 $\mathcal{P}_{PROSOCS}$: Identified parameters for testing PROSOCS

We first present the main identified parameters of variation of both individual computees and societies of computees. As mentioned above, they may concern the PROSOCS platform (and its components, SOCS-iC and SOCS-SI) and/or scenarios.

As for testing PROSOCS, one of the aims of the experimentation was to evaluate the computational cost of the various reasoning capabilities for the individual computees and the computational cost of the compliance verification for societies of computees. For both cases, as usual when evaluating the performance of a piece of software, *output parameters to be observed* as computational cost are the *time* and *space* trade-off.

$\mathcal{P}_{PROSOCS}$ computee side. The experimentation of SOCS-iC, the component supporting the KGP model and implementing it on top of the CIFF and Gorgias proof procedures, is mainly aimed at evaluating its computational cost.

For the experimentation phase, we have chosen to vary some of the KGP single reasoning capabilities, (e.g., Temporal Reasoning, Reactivity, as we will see below) by varying the underlying supports for them (e.g., by considering a simpler abductive proof procedure for TR, and a different version of CIFF), in order to evaluate what improvements can be achieved in terms of performance and effectiveness of capabilities by using different underlying proof procedures for Abductive Logic Programming (ALP). This also allows us to experiment and evaluate the modularity of the platform, when some (of the underlying) components are replaced.

Further details about parameters varied in CIFF are reported in the devoted section (5.2).

$\mathcal{P}_{PROSOCS}$ society side. Similarly, the experimentation of SOCS-SI, the SOCS compliance verifier, is mainly aimed at evaluating the computational cost of the compliance verification addressed by the $\mathcal{S}$CIFF proof procedure in terms of the *time* and *space* trade-off.

SOCS-SI (and its underlying proof procedure $\mathcal{S}$CIFF, in particular) was designed to be used for *on-the-fly* compliance check, as well as for static *a-posteriori* check of the interactions. Although the computation time is not a direct issue for the *a-posteriori* compliance verification, time becomes a critical issue in the case of *on-the-fly* verification. In the latter case, SOCS-SI should behave like a *real-time* tool, that acts as quickly as possible whenever the interaction between computees evolves.

Each $\mathcal{S}$CIFF computation produces a search tree whose *depth* and *breadth* determine the total number of nodes, and thus influence the time needed to explore the (whole) tree. We have considered two different versions of $\mathcal{S}$CIFF (*f-non-deterministic* or *f-deterministic*) in order to improve performance. Details about parameters varied in the two versions of $\mathcal{S}$CIFF are reported in the devoted section (5.5).

Memory requirements are also important in order to determine the (maximum) dimension of societies (in terms of participants and/or interactions) that the actual implementation of SOCS-SI (and $\mathcal{S}$CIFF) can support.

Input parameters that were taken into account as influencing the computational cost of a SOCS-SI computation, and varied in the experiments, are:

- the length of the interaction (which is in turn determined by the number of interacting computees, the length of a compliant interaction and the number of interactions between given computees);
- the interaction protocols (in particular, the number of social integrity constraints composing the protocols and the number of disjuncts in their heads);
- the version of $\mathcal{S}$CIFF proof procedure (*f-non-deterministic* or *f-deterministic*).

5.2 Improving the CIFF proof procedure (ICSTM-DIPISA)

Aim. The aim of this class of experiments has been to support us in our efforts to improve the efficiency of the CIFF proof procedure for ALP with constraints, as well as its implementation. We have presented CIFF 2.0 at JELIA-2004 [25] and the new version of the system has been publicly available since September 2004.[1]

Parameters considered and varied. We have considered different variants of the CIFF procedure by introducing the following parameters into its specification (these parameters can be considered in addition to $\mathcal{P}_{PROSOCS}$; we list them here since they are specific to CIFF):

(i) *Switching factoring on and off.* As shown in [26], the *factoring rule* proposed in [33] for the original IFF proof procedure is not required to ensure soundness (but neither does it jeopardize soundness). The first parameter is therefore whether or not to include factoring into the setup of CIFF.

(ii) *Guided propagation.* The *propagation rule* of CIFF allows for an atomic formula to be resolved with any matching atom in the antecedent of an implication. This rule can be refined by allowing propagation only with respect to the leftmost atom (that is neither an equality nor a constraint atom) in the antecedent of an implication. This refinement does not affect soundness, and can reduce the number of residues of integrity constraints to be considered in a proof.[2] The second parameter is therefore whether to use the original propagation rule or its refinement when running CIFF.

[1]The URL of the CIFF Web site is `http://www.doc.ic.ac.uk/~ue/ciff/`.

[2]As an example, consider a node including the implication $p \wedge q \rightarrow r$ and the atoms p and q. For the original version of the procedure, both $p \rightarrow r$ and $q \rightarrow r$ would be generated, while the refined version only generates the latter. The fact that r is derivable from the node is not affected by this refinement.

(iii) *Integrating unary disjunction rewriting.* The implementation of CIFF has to include certain proof rules that humans would apply implicitly, such as rewriting a disjunction consisting of only a single disjunct as that very disjunct. Our original heuristic has been to give high preference to these very simple rules, while our new approach is not to waste computing time on constantly testing for their applicability, but rather to integrate them into the more complex rules. For instance, as we have observed that *rewriting unary disjunctions* is often applied directly after an unfolding step, we have combined these two rules.

(iv) *Ordering of proof rules.* An important parameter in the configuration of CIFF is the order in which we attempt proof rules during a derivation. We have experimented with different ways of *ordering proof rules* to identify the configuration most suitable for typical queries in PROSOCS. Amongst the main rules, our new heuristics give highest priority to the constraint rules and lowest priority to the unfolding rule for implications, to case analysis for equalities, and to factoring. Minor proof rules, such as rewriting according to logical equivalences are also given lower priority than before.

(v) *Early equality-constraint rewriting.* The *equality-constraint rewriting rule* translates equalities such as `T=7` into explicit constraint predicates (in this case `T#=7`), whenever an equality includes a variable that also occurs elsewhere inside a temporal constraint (see [27] for a precise statement of the rule). In the original implementation, we have only performed this kind of rewrite step at the very end of any given branch in the proof tree; our new policy is to apply this step immediately on encountering a new equality.

(vi) *Triggering.* In the context of knowledge bases based on the abductive event calculus, we have experimented with a variant of CIFF where unfolding of the `holds_at` predicate inside an integrity constraint is not allowed (but the propagation rule should be used instead). This approach can both improve performance and avoid problems with non-allowed abductive programs. We call this variant of the procedure CIFF *with triggering* (because the use of the integrity constraints in question needs to be "triggered" by matching atomic goals). Besides studying its theoretical properties [34], we have also included triggering as a parameter in our experiments.

Knowledge bases. We have further experimented with a variety of the knowledge bases for CIFF-based capabilities of computees (namely planning, reactivity and temporal reasoning) described in the *Examples Document* annexed to D9 [3] and some variants thereof. For planning in particular, we have also experimented with knowledge bases taken from the blocks-world domain (see also the experiment reported in Section 5.4).

Runs executed. We have timed different calls to these capabilities for different settings of CIFF: either with or without factoring, with the standard propagation rule or with its refinement, and following different heuristics with respect to items (iii)-(v). In the case of temporal reasoning, we have also studied the behavior of CIFF with and without triggering in detail. Further details are available in Sections 5.3 (experiments with temporal reasoning) and 5.4 (experiments with planning).

Evaluation. Our experiments have led to significant improvements of CIFF over a wide range of examples, for all three of the CIFF-based reasoning capabilities available in PROSOCS (planning, reactivity and temporal reasoning). In particular, our experiments strongly suggest that our new heuristics for ordering and combining proof rules in the implementation of CIFF are superior to the previous version (as confirmed by all the runs executed). For planning in particular (see Section 5.4), following the early equality-constraint rewriting has resulted in noticeable improvements. In experiments based on a blocks-world scenario, the new version of CIFF can solve problems efficiently that previously led to loops, by cutting down the search space more effectively.

As concerns factoring, we have not been able to identify examples for which CIFF performs either significantly better or significantly worse by dropping the factoring rule. In the case of the refined propagation rule, we have also not been able to measure noticeable improvements in performance for any of our application examples. However, the new rule is clearly superior to the standard rule, as may easily be checked using theoretical arguments (rather than experimentation). It is not difficult to design examples that require fewer proof steps using the new rule rather than the old one.

While the use of triggering can result in very strong performance improvements for temporal reasoning, our experiments have also identified cases for which this variant of CIFF is in fact not sound (see Section 5.3).[3]

Related work. Implemented abductive proof procedures that are related to CIFF include the A-system [42] and the $\mathcal{S}$CIFF procedure developed within SOCS. However, given the improvements discussed here are very specific to both CIFF itself and to our concrete implementation of CIFF, there is no closely related work as such that we could compare our work with.

5.3 Temporal Reasoning: further development (UCY-DIPISA)

Aim. As part of the further development of the PROSOCS platform, we have run experiments aimed to test and possibly improve the computational efficiency of the implementation of the Temporal Reasoning (TR) component. More specifically we have tried to improve the computational support provided by the adopted proof procedure which TR relies on. A new proof procedure, different from CIFF, has been considered to be used as computational support for TR. This has the aim of

1. studying how the effectiveness of the capability can be improved by using a different underlying proof procedure for Abductive Logic Programming (ALP), and

2. experimenting and evaluating the modularity of the platform, when some components are substituted or others are plugged in.

The experiment has been conducted by re-engineering the core of the TR implementation on top of the ASLD(N,IC) system, and comparing the performance of the new component with the performance of the TR running on top of a new release of the CIFF proof procedure, which has been in the meantime developed by the SOCS Consortium.

Future extensions of the experiment may consist in varying the expressiveness allowed in the KB_{TR} of the computee with respect to the different proof procedure used as computational support.

[3]Note that this configuration of the temporal reasoning capability has only been considered experimentally, but not theoretically.

Parameters considered and varied: the experiment The experimentation regards mainly $\mathcal{P}_{PROSOCS}$ parameters relative to the platform. The ASLD(N,IC) system (see `http://www.cs.ucy.ac.cy/aclp` and `http://www.doc.ic.ac.uk/~or/`) has been used in order to provide the computational support for TR. This proof procedure extends the earlier ALP proof procedure of Kakas and Mancarella [41], with more sophisticated selection and pruning mechanisms.

Currently, we have worked on the capability in isolation by re-engineering the core of TR on top of the new procedure. This has required some adaptation to the different syntax of the ALP language accepted by the proof procedure, to the different API interface the procedure exposes, and to the different model of computation the procedure uses (informally speaking, while CIFF works by applying rewriting rules to a list representing the ALP theory, differently ASLD(N,IC) allows the theory to be imported in the current interpreted program). We expect that a possible integration of TR+ASLD(N,IC) in the PROSOCS platform should be facilitated by the fact that both the CIFF and the ASLD(N,IC) proof procedures are written in SICStus Prolog, which is already integrated in the engineered architecture.

The knowledge bases pertaining to the Temporal Reasoning capability needed small adjustments in order to be adapted to the new proof procedure, but this does not affect the overall interface of the capability with the rest of the architecture, and, basically, consists of minor syntactical adaptations.

In the following we illustrate and discuss the performance of CIFF and ASLD(N,IC) as supporting proof procedure for TR. In particular, we have compared the CIFF 2.0 with the switch *triggering* on (see Section 5.2), CIFF 2.0 with the switch *triggering* off, and ASLD(N,IC).

Knowledge bases. The experiment has been run for the case of ground knowledge bases and ground queries. Indeed, the SICStus Constraint Solver would need to be integrated either into TR or into ASLD(N,IC), in order to support more fully temporal constraints. However, for the purpose of comparing and studying TR performances relying on different proof procedures, this is not a limitation for the experiments (about the difficulties of the modular integration of different proof procedure in the overall $\mathcal{P}_{PROSOCS}$ architecture, we again expect that the Constraint Solver supported by SICStus would make the integration easy, in this respect).

The comparison has been done for a given example, consisting of the following domain dependent knowledge base and narration:

```
% FLUENTS: go, parked, fuel
% EVENTS: drive, park, refill

initiates(drive, _, go).

terminates(drive, _, parked).

initiates(park, _, parked).

terminates(park, _, go).

precondition(drive, fuel).

initiates(refill, _, fuel).
```

```
% NARRATION:
%

observed(neg(fuel),0).

executed(refill,5).

executed(drive,10).

executed(park,20).
```

Runs executed. Results, in terms of execution time and answers, are reported in Table 2. Even if the two implementations of TR have a very similar representation of the object theory (Abductive Event Calculus), they present some differences:

- ASLD(N,IC) is not provided with a (temporal) constraint solver and it is able to deal with ground(ed) theories only,
- CIFF has a syntactical representation of the object theory (a list of terms) and operates by means of rewriting steps,
- ASLD(N,IC) asserts the object theory (and integrity constraints are, currently, hard-wired in the implementation and not "user-definable").

Each run relative to a different proof procedure has been executed by restarting the Prolog interpreter and setting-up the object theory, then all the queries have been asked in the same session. The test has been run on a 1Gb 1.8Mhz Pentium III machine. Both the implementations of TR have some extra code for debugging (basically some "print message") and time measurements. The overhead should be comparable for both the implementations. No implementation presents not expected multiple-solutions.

Evaluation. CIFF with *triggering* on performs (much) faster (about 20 times) than ASLD(N,IC), and also faster than CIFF with *triggering* off. Unfortunately, it seems that *triggering* on does not guarantee soundness, when the procedure is used with an abductive theory with integrity constraints like the ones in TR. Indeed, it computes a wrong negative answer for the query `neg(fuel) 3` (see Table 2), clearly not consistently with the initial observation `neg(fuel)` (this "inconsistency" with the narration seems to underline the fact that integrity constraints are not properly dealt with — the triggering option does actually affect them). Differently, the *triggering* off CIFF computes the correct answer (the two cases differ only because of triggering), but requiring a significantly longer execution time (although it must be reminded that the previous version of CIFF 1.0 was often not terminating or returning an `out of memory` error, in several similar cases). In conclusion, amongst the correct implementations for ALP the ASLD(N,IC) system is significantly more effective.

We must also notice that the current implementation of ASLD(N,IC) does not have a temporal constraint solver and hence it supports only ground queries. This limitation can be easily overcame, following the theory developed in Deliverable D8. We would then be able to set up a new set of experiments to compare the systems on non-ground theories.

5.4 Planning: further development (ICSTM-DIPISA)

Aim. The aim of this experiment has been to explore a variant of the KGP model to allow an improved planning behavior of computees. We are not competing with state of the art planners, but making the KGP model applicable to planning problems. The variant of KGP uses a different tree representation of *Goals* and *Plan* in the state of the computee, whereby actions admit children (their preconditions). The variant also combines the Plan Revision (PR) and Goal Revision (GR) transitions within a single State Revision (SR) transition, as given in D13. Furthermore, the variant uses a simplified cycle theory. Finally, the variant relies upon a specified order amongst the proof rules of CIFF.

Formally, the variant of the KGP model described here has been given in [48].

CIFF 2.0 (triggering off)	Time [ms]	54.150	52.120	95.530	44.260	94.270	45.070	44.690	49.870	101.340	101.560	68.286	583,6
	answer	N	N	**Y**	N	Y	N	N	N	Y	Y		
CIFF 2.0 (triggering on)	Time [ms]	150	180	130	30	170	40	50	90	160	170	117	1
	answer	N	N	**N**	N	Y	N	N	N	Y	Y		
ASLD(N,IC)	Time [ms]	1.170	1.380	3.310	2.120	2.510	2.190	2.220	2.370	2.270	2.290	2.183	18,7
	answer	N	N	Y	N	Y	N	N	N	Y	Y		
Correct answer		N	N	Y	N	Y	N	N	N	Y	Y		
Query		*go 1*	*parked 1*	*neg(fuel) 3*	*neg(fuel) 11*	*go 15*	*neg(go) 15*	*parked 15*	*go 30*	*neg(go) 30*	*parked 30*	Average time	Ratio (over CIFF 2.0)

Table 2: Performance comparison for CIFF and ASLD(N,IC) supporting TR

Parameters considered and varied. The parameters considered for the new variant of the planning capability are as follows:

State Representation, Plan Introduction Transition and Action Selection Function
We have considered a variant of the computee state, whereby actions may admit children (and descendants). The children are goals, and they are the preconditions of the actions, generated by the $\models_{pre}$ capability, and descendants are the sub-goals of a goal. The new tree structure is enforced by the *Plan Introduction (PI)* transition, which is modified The modification of PI amounts to the new definition of $Subg(G_i)$ in part (i.2), which becomes:

(i.2) or $\mathcal{G}_i s \neq \perp, \mathcal{A}_i s \neq \perp$ and

$$Subg(G_i) = \{\langle l[t], G_i, T\rangle | (l[t], T) \in \mathcal{G}_i s\} \cup \{\langle l[t'], A_i, \{t = t'\}\rangle | KB, a[t] \models_{pre} C \wedge l[t] \in C\}, \text{ and}$$

$$Pplan(G_i) = \{\langle a[t], G_i, C, T\rangle | (a[t], T) \in \mathcal{A}_i s \wedge KB, a[t] \models_{pre} C\}.$$

Recall that in the original KGP model, preconditions are either inserted into the tree as siblings of actions, if $\models^{\tau}_{plan}$ does not already plan for them, or they do not appear at all in the tree resulting from planning, if $\models^{\tau}_{plan}$ has already planned for them, in which case any plan for the precondition will be added to the tree via siblings of the action. Our modification of the KGP model allows a finer control over the selection of actions to be executed within the Action Execution (AE) transition, in that actions whose preconditions have not been planned for and have not been observed to hold cannot be executed. This prevents executing actions prematurely. The *action selection function* c_{AS} has been modified accordingly, by adding an extra condition.

In addition to this modification, we have considered, in our experiments, a simplified version of the computee state. In particular, no reactive goals have been taken into account (since the Reactivity transition is not considered).

Revision Transitions
We have merged the revision transitions of the KGP model, i.e., GR and PR, into a new single revision transition SR, as documented in D13.

Cycle Theory
We have considered a simplified cycle theory, by limiting attention to the transitions:

Passive Observation Introduction (POI)
State Revision (SR)
Plan Introduction (PI)
Action Execution (AE)

Basically, preferential reasoning with the new cycle theory would amount to the following *imperative control*: first apply POI, then apply SR, then, if there exists some goals to plan for, apply PI, otherwise, if there exist some actions to execute, apply AE, otherwise, restart by applying POI.

Order of CIFF rules

We have found the following order amongst CIFF proof rules most effective for the planning experiments:

1. Constraint Checking and Case Analysis for Constraints
2. Splitting
3. Unfolding (for Atoms)
4. Propagation
5. Unfolding (within Implications)

Knowledge bases. We have experimented with a variant of KB_{plan}, without the integrity constraint

$$happens(A,T) \wedge precondition(A,P) \Rightarrow holds\_at(P,T)$$

We have incorporated checking and enforcing this integrity constraint into the PI transition (see the addition in condition (i.2)). By removing it from KB_{plan}, we have forced partial planning for goals, in the sense of not planning for preconditions of actions in the same planning phase leading to the introduction of these actions.

Our main experimentation domain has been the well-known, standard blocks-world domain for planning, where actions mv_table and mv have associated rules:

$$\begin{array}{lcl}
initiates(mv\_table(X),T,on\_table(X)) & & \\
initiates(mv\_table(X),T,clear(Y)) & \leftarrow & holds\_at(on(X,Y),T) \\
terminates(mv\_table(X),T,on(X,Z)) & \leftarrow & holds\_at(on(X,Z),T) \\
initiates(mv(X,Y),T,on(X,Y)) & & \\
initiates(mv(X,Y),T,clear(Z)) & \leftarrow & holds\_at(on(X,Z),T), Y \neq Z \\
terminates(mv(X,Y),T,clear(Y)) & & \\
terminates(mv(X,Y),T,on\_table(X)) & & \\
terminates(mv(X,Y),T,on(X,Z)) & \leftarrow & holds\_at(on(X,Z),T), Y \neq Z \\
precondition(mv\_table(X),clear(X)) & & \\
precondition(mv\_table(X),\neg on\_table(X)) & & \\
precondition(mv(X,Y),clear(X)) & & \\
precondition(mv(X,Y),clear(Y)) & &
\end{array}$$

Runs executed. We have run a series of experiments with CIFF for implementing $\models^{\tau}_{plan}$, given a number of instances of the blocks-world domain, including the Sussman anomaly. The experiments suggested the previously given ordering of CIFF proof rules as the most effective.

We have revised and simplified the SOCS-iC part of the PROSOCS platform to implement the new transitions SR and PI and the simplified cycle theory described earlier. We are experimenting with this implementation.

Evaluation. We have tested the overall implementation with very simple examples. These experiments aimed at evaluating the scalability of our planning technique. For this, we have targeted the task of building a tower composed of a growing number of blocks in the well-known blocks-world domain. The initial situation is that all the blocks are on the table. E.g. , for the test with 3 blocks a, b, c we have as initial situation

$$holds\_initially(on(a, table))$$
$$holds\_initially(on(b, table))$$
$$holds\_initially(on(c, table))$$

and as goal that of building a tower with a on top of b and b on top of c:

$$holds(on(b, c), 12)\ \ holds(on(a, b), 12).$$

For the experiments, we have adopted as the time in our goals $4 * N$, where N is the number of blocks involved. In the simple case above we have three blocks and so the goal time is 12. Indeed, our planning cycle takes 4 time-units to complete itself, and considering that each cycle plans only for a goal and it executes only one action, $4 * N$ is a good approximation for the number of time-units needed by the planner to achieve the goals.

Being this only a scalability test we do not consider observations or any action by other computees. The initial KB_0 is thus empty.

The experiments were run on a Mobile Intel Pentium IV - 1.7 Ghz, 256Mb DDR SDRAM, and had the following outcomes:

# of Blocks	**Goal Time**	**Execution Time (msec.)**
3	12	1833
5	20	9564
8	32	62300
8	40	100494
10	40	158839
15	60	836954
20	80	3066217
40	160	$> 1\,day$

In order to fully evaluate the implementation, and the overall approach to planning, we aim at considering further examples (both in the blocks-world setting and in other settings) in a dynamic settings (where KB_0 changes over time due to exogenous actions and observations. We also aim at comparing our planning technique with other planning systems (see below).

Related work. We are studying relationships between the planning system described in this section and other existing planning systems, notably planning with the A-system of [42]. We hope to have some comparative evaluation soon.

5.5 Improving the $\mathcal{S}$CIFF proof procedure (DIFERRARA)

Each $\mathcal{S}$CIFF computation produces a search tree whose *depth* and *breadth* determine the total number of nodes, and thus the time needed to explore the (whole) tree. Since the proof tree is explored by $\mathcal{S}$CIFF in a depth-first fashion, the depth of the tree and the *size* of a single node define space requirements. For both time and space, the worst case is when each branch leads to violation, since in this case the whole tree is explored.

Input parameters have been identified among those which influence the three factors above, namely, *depth*, *breadth* of the search tree, and *size* of nodes.

Intuitively, the *depth* of the search tree depends on the total *number of messages exchanged* within a society. This parameter can be varied, and incremented in particular, either by (*i*) having the same computees repeat more times the same interaction; or (*ii*) increasing the number of computees participating a society; or (*iii*) having "longer" interactions, where each compliant run is made up of different messages, increasing in number.

The *breadth* of the search tree, instead, is influenced by both the number of disjuncts in the head of social integrity constraints, and the alternative branches arising in several of the $\mathcal{S}$CIFF rewriting steps (see Deliverable D8 [34]).

For instance, in the *Fulfillment E* transition, the fulfillment of an **E** expectation leaves a choice open for non-unification between the expectation and the event that would fulfill it. In some cases, avoiding this branching does not change the behavior of $\mathcal{S}$CIFF with respect to fulfillment and violation: for instance, when a positive expectation can be fulfilled by only one event, the non-unification branch cannot be one of success. In such cases, cutting the non-unification branch is safe, and saves a considerable amount of computation space and time. We call such a $\mathcal{S}$CIFF behaviour *f-deterministic* meaning that it behaves deterministically on *Fulfillment*.

We let the user decide for which expectations $\mathcal{S}$CIFF should adopt the *f-deterministic* behaviour (or, in other words, which expectations will be *f-deterministic*), by means of one or more `fdet/1` directives, whose argument is a term T: all expectations that are instances of T will be treated as *f-deterministic*. Of course, it is possible to make all expectations *f-deterministic* by a directive such as `fdet(e(X,T))`. For simplicity, we will refer to $\mathcal{S}$CIFF version with this directive as the *f-deterministic* version of $\mathcal{S}$CIFF; the *f-non-deterministic* version of $\mathcal{S}$CIFF will adopt, instead, no `fdet/1` directive.

Aim. The aim of this class of experiments is to support us in our efforts to improve the efficiency of $\mathcal{S}$CIFF, the proof procedure of the society infrastructure for compliance verification, and evaluate performance.

Parameters considered and varied. In this experiment, we consider the variation of the following parameters:

1. $\mathcal{S}$CIFF version (*f-non-deterministic* vs. *f-deterministic*, as defined above);

2. interaction protocol (one version with a disjunction in the head of a social integrity constraint vs. one version with no disjunctions, with alternative expressed by means of variables with domain);

3. number of messages exchanged.

In particular, we measure the computation time for sequences of interactions exemplified by auctions with different numbers of bidders in the two following implementations of the protocols:

1. *f-non-deterministic* $\mathcal{S}$CIFF, protocol with disjunction (which we call the *first setup* of $\mathcal{S}$CIFF and protocol);

2. *f-deterministic* $\mathcal{S}$CIFF, protocol with no disjunction (which we call the *second setup* of $\mathcal{S}$CIFF and protocol).

Knowledge bases. In both cases, the goal is *true*, and the *SOKB* is reported in the document entitled *Experiments on Combinatorial Auctions* [6], annexed to this deliverable.

In the first case, we have replaced the fourth social integrity constraint reported in [6], by the constraint:

```
H(tell(B,A,bid(ItemList,P),Anumber),Tbid) /\
H(tell(A,B,openauction(Items,Tend,Tdeadline),Anumber),Topen)
--->
E(tell(A,B,answer(win,B,Itemlist,P),Anumber),Tanswer) /\ Tanswer >
Tend /\ Tanswer < Tdeadline \/
E(tell(A,B,answer(lose,B,Itemlist,P),Anumber),Tanswer) /\ Tanswer
> Tend /\ Tanswer < Tdeadline.
```

In the second case, the social integrity constraints are exactly those reported in [6]. In particular, the fourth IC_S is the following one:

```
H(tell(B,A,bid(ItemList,P),Anumber),Tbid) /\
H(tell(A,B,openauction(Items,Tend,Tdeadline),Anumber),Topen)
--->
E(tell(A,B,answer(Answer,B,Itemlist,P),Anumber),Tanswer) /\
Tanswer > Tend /\ Tanswer < Tdeadline /\ Answer::[win,lose].
```

where the alternative is expressed by the `Answer` CLP variable having the domain `[win,lose]`, rather than by a disjunction in the head of the integrity constraint, as in the first case.

Runs executed. The protocols have been run by varying the number n of bidders, in two different cases.

- In each run of the first case:

  1. the auctioneer sends an `openauction` message to each of the n bidders;
  2. each of the n bidders places a `bid`;
  3. the auctioneer issues a `closeauction` message to each of the n bidders;
  4. the auctioneer notifies each of the n bidders with either a `win` or a `lose` message,

  thus resulting in $4n$ total messages exchanged.

- In each run of the second case, the last notification is missing, thus resulting in a violation of the protocol and $4n - 1$ total messages (the auctioneer replies to all bidders, except one).

The experiments have been run on a PC with a 2 GHz Pentium IV CPU, 512 MB of RAM, Linux 2.4.18, glibc 2.2.5 and SICStus Prolog 3.10.1. Reported times are in seconds.

In case of fulfillment (see Table 3), the first setup of $\mathcal{S}$CIFF and protocol seems to scale well with the number of bidders and, in fact, it achieves better execution timing than the

f-non-deterministic, disjunction		*f-deterministic, domain*	
Bidders	Time(sec.)	Bidders	Time(sec.)
5	1	5	1
10	1	10	1
15	2	15	2
20	3	20	6
25	4	25	8
30	6	30	10
35	9	35	15
40	10	40	18
45	12	45	23
50	21	50	30

Table 3: Combinatorial Auction case 1: Fulfillment

f-non-deterministic, disjunction		*f-deterministic, domain*	
Bidders	Time(sec.)	Bidders	Time(sec.)
3	7	3	0
4	55	4	0
5	-	5	0
10	-	10	1
15	-	15	3
20	-	20	4
25	-	25	7
30	-	30	10
35	-	35	14
40	-	40	17
45	-	45	22
50	-	50	26

Table 4: Combinatorial Auction case 2: Violation

second. This is basically due to the fact that the chosen setup of interactions directly leads to a successful $\mathcal{S}$CIFF derivation, and only one branch of tree is explored.

In the case of violation (see Table 4), however, the first setup of $\mathcal{S}$CIFF and protocol explodes for a very small number of bidders. The run with 5 bidders was suspended since this did not reach the answer of violation after several minutes of computing time; no further run was performed with a higher number of computees, which would have made things even worse. The symbol $-$, therefore, represents time explosion. The second setup, instead, scales very well also in case of violation. In this case, a CLP(FD) solver, written in CHR [31], directly manages the two alternative values for variable `Answer`.

Evaluation. The difference between the two setups of $\mathcal{S}$CIFF and protocol becomes apparent in the worst case (i.e., the case of violation) when the whole search tree is explored. With the first setup, the choice points left open in case of fulfillment and the disjunctions in the head of the integrity constraint make the number of nodes in the proof tree explode even for small number of bidders. With the second setup, instead, the tree has only one branch, and is thus explored in a reasonable time when the number of bidders increases.

5.6 Correlation between events and compliance checking (UNIBO)

Aim. The aim of this experiment is to investigate the correlation between the total amount of events in a computee interaction and the time needed by $\mathcal{S}$CIFF to check the compliance of the interaction to the protocols. The test aims in particular to establish both the time needed to find a solution, as well as to determine the computational limit using a particular hardware configuration, taken as reference for all the tests.

Parameters considered and varied. We have varied only the number of messages, taking as output the time spent by $\mathcal{S}$CIFF to produce an answer. The test has been conducted using the "no GUI" version of SOCS-SI: the overhead introduced by the Java GUI will be discussed in a further test. Also only the *f-non-deterministic* version of the $\mathcal{S}$CIFF has been used: a comparison between the two versions of $\mathcal{S}$CIFF was shown in previous test.

Knowledge bases. We decided to abstract from any "real" scenario or use-case, in order to obtain results, as much as possible, independent of particular cases.

In particular, we decided to perform this test by simulating an increasing number of computees, each one uttering only one message. Since the $\mathcal{S}$CIFF computing time depends also on the complexity of the protocol, we adopted a very simple one, made up by only one IC:

```
H(tell( A, B, aPerformative( aContent), D), T) --->
E(tell( C, B, aPerformative( aContent), D), T1)
/\ T1 > T.
```

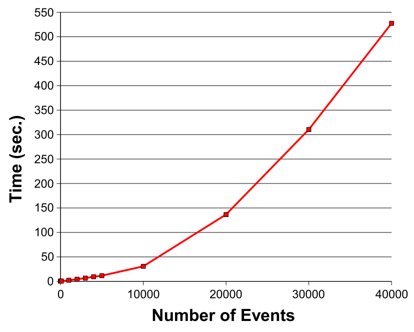
Such a protocol minimizes the computational cost given by the complexity of the protocol itself, since it generates only one expectation for each event; moreover, this expectation is fulfilled by the following event, hence the expectation set is always kept at a minimum size.

Computees interaction have been simulated using event histories like the following one, where N has been increased in the different runs:

```
tell([s0], d, agent1, receiver, aPerformative, [aContent], 1).
tell([s0], d, agent2, receiver, aPerformative, [aContent], 2).
...
tell([s0], d, agentN, receiver, aPerformative, [aContent], N).
```

Runs executed. All the runs have been executed 100 times, and their average value is reported in the following table. Table 5 and the companion chart show the variation of the computing time with respect to the number of events. Reported times are in seconds.

The PC used for the tests has a Pentium IV 2.8GHz CPU, 512MB of RAM and Windows XP.



Number of events	Average Time (sec.)
1	0,41
10	0,41
100	0,57
1000	2,23
10000	30,71
40000	527,45

Table 5: Time spent elaborating histories of various length

Evaluation. It is immediately clear that performances of SOCS-SI get worse with a higher number of messages. In fact, the time requirements for elaborating messages do not increase linearly; rather, they tend to increase following a square factor, since each positive expectation is compared with each of the happened events.

The time needed to process a single event is not constant, but it grows along with the growth of the history itself. This behavior can be explained by the fact that SOCS-SI, in the actual implementation, keeps track (in central memory) of all the events happened, as well as of all expectations (fulfilled, violated or pending) and of Partially Solved Integrity Constraints (named PSIC, for short, in Deliverable D8 and D9). This leads to an increment of the memory needed, and of course to an increase of time cost, to access a bigger amount of data. Then, each new event must be "checked" against a bigger set of PSIC.

6 Testing with scenarios

6.1 $\mathcal{P}_{SCENARIOS}$: Identified parameters for testing with scenarios

We have identified some notable parameters to be considered in the experiments with the chosen scenarios, in addition to those concerning PROSOCS, for both individual computees and societies of computees.

$\mathcal{P}_{SCENARIOS}$ **computee side.** The main parameter of variation of individual computees for experiments on scenarios is the *behavior profile* of the computees, as this is given to them by their *cycle theories*. This links to the formal properties of Identification of Profiles of Behavior that we are examining (see devoted section in Deliverable D13). Another, somewhat related parameter, for individual computees is their *social attitude or personality* as captured in Goal Decision policies and components of this such as their private negotiation policies.

Further parameters to be varied are represented by the various KBs for the single capabilities of the individual computees, e.g., planning, reactivity, goal decision, etc.

The main output parameter of evaluation or comparison (from the viewpoint of individual computees) of different experiments is the degree of *individual welfare* achieved as this is formalized and studied in WP5 (see devoted section in Deliverable D13). In particular, different notions of welfare relating to the computees individual preferences (e.g., its goal decision preferences) and to the society protocol can be examined (separately or combined together).

$\mathcal{P}_{SCENARIOS}$ society side. Beside $\mathcal{P}_{PROSOCS}$ parameters, in the experimentation pertaining scenarios we have varied the kind of society itself, by changing the *knowledge* of the society, its *goals* and its *interaction protocols.*

6.2 Combinatorial Auction scenario

In the vision of computational infrastructures available globally and able to provide services of different kinds (e.g., for communication, co-operation and mobility, resource usage, security policies and mechanisms, etc.,), a central issue is that of resource and task allocation. To this end, particularly suited tools are market-based negotiation protocols and theories, such as those related to auctions and the increasingly popular combinatorial auction mechanism. Combinatorial auctions [22] are a good candidate to address issues of service programmability, resource distribution and management, scalability, and distribution transparency, all central topics in the global computing perspective.

We therefore propose to test the SOCS infrastructure on a real world e-commerce problem, such that of Combinatorial Auctions. Combinatorial Auctions represent a form of auction where bidders can bid on a combination of items (or services).

This scenario is also chosen, besides its relation Global Computing features mentioned above, because:

- it exhibits all the SOCS required features (i.e., the need for programming KBs of single computees and rule their interactions by social constraints);
- it integrates all the PROSOCS components (e.g., reactivity, goal decision, Social Integrity Constraints) and represents a suitable scenario for their test;
- it needs to also incorporate additional components (i.e., an external solver, see below) by using PROSOCS objects.

This kind of auction has not found in the past many applications because, although it is a very powerful and expressive way of buying and selling goods, the task of determining the winning bids is NP-hard.

Recently, sophisticated solvers and algorithms have been developed (see for example [61]) to efficiently solve the winner determination problem. Therefore, combinatorial auctions are becoming a standard application in e-commerce, and they are a significant scenario to be experimented upon with PROSOCS.

We consider in this experiment the *single unit reverse auction* where each item is distinguishable (and therefore named *single unit*), and the auctioneer wants to buy a set of goods while bidders sell items (and therefore named *reverse*). Intuitively, the auctioneer proposes a set of items M that she/he wants to buy and each bidder proposes a bid composed of a set of items (a subset of M) and a price she/he is willing to pay if her/his bid is considered as winning. After the auction is closed, the auctioneer is supposed to solve the Winner Determination Problem by choosing a set of winning bids covering all items at the minimum cost.

For this problem we have implemented an Auction Solver, described in detail in [36], embedding a portfolio of algorithms for solving the problem. In addition, the Auction Solver is enhanced with an automatic selection of the most efficient algorithm in the portfolio given the auction instance to be solved. The selection mechanism is described in [37]. A more detailed description of the scenario can be found in [6], annexed to this deliverable.

In the following, we report about the experiments we performed in this scenario. We performed experiments on single computees, on the society alone and an integrated experiment where computees and the society are tested together, exploiting the Medium.

As far as computees are concerned, the main experiments carried out aimed at testing whether computees are flexible enough to cope with *external objects* providing services to the computees themselves, and to exploit the different profiles of behaviour available, and the reactive capability, in particular, among others.

Concerning the society alone, we performed experiments on the efficiency and the scalability of the proof procedure by using files containing conforming histories. In addition, we performed other tests using the e-mail system as the communication layer, instead of the Medium. A new component has been implemented, for the purpose of letting SOCS-SI to easily interface with the email where human agents exchange messages through properly formatted email messages. We are currently conducting two different groups of experiment: in the first group computees are simple reactive programs using email accounts; in the second group human users are participating to a combinatorial auction, while SOCS-SI checks for compliance to the protocol of this human agents. Experiments using the email are not intended to test scalability, but the flexibility of the $\mathcal{S}$CIFF proof procedure to deal with delays due to the communication layer used.

Experiments on the society alone are currently carried on different scenarios where combinatorial auction come into play. In particular, we experimented:

- single auction: this scenario models a traditional auction where a set of bidders post their bids on combinations of items and the auctioneer decides which bids win.
- an auction plus the buying/selling step: this experiment involves an auction and the next step where items that are promised in the auction are indeed sold and bought at the corresponding price. We used the NetBill protocol for this step.
- double auction: the third experiment implements two auctions. Indeed, when an auction is opened and the bidders have access to the list of items the auctioneers posts, some of them can decide of opening a second auction to collect more items and perform more appealing bids.

Finally, we run an integrated experiment which concerns only the single auction, and we put together the society and the computees using the Medium as communication layer. The integrated example will be shown during the demo to be given at the next review meeting.

6.2.1 Modelling Computees for Combinatorial Auctions (ICSTM-CITY)

Aim. The aim of our experiments on modelling both an auctioneer and bidders using the PROSOCS platform has been to verify that our framework can support the basic communication processes required to run a combinatorial auction. The main capability that needs to be programmed in this scenario is reactivity (and, to a lesser degree, also planning). These experiments have also been a first application of the extension of PROSOCS with the concept of an *external object* to carry out specific tasks for a computee (in this case, determining the winners in a combinatorial auction and collecting a set of bids using a simple object that can act as a "notepad"). That is, these experiments were also aimed at testing the functionality of external objects within PROSOCS.

Parameters considered and varied. As concerns $\mathcal{P}_{PROSOCS}$, in theory, we could have chosen between either mapping certain tasks to external objects or writing appropriate planning or reactivity knowledge bases for carrying out these tasks. In the combinatorial auction scenario, we have opted for using two external objects: the ILOG auction solver and the "notepad" [46, 6]. In the case of the auction solver it seems unreasonable to attempt to implement a sophisticated optimization tool in terms of the abductive event calculus (so, in fact, in practice this is not really a parameter that could be varied). In the case of the "notepad", we have found this a very convenient means of programming the auctioneer, but other solutions may be possible as well. As concerns $\mathcal{P}_{SCENARIOS}$, parameters to be varied concern the types of bidding computees used (characterized by the choice of knowledge base for reactivity), the number of computees participating in an auction, and the number of items on auction. We have experimented with a small number of different types of bidders. We have also tested the platform for different numbers of bidders and different numbers of bids per bidder.

Knowledge bases. We have designed generic knowledge bases for both the planning and the reactivity capability of a computee acting as an *auctioneer*. A computee equipped with these knowledge bases can open an auction (assuming it has a goal to do so) by sending out an appropriate message, it can check incoming bids for validity (by verifying that the bidder has in fact been invited to bid, that the set of items is a subset of those on auction, and that the specified deadline has not yet passed), and it can collect bids deemed valid for processing after the end of the auction. Collecting bids is achieved by "writing" valid bids to an instance of the "notepad" object described in [46]. Finally, the reactivity knowledge base includes rules that allow the computee to retrieve the list of collected bids, to forward it to another external object implementing a combinatorial auction solver, and to disseminate the result received from that solver amongst the bidders. The knowledge bases are generic in that they can support different types of auctions (normal or reverse combinatorial auctions; with or without time windows). The knowledge bases are described in detail in [6], annexed to this deliverable. To date, we have also written a number of simple reactivity knowledge bases for computees acting as *bidders* in a combinatorial auction. These encode either hard-wired bids that will be sent out as soon as an auction is opened, or very simplistic strategies (such as bidding in any auction that offers or asks for a particular item, with a fixed price) [6]. Implementing more sophisticated bidding strategies is possible in principle, but would require substantial further work on choosing an appropriate knowledge representation.

Runs executed. To test the scalability of the system, we have run experiments with different numbers of bidders and different numbers of bids per bidder. As the main computational burden lies with the auctioneer, these runs test, in particular, the scalability of the computee implementing the auctioneer. The overall runtime for 9 different settings (with 2–4 bidders and 2–4 bids per bidder concerning 6 items on auction) are reported in [6], annexed to this deliverable..

Evaluation. Our experiments show that PROSOCS (and its integration with SOCS-SI) is a suitable platform to deal with the communication requirements for running a combinatorial auction. In particular, our experiments have demonstrated the feasibility of the *external objects* approach, both for a very simple object (the "notepad") and for a complex and sophisticated piece of third-party software (the ILOG solver implementing the winner determination algorithm [6]). While these results are very encouraging, we have also encountered problems with the scalability of PROSOCS for this type of application. This is not surprising given the prototypical character of the platform and the generality of our approach.

6.2.2 Experimenting the Auction Solver (UNIBO)

Aim. In the Combinatorial Auction scenario we have to cope with a complex combinatorial optimization problem: the Winner Determination Problem. Having an efficient, scalable and flexible tool that solves this problem is crucial for the efficiency of the overall system.

In this experiment, we exploit an Auction Solver implemented in ILOG solver [38] suitably wrapped in to Java. The Auction Solver is so efficient that it gives the possibility of scaling the auction dimension, and test the performances of the SOCS social infrastructure.

Parameters considered and varied. We describe here the experiments done on the Auction Solvers. The complexity of the problem to be addressed depends on the number of bidders attending the auction. If the number of bidders grows, the task of determining the winning bids becomes more and more time consuming, being NP-hard.

Knowledge bases. In this case, since we are experimenting an external object, the society knowledge base is empty, being the Winner Determination Problem solved by the auction solver. The solver is described in detail in [36].

Runs executed. The Auction Solver has been extensively experimented and results are extremely encouraging. We run tests on the Auction Solver for an increasing numbers of bids. Qualitative results are reported in [6].

Evaluation. The Auction Solver implemented in ILOG is extremely efficient since it scales up to 1000 bids within 2 seconds. We will see in the next section that slower performances are achieved by the conformance checking of the society protocol (around 30 seconds for checking messages exchanged in auctions with 50 bidders). However, even if the components have different performances, from the testing, we can conclude that both components can be used in a real combinatorial auction scenario since the time limits required for answering are much larger than the sum of the answer times of the components.

6.2.3 Modelling Societies for Combinatorial Auctions (UNIBO-DIFERRARA)

Aim. The motivation for performing tests on the society alone includes the need to show that the social infrastructure (and its implementation, SOCS-SI) can successfully deal not only with toy problems, but also with real applications.

We also show that, beside computees, we can also introduce *external objects* providing a service to the society or to a single computee (as shown in the example discussed in Section 6.2.1 above).

Parameters considered and varied. We describe here the experiments done on the conformance checking to the society protocol and on the Auction Solvers. Also in this case, the complexity of the problem to be addressed depends on the number of bidders. If the number of bidders grows, as the number of messages exchanged proportional to the number of bidders, then the computational cost of the compliance verification increases.

In addition, we varied also the protocol to be tested. In particular we defined three variants of the scenario so as to test the flexibility of the proof and the expressivity of the language.

The three protocols implemented are the following:

- **Protocol 1** single auction: this protocol models the interactions between the auctioneer and the single bidders during a traditional auction. The protocol checks the messages exchanged between the open auction and the communication of the auctioneer to all the bidders if they win or lose.
- **Protocol 2** an auction plus the buying/selling step: this protocol goes one step further with respect to the previous one. In fact, it models interactions ending when all items belonging to the winning bids are indeed sold and bought at the price decided in the auction. For this purpose, we used a part of the NetBill protocol [20].
- **Protocol 3** double auction: this third protocol models, beside a traditional auction the possibility of having a bidder opening a second auction for collecting items to be proposed in its own bid. Therefore, we have two auctions going on, the second ending before the end of the first.

The integrity constraints representing these protocols are reported in the annexed document [6].

Knowledge bases. The society knowledge base for these experiments and the auction protocol as IC_S are reported in [6], annexed to this deliverable.

Runs executed. To test the scalability of the society alone, we performed tests using files containing conforming histories. In this way, we can test the efficiency of the proof without the delays introduced by exchanged messages.

The first experiments on which we already have results is the one on a single auction. Results show the time spent to check the conformance of exchanged messages to the society protocol in the presence of an increasing number of messages, which in turn is proportional to the number of existing bidders. The version of $\mathcal{S}$CIFF used is the *f–deterministic* one, which scales better with an increasing number of messages. Results are reported in [6].

In order to test the society in a realistic, but human environment, we also implemented an e-mail system as communication layer, in alternative to the Medium. In this setting, human users exchange messages through properly formatted emails. In this way we can make both computees and human users bid together in an auction, while SOCS-SI checks for compliance to the protocol of utterances made by these agents. This does not affect the underlying SOCS model, but of course extends the applicability of PROSOCS to human agents.

Evaluation. The scalability of the *f–deterministic* version of the proof is quite good, even if far from the Auction Solver performances. Although there is room for improvement in the implementation of the proof, for the moment it is efficient enough for performing our tests.

From the experiments we did using the email system, we can say that although there are delays introduced by the email system, the proof smoothly deals with them and works properly.

6.2.4 Integrated example (ICSTM-UCY-CITY-UNIBO-DIFERRARA)

Aim. After testing the single components separately, we have also tested the scalability of the overall system by putting together computees exchanging messages with the society infrastructure checking for computees communication conformance. The experiments have been done on the traditional combinatorial auction since the aim of this test is to check the real integrability of the components and their interactions. We have encountered and solved some problems related to the integration of different components and problems related to deadlines. We ended up with a stable system, but experimentation is still ongoing. One integrated sample demo will be shown at the next review meeting in Edinburgh on April 4th.

Parameters considered and varied. In the integrated experiment we have done and plan to develop further, we have changed the number of bidders and the number of bids per bidder.

Knowledge bases. The knowledge bases of the society and of the computees are reported in [6]. It is important to notice that the computees exploit in this experiment only their reactive capability. In the future, testing the planning capability for defining more sophisticated bidding strategies will be considered.

Runs executed. We have run a set of experiments with 2, 3 and 4 bidders with 2, 3 and 4 bids per bidder.

Evaluation. Results show that the components integrated well. Up to now, only small examples (with few bidders) have been solved. To really test the scalability of the platform and its performance for the integrated case, further experimentation is needed, indeed, and it as been planned already as work in progress.

6.2.5 Related Work

Combinatorial Auctions are increasingly studied since they have the advantage that bidders can bid on combinations of items. This leads to more efficient allocation than traditional auctions where the bidders valuations are only additive. The drawback is that evaluating bids and determining the winning bids is a NP-hard problem. However, there are different systems and methods to solve a combinatorial auction in an efficient way. The methods used to solve the problem exploit dynamic programming techniques [56], approximate methods that look for a reasonably good allocation of bids [32, 60], integer programming techniques [19, 53], search algorithms [62].

An aspect which plays an important role in auctions in general is negotiation. Negotiation is one of the main research areas in distributed systems. The area of negotiation is broad and applies to different scenarios, such as for instance multi-agent systems [39, 55]. In most cases agents need to negotiate because they operate in environments with limited resource availability. For example, one-to-many negotiation is used for auctions, where auctioneers and bidders reach an agreement on the cost of the items on sale.

An interesting example of system similar to ours is ISLANDER [64], a tool to specify protocols in a system ruled by electronic institutions that has been applied to a Dutch auction (and other scenarios). Their formalism is multi-levelled: agents have *roles*, agents playing a role are constrained to follow *protocols* when they belong to a *scene*; agents can move from a scene to another by means of *transitions*. As in various works, protocols are defined by means of transition graphs, in a finite state machine. Our definition of protocols is wider than finite state machines, and leaves more freedom degrees to the agents.

More related work pertaining learning applied to (combinatorial) auctions is mentioned in [6].

Relevance to Global Computing. With the GC vision in mind, we found it important to demonstrate experimentally that the models and tools developed in WP1-WP4 are flexible enough to address aspects such as heterogeneity, resource achievement, and scalability, often present in open environment. This was one of our main criteria when we had to choose experimental settings and scenarios. Combinatorial auctions are an example. They can be used to facilitate resource achievement, and can be tested by varying a number of parameters significant in the GC vision, like profiles and number of computees involved in the resource exchange process. Besides, the auction mechanism can itself be considered as a service, and the possibility to define different protocols for resource exchange is a demonstration of programmability of services. We have considered a set of protocols for various auction mechanisms (like English auction, combinatorial auction and others). The intuitive and easily readable specifications of such protocols demonstrates the flexibility of a fully declaratively programmable tool. This, together with the guaranteed verifiability of interaction in all our test cases, demonstrates that exporting the SOCS models into a larger scale is something feasible. The diversity of tested protocols in the case of combinatorial auctions demonstrates the SOCS model's ability to help the integration of an open and global computing reality, where entities can live with pre-existing interaction mechanisms and a new associated social semantics. Incrementally, diverse (and possibly incompatible) implementations can in this way move towards integration and global communication.

6.3 Experimenting with the Music for Beer scenario

This scenario was first proposed by Küngas and Matskin [43], to show agent cooperative problem solving achieved by use of linear logic and partial deduction. It is fully described with several variations in the document *Further Examples for the functioning of computees* [23], annexed to Deliverable D9). That document contains also the detailed knowledge bases and cycle theories of the computees and the society protocols used in variations.

Experimentation for this scenario was partially done, along the first sixth months of third year. With respect to task allocation, unfortunately UCY was in a difficult position since it has lost its RA and despite efforts it was difficult to find a replacement. Therefore, this scenario was only partially tackled. However, we report here, the work done along the first six months of the third year of the project.

6.3.1 Collaborative Achievement of Individual Goals and Resource Allocation (UCY-UNIBO)

Aim. The experiments concern the investigation of collaborative (operated jointly) problem solving by computees to achieve their individual goals. It aims to compare the performance of computees by examining different designs of the computees and different external circumstances. In particular, variations that link the computees with their societies will be examined.

Apart from testing the SOCS models themselves, we also aim at showing how this differs and extends existing approaches, e.g., [43], in several aspects, including *autonomy*, as the amount of information shared by computees is tuned by the policies, and computees are not necessarily cooperative, *planning capabilities*, *modularity* of knowledge representation, and finally the ability to reason on dynamic and contextual information.

Parameters considered and varied. The main parameters to be varied are of two types (*i*) different resource allocation and collaborative achievement problems and (*ii*) different types of computees operating. More specifically these parameters are:

- external circumstances of time and communication;
- social expectations of the society;
- behavior profile of the computees;
- social attitude and negotiation policies of the computees
- number of computees and tasks involved.

Evaluation criteria. The main evaluation criteria for these experiments will be: (*i*) the distribution of the individual welfare (linking to the properties, see deliverable D13 [2]) achieved in each case and comparisons of these, (*ii*) conformance or not to social expectations for variations of the experiments that consider these and (*iii*) efficiency of operation of computees as we scale up their numbers.

Runs executed. We have implemented (using an earlier version of the PROSOCS platform) the standard scenario of Music for Beer and tested its appropriate behavior. Also a variation of the scenario where we simulate social expectations is tested showing the effect of these on the behavior of the computees.

Evaluation. Our initial limited experiments already point out the flexibility that the SOCS framework has in addressing such problems. But proper systematic experimentation was not carried on, due to lack oh human resources at UCY. This prevented us from carrying on all the planned experiments.

6.3.2 Related Work

There are several logic-based frameworks for agent negotiation which are related to our work on this experiment. As we have mentioned above the experiment has been inspired by the work of Küngas and Matskin's [43] who have studied how cooperative problem solving could be achieved with the use of linear logic and partial deduction. Our experiments show firstly the relative comparative ease with which such complex requirements can be implemented in PROSOCS and secondly to carry out a first evaluation of the difficulties in implementing further more reasoning and acting modules of the agent. In addition, our work differs from other work such as Küngas and Matskin's [43] in several aspects, including *autonomy*, as the amount of information shared by agents is tuned by the policies, and agents are not necessarily cooperative, *planning capabilities*, *modularity* of knowledge representation, and finally the ability to reason on dynamic and contextual information. All these additional aspects have been implemented in the demonstration example of Music for Beer.

Other related work includes [15] and [59] where the authors focus on the ability of agents to produce dialogues, on protocol compliance, and on the ability to solve resource reallocation problems. However, their architecture, focussing on the reasoning needed to produce dialogues, is independent of other agent capabilities, such as planning. Moreover, differently from [59], our use of preference reasoning allows for a modular agent programming, which allows for combination of individual reasoning with social expectations.

Relevance to Global Computing. The Music for Beer experiment was designed to check how our KGP model of agency could accommodate several variations and extensions of agent distributed problem solving required by Global Computing. These include problems where the agents are heterogeneous, e.g., where some agents are cooperative and some are not. Similarly, we can accommodate issues that relate to adopting social expectations and adapting the negotiation between agents to different circumstances, e.g., the type of request or the relative roles of the agents involved, by exploiting the added flexibility of the private agent policies.

The main Global Computing characteristics that are illustrate through our experiments are the autonomy of computees to make their own decision in an open and changing environment, the heterogeneity of computees (cooperative or non co-operative), their ability to distribute their tasks and to adapt their decisions and behaviour in the face of new information e.g. as provided by the other computees. In particular the following criteria have been addressed by the experiment:

Adaptability: Adjustment where the computee decides to respond differently to requests depending on the needs it can currently satisfy and/or on the needs that it currently has for itself.

Partial Information: Conditional decisions where computees develop plans conditional on the assumption that a request for a need within the plan will be accepted and thus provided.

Distribution: Decentralization where a computee decides to achieve a goal with a plan which also involves other computees. The computee has thus distributed its task.

Heterogeneity: Overall Behaviour where the cycle theory and goal decision preference policies of a computee regulate its cooperative or not behaviour and the relative importance that a computee gives to requests from other computees.

Heterogeneity: Personality where the personality policy of the computee within its goal decision policy affects its decisions as to whether to accept or not a request to provide a need.

The relatively complex scenario of the experiment illustrates how the different capabilities of the computee are composed together in order to produce the enhanced functionality required by the global computing setting. They show the synthesis of the isolated components of the model together with the cycle theory and how it is possible to exploit different parts of the knowledge of the computee specified *modularly* in different components of the model.

7 Experimentation for properties

Our classification of verifiable properties of societies of computees is based on the catalogue of properties put forward in deliverable D13 [2], consisting of four parts: (*i*) properties of proof procedures; (*ii*) properties of individual computees; (*iii*) properties of the social infrastructure; and (*iv*) properties specifically related to protocol conformance.

It is important to distinguish between (formal) *properties* and (observable) *characteristics* of (societies of) computees. Formal properties are invariants which have been formally proven within WP5. Characteristics refer to properties which can only be observed dynamically, thus they can be also considered as "emerging" properties. Therefore, we consider as emerging a characteristic which can be dynamically observed.

Experimentation is not a suitable way to prove formal properties, but it can be used to observe characteristics of the behavior of an animated society and of individuals within the society which may be candidates for a formal proof. Also, experimentation can be used to disprove that formal properties hold. The idea goes back to D. Hume, and his work entitled *An Enquiry Concerning Human Understanding*, dated 1748.

To the purpose of experimentation, we have tried out some of the properties identified and verified in WP5 (see deliverable D13 [2]) to make predictions that can be tested by designed experiments.

For properties of individual computees, we have identified in deliverable D13 interesting profiles of behavior and characterized them first in terms of properties of the states of computees and then design cycle theories that will induce traces whose states have these properties. Experiments were done to confirm that such profiles are advantageous given specific conditions. In particular, we have identified a triple of individual computees to be experimented upon, the focussed profile of behavior (see Section 7.1.1), the cautious profile of behavior (see Section 7.1.2), and the punctual profile (see Section 7.1.3) among others.

As concerns properties specifically related to protocol conformance, in the following we report about experiments for on-the-fly conformance. According to the D13 classification, on-the-fly conformance is a property of an interaction (between two or more computees) with respect to a given society protocol. It assesses, at runtime, whether or not the interaction is legal according to the protocol. The experiments concerns the application of the $\mathcal{S}$CIFF proof procedure for this purpose to the Needham-Schroeder security protocol [52], see Section 7.2. We choose this protocol since it was used for several years, and later proved flawy by applying model checking techniques [18, 51]. On this same protocol, we have also experimented a proof theoretic approach for proving its flawyness.

In fact, a further issue has been investigated and experimented upon along this year of the project, in particular for properties of the social infrastructure. As described in D13, a proof theoretic approach for the automatic verification of properties of protocols has been proposed, and experiments have been done. To this purpose, we extend the $\mathcal{S}$CIFF proof procedure and show how it can be used to verify whether a property, expressed as an existentially or universally quantified formula, is a logical consequence of the Social Integrity Constraints of a given society. This approach has been applied to prove the good atomicity of the NetBill protocol, and to show the flawyness of the Needham-Schroeder protocol.

7.1 Experiments on Profile of Behavior

In deliverable D13 [2], we have identified interesting profiles of behavior and characterized them first in terms of properties of the states of computees, and then design cycle theories that will induce traces whose states have these properties.

In the following, we report on three experiments performed for confirming the expected behavior as determined by the engineered profile. Some of these profiles will be demonstrated at the next review meeting in Edinburgh on April 4th.

7.1.1 The Focussed Profile of Behaviour (ICSTM-UCY-CITY)

Aim. The aim of this experiment is to provide empirical evidence for the advantages of the focussed profile of behaviour discussed in D13 [2]. Recall that a focussed computee would be expected to be more successful than a computee that is not focussed in situations where both computees have (at least) two top-level goals that are mutually exclusive in the sense that executing a plan for the first goal will make any potential plan for the other goals infeasible.

Parameters considered and varied. We compare the behaviour of two computees: computee 1 is focussed, while computee 2 uses the normal cycle theory. In a nutshell, a computee is focussed iff any basic rule in its cycle theory that enables the PI transition has got an enabling condition that only allows for the selection of (*i*) a top-level goal if no other top-level goals have children and (*ii*) a non-top-level goal G if the top-level goal G' that is the ancestor of G is the only top-level goal with children.

Knowledge bases. We consider a scenario where both computees aim at achieving the two goals `g1` and `g2`. However, the plans for these two goals are not compatible. The domain-dependent part of KB_{plan} is as follows:

```
initiates(a1,_,g1).  precondition(a1,p).  initiates(a3,_,p).
initiates(a2,_,g2).  precondition(a2,q).  initiates(a4,_,q).
[happens(a1,T1),happens(a4,T2)] implies [false].
[happens(a2,T1),happens(a3,T2)] implies [false]. executable(_).
```

The domain-dependent part of KB_{TR} is defined accordingly. KB_{react} is empty.

Runs executed. We have run both computees, initializing them with the two goals `g1` and `g2`, to be achieved before time `20`. Both computees start by planning for `g1`, generating the action `a1` and the subgoal `p`. Then computee 2 is free to plan either for `g2` or for `p`, while the focussed computee 1 is forced to plan for `p` during the next PI transition. Only planning for `p` will lead to a feasible plan for `g1`, while planning for `g2` causes the integrity constraints in KB_{plan} to fire during future calls to planning (resulting in a failure).

We should stress that computee 2 *could* also plan for `p` and proceed in the same way as computee 1. However, in the concrete implementation, the top-level goal `g2` will be selected before `p` when the standard action selection function is used. Using the focussed profile, the right choice of goals to plan for is guaranteed, independently from such low-level features of a concrete implementation.

Evaluation. The experiment is intended to demonstrate a situation in which a focussed computee is more successful than other computees. This would suggest that in certain domains, where computees have to choose amongst a number of mutually incompatible goals, the focussed profile of behaviour can aid computees to achieve at least a minimum number of goals in good time. In our experiments, the two computees do indeed exhibit the expected difference in behaviour. However, these experiments have also revealed a problem (with both the formal model and the implementation), which causes the focussed computee to execute actions `a1` and `a3` in the wrong order. This issue is currently under investigation.

Related work. We refer to deliverable D13 for related work on behaviour profiles.

7.1.2 The Cautious Profile of Behaviour (UCY-ICSTM)

Aim. The aim of this experiment is to provide experimental result to verify the desired behaviour of the cautious profile as discussed in D13 [2]. A cautious computee will check the precondition of an action is satisfied before it plans for the action to be executed.

Parameters considered and varied. We compare the behaviour of two computees: computee 1 is cautious and computee 2 uses normal cycle theory. The main difference is that in the cautious profile, the enabling condition of the AE transition will only allow selection of an action iff its precondition can be proved to be satisfied.

Knowledge bases. We consider the following scenario: because of bad transactions in the past, from now on all bidders are required to physically possess the goods for sale before they can place their initial bid.

For KB_{react}, this experiment will use the same KB as other Combinatorial Auction bidders.

For KB_{plan}, it includes an overall precondition for the action tell:

```
precondition(tell(evelina,_,bid(_,_),_), have_product).
```

To enforce the condition that before a bid can be placed, the bidder must be able to check that it already have the product.

For KB_{TR} it has a single causal effect to allow the computee to sense the arrival of the goods hence the fulfilment of the precondition:

```
initiates(tell(warehouse,evelina,order_arrived,_,_),_,have_product).
```

Runs executed. The bidders are assumed not to have the goods for the auction at the beginning. When the two computees receive an open auction message from the auctioneer, the normal computee will start placing bids straight away, which is in violation of the protocol. The cautious computee instead will check the precondition of the bid action, i.e., it really possesses the product, before placing a bid.

Evaluation. The experiment shows that the cautious computee does indeed check for the precondition before planning for an action. Notes that if the precondition of the bid action cannot be verified in time, it is possible that the cautious bidder may miss the bidding dead line. However, this profile is beneficial in certain domain where it is paramount to follow a pre-defined protocol. For example, if there is a penalty for violation of the protocol then the cautious computee can ensure it will not be penalized.

Related work. Formal proof of the profile of behaviour is presented in D13 [2].

7.1.3 The Punctual Profile of Behaviour (ICSTM-DIPISA-UCY)

Aim. The aim of this experiment is to provide empirical evidence of the features of the punctual profile of behaviour discussed in D13 [2]. As illustrated in D13, the punctual profile is able to sequence action execution according to the ordering induced by the temporal constraints of actions. Moreover, the punctual computee is committed to execute actions and plan for goals when they become urgent, in the sense that they are "dangerously" close to their deadlines. The measure of danger is given by a parameter that indicates how much close they must be in order to be considered urgent. The tuning of the parameter may depend on specific application domains.

Parameters considered and varied. We compare the normal profile, which uses the normal cycle theory, with an implementation of the punctual profile, which uses a cycle theory where *i)* the basic rules enabling AE and PI transitions restrict the set of selected input actions and goals to those that are closer than the others to deadlines, and *ii)* the behaviour rules give higher priority to AE or PI whenever there are actions or goals that are urgent. Here urgent means closer than 5 instants to deadlines.

Knowledge bases. We consider a scenario where both computees aim at achieving the goal of going to work before time 20. Their plan is to get the bus, with having a ticket as precondition. The plan for having a ticket is simply to buy one. The domain dependent part of KB_{plan} is as follows:

```
initiates(get_bus, _, go_to_work).
precondition(get_bus, have_ticket).
initiates(buy_ticket,_,have_ticket).

executable(_).
```

The domain-dependent part of KB_{TR} is defined accordingly. KB_{react} is empty.

Run executed. We have run both computees, initialized with the goal `go_to_work` to be achieved before 20. In this oversimplified scenario, the normal computee selects actions and goals independently of their temporal deadlines. It first plans for the top goal and generates the action `get_bus` and the sub-goal `have_ticket`. Then, it first (attempts to) gets the bus, and then, after having devised a plan for the sub-goal, to buy the ticket. Actually, it fails to achieve its goal that can not be proved to hold according to the final KB_0 (indeed, the action `get_bus` has been executed at time 2, when the precondition `have_ticket` was not enforced by the action `buy_ticket`, which has been executed at 6).

Differently, the punctual computee, in this implemented version that executes actions when they become (very) urgent, is able to fully develop its plan. Then, when the execution of actions can not be further delayed, it executes the planned actions, just in time before they, and the goals they have been planned for, become timed out. The computee succeeds in achieving its goal, that is now entailed by the final KB_0, where `buy_ticket` has been executed at 14 and `get_bus` at 15, just 5 instants before their deadlines. It is also interesting to note how in steps 14 and 15 two consecutive AE, which are not usually admitted in a normal profile, have been forced due to the presence of very urgent items.

Even if the normal computee could have also preferred to fully devise its plan before executing actions, since the normal cycle theory presents non-deterministic choices, the punctual computee does guarantee both the proper ordering and the timeliness of action execution.

Evaluation. The experiment is intended to demonstrate a situation in which a punctual computee is more successful than other computees. This would suggest that in certain domains, where the ordering of actions and their timeliness is relevant, the computees adopting a punctual profile of behaviour can exhibit a more effective behaviour as far as temporal constraints (and hence goal achievement) are concerned. In our experiments, the two computees do indeed exhibit the expected difference in behaviour, since the punctual one does execute actions in the correct ordering and before they are timed out. Although this might have happened for the normal profile, a correct tuning of the characteristic parameters of the punctual profile guarantee this behaviour. The evaluation of how to properly tune such parameters is scope for future investigation.

Related work. We refer to deliverable D13 for related work on behaviour profiles.

7.2 On the fly conformance checking (UNIBO-DIFERRARA)

A number of experiments have been performed in order to verify and demonstrate that SOCS-SI can be used for on-the-fly (as defined by Guerin and Pitt [35]) conformance checking, so achieving one of the main practical applications of the SOCS social framework defined in deliverables D5 [50] and D8 [34].

Another aim of these experiments is to evaluate the applicability of the SOCS-SI approach to on-the-fly conformance checking to real-life cases, as concerns performance and scalability.

In the following, as an example, we report on experiments in the Needham-Schroeder (Sect. 7.2.1) scenario, a well-known security protocol. Quantitative performance tests were obtained for the auction scenario (see [49]).

7.2.1 On the fly conformance checking for the Needham-Schroeder protocol

The Needham-Schroeder protocol (NS, for short) has been presented in [52], where the authors discuss a way for ensuring the mutual exchange of a secret (a pair of numbers, called *nonces*) between two peers over an *insecure network connection.* The purpose of the protocol is to ensure mutual authentication while maintaining secrecy. In other words, once agents A and B have successfully completed a run of the protocol, A should believe his partner to be B if and only if B believes his partner to be A.

In support of the authentication procedure, agents rely on the well-known public key encryption technology. By following the protocol, the two agents involved in a communication session

(1) $A \rightarrow B : \langle N_A, A \rangle_{pub\_key(B)}$

(2) $B \rightarrow A : \langle N_A, N_B \rangle_{pub\_key(A)}$

(3) $A \rightarrow B : \langle N_B \rangle_{pub\_key(B)}$

Figure 5: The Needham-Schroeder protocol (simplified version)

(conversation) challenge each other to make sure that each one's partner in the conversation is actually the holder of the private key associated with his public key.

The protocol consists of seven steps, but – like other authors [24] – we focus on a simplified version of it, consisting of only three steps, which are the kernel of the protocol. The simplification means that we assume that all the agents know the public key of the other agents. Hence the interaction can be represented as in Figure 5, where $\langle M \rangle_{PK}$ means that M is encrypted with public key PK.

By message (1), A challenges B to decrypt his nonce N_A encrypted using B's public key. By message (2), B responds to A's challenge, by attaching to N_A a new nonce N_B, which he generated himself, and encrypting the whole set of two nonces using A's public key, thus challenging A to decrypt N_B and prove to be the holder of A's private key. At this point of interaction, A believes that he is speaking with B, since the latter proved to be able to decrypt the message (1) and answering back the N_A. Of course, this is reasonable under the assumption that it is extremely improbable that an agent could guess the nonce N_A, hence considering it impossible. By message (3), A responds to B's challenge, giving a proof (the N_B sent in message (2)) of being A. In similar way to what happens after messages (1) and (2), B believes his fellow is A upon receiving message (3).

Aside the insecure connection conditions, we make assumptions about the agents' abilities. In doing that, we refer to the Dolev-Yao model [21], which relies on the perfect cryptography assumption (nothing can be learned on a plain text from its encrypted version, without knowing the decryption key). In particular, if we want to define the perfect cryptography assumption in terms of exchanged messages, we say that an agent can:

- decrypt messages encrypted with his own public key;
- generate messages with nonces that (*i*) have never been generated by other agents, or (*ii*) that he received in a message encrypted with his own public key;
- forward messages.

Eighteen years after the publication of the NS protocol, Lowe [45] proved it to be prone to security attack, by showing a breach. Lowe's attack on the protocol is presented in Figure 6, where a third agent I (standing for *intruder*) manages to successfully authenticate himself as agent A with a third agent B. Although the protocol is correctly followed, B believes he is communicating with A, while instead he is communicating with I. Therefore, the messages composing the attack belong to two different dialogues, A with I and I with B. Each dialogue follows the protocol, but I exploits the information of the first dialogue to deceive B in the second dialogue.

(1) $A \rightarrow I : \langle N_A, A \rangle_{pub\_key(I)}$

(2) $I(A) \rightarrow B : \langle N_A, A \rangle_{pub\_key(B)}$

(3) $B \rightarrow I(A) : \langle N_A, N_B \rangle_{pub\_key(A)}$

(4) $I \rightarrow A : \langle N_A, N_B \rangle_{pub\_key(A)}$

(5) $A \rightarrow I : \langle N_B \rangle_{pub\_key(I)}$

(6) $I(A) \rightarrow B : \langle N_B \rangle_{pub\_key(B)}$

Figure 6: Lowe's attack on the Needham-Schroeder protocol

```
    H( send( X, B, content( key( KB), agent( A), nonce( NA))), T1)
--->
    E( send( B, X, content( key( KA), nonce( NA), nonce( NB))), T2)
    /\ isPublicKey( A, KA) /\ isNonce( NB) /\ NA!=NB
    /\ isMaxTime( TMax) /\ T2 > T1 /\ T2 < TMax.

    H( send( X, B, content( key( KB), agent( A), nonce( NA))), T1)
    /\ H( send( B, X, content( key( KA), nonce( NA), nonce( NB))), T2)
    /\ T2 > T1
--->
    E( send( X, B, content( key( KB), nonce( NB), empty( 0))), T3)
    /\ isMaxTime( TMax) /\ T3 > T2 /\ T3 < TMax.
```

Figure 7: Social Integrity Constraints defining the Needham-Schroeder protocol

Aim. The aim of this experiment is to verify that the SOCS social framework is expressive enough to represent the well-known NS protocol, and that the implementation of the $\mathcal{S}$CIFF proof procedure can correctly check on the fly conformance to the protocol, in case of Lowe's attack.

Parameters considered and varied. The executed runs only differ for the considered histories (see below).

Knowledge bases. Events that represent the messages are the following:

- $\mathbf{H}(send(A, B, content(key(K), agent(A), nonce(N_A))), T)$

  (agent A has sent to agent B its own identifier and a nonce, encrypted with a key K, at time T)

- $\mathbf{H}(send(A, B, content(key(K), nonce(N_A), nonce(N_B))), T)$

  (agent A has sent to agent B two nonces, encrypted with a key K, at time T)

- $\mathbf{H}(send(A, B, content(key(K), nonce(N_A), empty(0))), T)$

  (agent A has sent to agent B a nonce, encrypted with a key K, at time T)

The IC_S used for defining the protocol are of two types:

```
isPublicKey( PK) :-
   isPublicKey( _, PK).

isPublicKey( i, ki).
isPublicKey( b, kb).
isPublicKey( a, ka).

isMaxTime( 7).

isAgent( i).
isAgent( a).
isAgent( b).
```

```
isNonce( N) :-
    isNonce( _, N).

isNonce( A, N) :-
   checkIfNonce( A, N).

notIsNonce( A, NB) :-
   \+( checkIfNonce( A, NB)).

checkIfNonce( b, nb).
checkIfNonce( a, na).
```

Figure 8: Social Organization Knowledge Base – SOKB

```
h(send( a, b, content( key( kb), agent( a), nonce( na))), 1).
h(send( b, a, content( key( ka), nonce( na), nonce( nb))), 2).
h(send( a, b, content( key( kb), nonce( nb), empty( 0))), 3).
```

Figure 9: A compliant history.

- A first group of IC_S, depicted in Figure 7, defines the (simplified version of the) protocol itself.
- A second group of IC_S is needed in order to impose the condition that an agent is not able to guess another agent's *nonce*, neither a private key that he does not own. These constraints can be found in [11], available at the Web page: `http://www-lia.deis.unibo.it/Research/Projects/SOCS/guests/publications/D14.html`.

The *SOKB* of this experiment is in Figure 8.

Runs executed. On the fly conformance has been checked for four hand-written histories, which represent typical interactions in the NS protocol.

- Figure 9 shows an example of compliant history.
- Figure 10 is an example of an history that is not compliant because the protocol has not been completed. In fact, the two events in the history propagate the second integrity constraints of Figure 7 and impose an expectation

  ```
  e(send( a, b, content( key( kb), nonce( nb), empty( 0))), T3)
  ```

  (with the CLP constraint `T3>2`), which is not fulfilled by any event in the history.
- The history in Figure 11, instead, while containing a complete protocol run, is found by $\mathcal{S}$CIFF to violate the integrity constraints because agent `a` has used a nonce (`nc`) that it cannot know, because is not one of `a`'s nonces and `a` has not received it in any previous message. In terms of integrity constraints, the history satisfies those in Figure 7, but it violates one Ic of the second group.
- Figure 12 depicts Lowe's attack, which $\mathcal{S}$CIFF finds compliant to the protocol as defined by the integrity constraints.

```
h(send( a, b, content( key( kb), agent( a), nonce( na))), 1).
h(send( b, a, content( key( ka), nonce( na), nonce( nb))), 2).
```

Figure 10: A non-compliant history (the third message is missing).

```
h(send( a, b, content( key( kb), agent( a), nonce( nc))), 1).
h(send( b, a, content( key( ka), nonce( nc), nonce( nb))), 2).
h(send( a, b, content( key( kb), nonce( nb), empty( 0))), 3).
```

Figure 11: A non-compliant history (agent a has used a nonce that it cannot hold).

Evaluation. This experiment shows that the SOCS social framework can express the Needham-Schroeder protocol and that the implementation of the $\mathcal{S}$CIFF proof procedure can verify correctly on the fly compliance for typical interactions. These results are important, given the relevance of the Needham-Schroeder protocol in the literature on the automatic proof of properties of security protocols. In particular, we have tested that Lowe's attack, when mapped into a proper history of events, is processed by $\mathcal{S}$CIFF as compliant to the protocol, therefore testing the flawyness of the protocol itself. Furthermore, in Section 7.3.2, we will exploit a variant of $\mathcal{S}$CIFF in order to generate a compliant history representing Lowe's attack for this protocol.

7.3 Automatic proof of properties (UNIBO-DIFERRARA)

In this section, we report on experiments performed as support to the proposed approach for automatically proving protocol properties, as documented in deliverable D13 and in the papers [12, 11] (they both are available at the mentioned site containing papers relevant to D14).

Agent interaction verification is one aspect which has been intensively studied under many perspectives in the area of open computational systems. As stated in D13, the capability of the SOCS social model to automatically prove or refute significant protocol properties is a demonstration of the expressiveness of the formal model and of the effectiveness of its operational counterpart; more generally, it shows the applicability of Computational Logic to problems that, so far, have been tackled mainly by Model Checking techniques [51].

Several other frameworks in the literature aim at verifying properties about the behavior of social agents at design time. Often, such frameworks define structured hierarchies, roles, and deontic concepts such as norms and obligations as first class entities (see deliverable D5 [50], too). Notably, among them, ISLANDER [28] is a tool for the specification and verification of interaction in complex social infrastructures, such as electronic institutions. ISLANDER allows to analyze situations, called scenes, and visualize liveness or safeness properties in some specific settings. The kind of verification involved is static and is used to help designing institutions.

Among related approaches, model checking [18, 51] is a well-known and widely used approach for automatic analysis of reactive systems. The inputs to a model checker are usually a finite state description of the system to be analyzed, and a number of properties, often expressed as formulas of temporal logic that are expected to hold of the system. The model checker either confirms that the properties hold or reports that they are violated. In the latter case, it provides a counterexample: a run that violates the property. Such a run can provide valuable feedback and points to design errors. Model checking has been extensively applied to prove properties of

```
h(send( a, i, content( key( ki), agent( a), nonce( na))), 1).
h(send( i, b, content( key( kb), agent( a), nonce( na))), 2).
h(send( b, i, content( key( ka), nonce( na), nonce( nb))), 3).
h(send( i, a, content( key( ka), nonce( na), nonce( nb))), 4).
h(send( a, i, content( key( ki), nonce( nb), empty( 0))), 5).
h(send( i, b, content( key( kb), nonce( nb), empty( 0))), 6).
```

Figure 12: Lowe's attack, recognized as a compliant history.

reactive and distributed systems, and it has been also applied to prove properties of security, communication and electronic commerce protocols (see, for instance, [29, 51]).

An alternative approach for proving properties is the proof theoretic one. Suppose we have a logic-based specification ϕ_P of a system P, (for example, P can be a communication protocol), and we want to know whether it satisfies some property ϕ, which we assume can be expressed in logic too. One way to do this is to check whether $\phi_P \Rightarrow \phi$ is valid (i.e., ϕ is true in all the models of ϕ_P). Model checking represents P by a Kripke structure M_P: the states in M_P represent the possible global states of P. Checking whether P satisfies ϕ amounts at checking whether ϕ is true in the initial state in M_P. In the proof theoretic approach, instead, we can rely on the syntactic structure of ϕ_P and ϕ. Usually, the proof is given by contradiction. The drawback of this approach is due to undecidability of first order logic, and its (exponential) complexity.

Our approach aims at proving or refuting properties automatically, by following the proof theoretic approach, and grounding on Computational Logic. For this purpose, we have extended the $\mathcal{S}$CIFF proof procedure, used for on-the-fly conformance checking, with a capability to generate histories that are compliant to a given protocol and to a $\mathcal{S}$CIFF goal (see paper [12], available at the Web page: `http://www-lia.deis.unibo.it/Research/Projects/SOCS/guests/publications/D14.html`).

The extended proof procedure, named g-$\mathcal{S}$CIFF, where the *g* stands for *generative*, changes the $\mathcal{S}$CIFF *Fulfillment E* transition into a new one, and always turns expected behaviour (and positive expectations, in particular) into events, possibly containing variables (see [12], for details). Therefore, the result of a successful g-$\mathcal{S}$CIFF computation for a goal G is an output history representing one history of events (possibly with variables) supporting the achievement of G. A partial history can be also specified as input to the g-$\mathcal{S}$CIFF computation.

This approach has been applied to both existential and universal properties, defined as in deliverable D13. Informally:

- *existential* properties are formulae that hold for at least one history compliant to the protocol;
- *universal* properties are formulae that hold for all the histories compliant to a protocol.

Up to now, soundness of $\mathcal{S}$CIFF has been proved (see Deliverable D13). Thanks to its soundness (proved in [12]), operatively, g-$\mathcal{S}$CIFF can be exploited to verify existential properties and disprove universal ones, as follows. If an existential property f can be expressed as a $\mathcal{S}$CIFF goal, and by running g-$\mathcal{S}$CIFF we get a success with an output history, then we conclude that f is an existential property of the considered protocol, and the output history (or better, one of its grounding) is a witness, compliant history that satisfies f.

A universal property f can be disproved, instead, by expressing (the existential property) $\neg f$ as a $\mathcal{S}$CIFF goal, and running g-$\mathcal{S}$CIFF. If g-$\mathcal{S}$CIFF returns success with an output history, then we conclude that $\neg f$ is proved and therefore f is not a universal property of the considered protocol. The output history (or better, one of its grounding) is a counterexample for the universal property f, i.e., it represents a compliant history for which f does not hold (since $\neg f$ holds in it, instead).

This approach has been experimented in order to confirm and disconfirm properties of two well-known interaction protocols, one in the context of electronic commerce (the NetBill protocol, Sect. 7.3.1) and one in the context of security (the Needham-Schroeder protocol, Sect. 7.3.2). In these Sections, we also mention a related proof theoretic approach based on abduction for proving properties. Literature on properties of the two considered protocols is surveyed in the mentioned papers [12, 11].

7.3.1 Good atomicity in the NetBill protocol (DIFERRARA)

NetBill (see [20]) is a security and transaction protocol optimized for the selling and delivery of low-priced information goods, like software or journal articles. The protocol rules interactions between two agents: *merchant* and *customer*. A NetBill server is used to deal with financial issues such as those related to credit card accounts of customer and merchant.

In the following, we focus on the type of the NetBill protocol designed for non zero-priced goods, and do not consider the variants that deal with zero-priced goods.

A typical protocol run is composed of three phases:

1. *price negotiation*. The customer requests a quote for a good identified by an identifier *PrId* (`priceRequest(PrId)`), and the merchant replies with (`priceQuote(PrId,Quote)`).

2. *good delivery*. The customer requests the good (`goodRequest(PrId,Quote)`) and the merchant delivers it in an encrypted format (`deliver(crypt(PrId,Key),Quote)`).

3. *payment*. The customer issues an Electronic Payment Order (EPO) to the merchant, for the amount agreed for the good (`payment(epo(C,crypt(PrId,K),Quote))`); the merchant appends the decryption key for the good to the EPO, signs the pair and forwards it to the NetBill server (`endorsedEPO(epo(C,crypt(PrId,K),Quote),M)`); the NetBill server deals with the actual money transfer and returns the result to the merchant (`signedResult(C,PrID,Price,K)`), who will, in turn, send a receipt for the good and the decryption key to the customer (`receipt(PrId,Price,K)`).

The customer can withdraw from the transaction until he has issued the *EPO* message; the merchant until he has issued the *endorsedEPO* message.

Aim. This experiment considers a universal [2] *good atomicity* property of the protocol: i.e., *the merchant receives the payment for a good G iff the customer receives the good G*.

Since $\mathcal{S}$CIFF deals with (communicative) events and not with the states of the agents, we need to express the properties in terms of happened events. To this purpose, we can assume that merchant has received the payment once the NetBill server has issued the *signedResult* message, and that the customer has received the good if he has received the encrypted good (with a *deliver* message) and the encryption key (with a *receipt* message).

Thus, the universal property can be expressed as

$$\begin{aligned} & H(\mathit{tell}(\mathit{netbill}, M, \mathit{signedResult}(C, \mathit{PrId}, \mathit{Quote}, K), \mathit{Id}), \mathit{Tsign}) \\ \iff & H(\mathit{tell}(M, C, \mathit{goodDelivery}(\mathit{crypt}(\mathit{PrId}, K), \mathit{Quote}), \mathit{Id}), T) \\ & \wedge H(\mathit{tell}(M, C, \mathit{receipt}(\mathit{PrId}, \mathit{Quote}, K), \mathit{Id}), \mathit{Ts}) \end{aligned} \tag{1}$$

whose negation is expressed as the following $\mathcal{S}$CIFF goal:

$$\begin{aligned} g \leftarrow & EN(\mathit{tell}(\mathit{netbill}, M, \mathit{signedResult}(C, \mathit{PrId}, \mathit{Quote}, K), \mathit{Id}), \mathit{Tsign}), \\ & E(\mathit{tell}(M, C, \mathit{goodDelivery}(\mathit{crypt}(\mathit{PrId}, K), \mathit{Quote}), \mathit{Id}), T), \\ & E(\mathit{tell}(M, C, \mathit{receipt}(\mathit{PrId}, \mathit{Quote}, K), \mathit{Id}), \mathit{Ts})). \\ g \leftarrow & E(\mathit{tell}(\mathit{netbill}, M, \mathit{signedResult}(C, \mathit{PrId}, \mathit{Quote}, K), \mathit{Id}), \mathit{Tsign}), \\ & EN(\mathit{tell}(M, C, \mathit{goodDelivery}(\mathit{crypt}(\mathit{PrId}, K), \mathit{Quote}), \mathit{Id}), T). \\ g \leftarrow & E(\mathit{tell}(\mathit{netbill}, M, \mathit{signedResult}(C, \mathit{PrId}, \mathit{Quote}, K), \mathit{Id}), \mathit{Tsign}), \\ & EN(\mathit{tell}(M, C, \mathit{goodDelivery}(\mathit{crypt}(\mathit{PrId}, K), \mathit{Quote}), \mathit{Id}), T)) \end{aligned} \tag{2}$$

By running g-$\mathcal{S}$CIFF, any compliant history achieving such a goal is a counterexample of the original, universal property, thanks to g-$\mathcal{S}$CIFF soundness.

Knowledge bases. KBs and IC_S used in the experiment are contained in [12]. IC_S are divided into two sets, the former containing only backward constraints (i.e., relating happened events to expectations on past events), the latter only one forward constraint (i.e., relating happened events to expectations on future events).

Runs executed. Two runs were executed:

1. For the first run, we have used a complete representation of the protocol, (all the IC_S backward and forward);
2. for the second run, we have removed the forward constraint, in order to verify that such constraint is not redundant with respect to the property that we want to check.

The results of the two runs are the following:

1. g-$\mathcal{S}$CIFF reports failure, i.e., it is not able to generate a history that satisfies the goal;
2. g-$\mathcal{S}$CIFF reports success, and the following history is generated:

```
h(tell(_E,_F,priceRequest(_D),_C),_M),
h(tell(_F,_E,priceQuote(_D,_B),_C),_L),
h(tell(_E,_F,goodRequest(_D,_B),_C),_K),
h(tell(_F,_E,goodDelivery(crypt(_D,_A),_B),_C),_J),
h(tell(_E,_F,payment(_E,crypt(_D,_A),_B),_C),_I),
h(tell(_F,netbill,endorsedEPO(epo(_E,_D,_B),_A,_F),_C),_H),
h(tell(netbill,_F,signedResult(_E,_D,_B,_A),_C),_G),
_I<_H, _H<_G,
_L>_M, _K>_L, _I>_J, _J>_K,
```

The *receipt* event is missing, which would violate the integrity constraint that has been removed.

Evaluation. The result of the first run is a failure. This suggests that there is no history that entails the negation of the good atomicity property while respecting the protocol, i.e., the property is likely to hold if the protocol is respected. However, yet no guarantee can be given unless g-$\mathcal{S}$CIFF is proven complete.

The result of the second run is a success. Thanks to g-$\mathcal{S}$CIFF soundness, this run therefore shows that the good atomicity property does not hold for the second version of the protocol. In this way, a protocol designer can make sure that an integrity constraint is not redundant with respect to a desired, universal property of the protocol.

Related work. In [57] the authors discuss the application of abductive reasoning for the analysis of safety properties of declarative specifications expressed in the Event Calculus. In their abductive approach, the problem of proving that, for some invariant I, a domain description D entails I ($D \models I$), is translated into an equivalent problem of showing that it is not possible to consistently extend D with assertions that particular events have actually occurred (i.e., with a set of abductive hypotheses Δ), in such a way that the extended description entails $\neg I$. In other words, there is no set Δ such that $D \cup \Delta \models \neg I$. They solve this latter problem by a complete abductive decision procedure, thus exploiting abduction in a refutation mode. Whenever the procedure finds a Δ then the assertions in Δ act as a counterexample for the invariant.

Our approach is similar to the one described in [57], when g-$\mathcal{S}$CIFF is able to refute a given (universal) property by finding a counterexample for its negation. This has been exploited in this experiment to disprove good atomicity for the NetBill protocol. Nonetheless, completeness is a requirement for proving, instead, a universal property.

Further related work about logic-based approaches for security protocols is reported in [12].

7.3.2 Generating Lowe's attack for the Needham-Schroeder protocol (UNIBO-DIFERRARA)

Aim. In the idea of the Needham-Schroeder protocol, an agent trusts the identity of the agent with whom he is communicating by associating his name with his public key and receiving back a nonce that he forged, encrypted in his own public key. If we had to define the idea of an agent B 'trusting' that he is communicating with A, we could do it by using a combination of messages in which an agents responds to a challenge posed by another agent and successfully decrypts a nonce.

Definition 7.1. *We say that B trusts that the agent X he is communicating with is A,*[4] *and we write $trust_B(X, A)$ once two messages have been exchanged at times T_1 and T_2, $T_1 < T_2$, having the following sender, recipient, and content:*

$$(T_1)\ B \rightarrow X : \{N_B, \dots\}_{pub\_key(A)}$$

$$(T_2)\ X \rightarrow B : \{N_B, \dots\}_{pub\_key(B)}$$

where N_B is a nonce generated by B.

[4]We restrict ourselves to only one communication session, all the definitions will therefore have as a scope the session.

Note that B is unable to judge whether N_A is a nonce actually generated by X or not, therefore no condition is posed on the origin of such nonce.

Symmetrically, we can consider, from A's viewpoint, messages (1) and (2) as those that prove the identity of B. We therefore implement Def. 7.1 in Def. 7.2, where messages are expressed using the notation of the $\mathcal{S}$CIFF, namely as events which are part of some "history" **HAP**. The content of messages will be composed of three parts, the first showing the public key used to encrypt it, the second and third containing agent names or nonces or nothing (in particular, the last part may be empty).

Definition 7.2. *Let A, B and X be agents, K_A and K_B respectively A's and B's public key, N_B a nonce produced by B, and let $\mathbf{HAP}_1$ and $\mathbf{HAP}_2$ be two sets of events each composed of two elements, namely:*
$\mathbf{HAP}_1 = \{$
$\mathbf{H}(send(B, X, content(key(K_A), agent(B), nonce(N_B))), T_1),$
$\mathbf{H}(send(X, B, content(key(K_B), nonce(N_B), nonce(...))), T_2)$
$\}$*, and*
$\mathbf{HAP}_2 = \{$
$\mathbf{H}(send(B, X, content(key(K_A), nonce(...), nonce(N_B))), T_1),$
$\mathbf{H}(send(X, B, content(key(K_B), nonce(N_B), empty(0))), T_2)$
$\}$.

Then, $trust_B(X, A)$ holds iff $\mathbf{HAP}_1 \subseteq \mathbf{HAP}$ or $\mathbf{HAP}_2 \subseteq \mathbf{HAP}$.

Knowledge bases. The protocol is represented by the same *SOKB* and IC_S adopted in Section 7.2.1, and reported extensively in [11].

Runs executed. The universal property that we want to disprove is $\mathcal{P}_{trust}$ defined as

$$trust_B(X, A) \rightarrow X = A$$

i.e., if B trusts that he is communicating with A, then he is indeed communicating with A. We obtain a problem which is symmetric in the variables A, B, and X. In order to check if we have a solution we can ground $\mathcal{P}_{trust}$ and define its negation $\neg\mathcal{P}_{trust}$ as a goal, $g3$, where we choose to assign to A, B, and X the values a, b and i:

$g3 \leftarrow isNonce(NA), NA \neq nb,$
  $\mathbf{E}(send(b, i, content(key(ka), nonce(NA), nonce(nb))), 3),$
  $\mathbf{E}(send(i, b, content(key(kb), nonce(nb), empty(0))), 6).$

Besides defining $g3$ for three specific agents, we also assign definite time points (3 and 6) in order to improve the efficiency of the proof.

Running the g-$\mathcal{S}$CIFF on $g3$ results in a compliant history:
$\mathbf{HAP}_{g3} = \{$
$h(send(a, i, content(key(ki), agent(a), nonce(na))), 1),$
$h(send(i, b, content(key(kb), agent(a), nonce(na))), 2),$
$h(send(b, i, content(key(ka), nonce(na), nonce(nb))), 3),$
$h(send(i, a, content(key(ka), nonce(na), nonce(nb))), 4),$
$h(send(a, i, content(key(ki), nonce(nb), empty(0))), 5),$
$h(send(i, b, content(key(kb), nonce(nb), empty(0))), 6)$
$\}$,

which is indeed Lowe's attack on the protocol. $\mathbf{HAP}_{g3}$ represents a counterexample of the property $\mathcal{P}_{trust}$.

Evaluation. Thanks to the soundness of g-$\mathcal{S}$CIFF, the generated history can be considered a counterexample of the property that was being verified.

Related work. Various techniques have been adopted to the task of automatic verification of properties to the case study of the Needham-Schroeder, or its evolution proposed by Lowe. They are reported extensively in [11].

Part III
Achieved results

In the following, we summarize achieved results of work-package WP6, and present an evaluation of it in terms of the principle set out in deliverable D3 [44].

In particular, a catalogue of behaviour profiles was implemented, and added to PROSOCS platform, as already discussed in Section 2.3.

Furthermore, besides the experiments reported in this document, it is worth to be mentioned that one of the underlying PROSOCS proof procedure, Gorgias, has been adopted for the development of real-life applications [**?**].

Here, we also describe a catalogue of interaction protocols, developed and experimented in the context of work-package WP6, and available on line at the Web site `http://www.lia.deis.unibo.it/Research/Projects/SOCS/partners/societies/protocols.html`.

After evaluation and self-assessment, we also mention further results, in terms of publications and demos given, mainly pertaining WP6, among others work-packages.

8 A Catalogue of Protocols (UNIBO-DIFERRARA

In this Section, we provide a brief description of the catalogue of interaction protocols experimented within WP6, and available on line at the Web site `http://www.lia.deis.unibo.it/Research/Projects/SOCS/partners/societies/protocols.html`.

In particular, protocols are grouped into families, according to their application domain. The Web site was conceived as a dynamically one: registered users may insert the specification of a society in terms of triples, namely $SOKB$, *Goal* and IC_S, and also provide different histories used for specifications' testing. Finally, each society instance (i.e., a specification of a society equipped with a given history) can be associated with a run (and with both qualitative and quantitative tests executed) by choosing the underlying proof procedure. Today, the following protocols are available:

- **NetBill Protocol** NetBill [20] is a security and transaction protocol optimized for the selling and delivery of low-priced information goods, like software or journal articles. The protocol rules transactions between two agents: a merchant and a customer. A NetBill server is used to deal with financial issues such as those related to credit card accounts of customer and merchant. After a price negotiation phase, the merchant delivers the good in an encrypted format; the customer replies with an Electronic Payment Order (EPO). Then, the merchant appends the good's decryption key to the EPO and forwards it to the NetBill server. The latter executes the money transaction and return an answer to the merchant, who should forward it to the customer.

- **FIPA Query Interaction Protocol-simplified version** In the FIPA Query Interaction Protocol a receiving agent is requested to perform some kind of inform action. In our case, a computee A requests some information to a computee B using a $query-ref(info)$ performative. Then, B is expected to answer $inform(info, response)$, giving the requested information referred in info, or $refuse(info)$, but not both of them.

- **Semi-Open Society** A semi-open society is a society that can be joined by a computee executing an entering protocol. We suppose that an agent, called *gatekeeper*, manages joining requests. The *gatekeeper* requires, for the computee to enter, some form to be filled. The latter fills it, and sends it back; the gatekeeper should then decide (and communicate the decision) to accept or reject the joining request. The *accept* message can be used to testing the agent's membership to society.

- **FIPA Request Interaction Protocol** The FIPA Request Interaction Protocol allows one agent to request another to perform some actions. The latter may decide to refuse or accept the request (with an *agree* performative). In case of agreement the agent must communicate either a $failure$ (if it cannot fill the request) or a $inform-done/inform-result$ (to indicate that he completed successfully the request and, eventually, to notify the result).

- **Needham-Schroeder** The Needham-Schroeder authentication protocol [52] aims at establishing a connection between two agents, A and B, in which the agent A who initiates the protocol is authenticated with B. In order to reach this goal, A and B rely on the well-known public key encryption technology (an agent is associated to two keys: a public one, available to other agents which use it to encrypt messages for its owner, and a private one, known only by the owner and used for the "exclusive" decryption).

- **Auctions** [6] In an auction we have two families of participants: the *auctioneer* and *bidders*. Depending on the kind of auction, the auctioneer either sells goods/services or buys a set of goods/services. Bidders have the goal to obtain/sell their goods/services under convenient conditions as far as price is concerned. The auctioneer has the goal to sell/obtain a set of goods/services maximizing the profit (or minimizing the cost) at the minimum risk. The auctioneer may delegate some operations to an external object (i.e. a software component with an API). An example of such an object is an auction solver object that allows an auctioneer to use the optimization techniques and constraint capabilities of the ILOG solver for the winner's determination.

  Examples of auctions' protocols:

  - **First Price Sealed Bid Auction** The first price sealed bid auction is an auction in which bidders simultaneously submit bids to the auctioneer without knowledge of the amount bid by other participants. The highest bidder is declared the winner. The protocol controls therefore that agents place bids only for valid auctions (i.e., opened auctions) and that all bidders receive an answer (win or loose).
  - **Single Combinatorial Auction** In combinatorial auctions, bidders can bid on combination of items. In this context, the auctioneer has the NP-hard problem to find the set of winning bids at a minimum cost or maximum revenue (Winner Determination Problem). The single combinatorial auction is a traditional auction where bidders post their bids on combinations of items and the auctioneer decides which bid wins. The protocol says that a bidder cannot start placing bids if the auctioneer has not opened an auction. When the auction's deadline fires, the auctioneer should answer to each bidder, saying if it wins or looses. Note that the protocol doesn't deal with the Winner Determination Problem, which is delegated to the auction solver.
  - **Double Combinatorial Auction** In the double combinatorial auction, it is crucial that the bids cover many items. Proposing appealing bids makes the probability of winning higher. Therefore, in order to produce more appealing bids, one or more bidders can decide to propose

a bid with the items he/she owns plus some other he buys on the market. For buying these items, he/she opens a second auction looking for missing items at the best price. This second auction has clearly some constraints linking it to the first one. The most straightforward is that the second auction should be closed before the first one is. In this way, the bidder that opened the second auction buys all the missing items and proposes a bid (to the first auction) containing his/her items plus the ones he/she boughs.

  – **Combinatorial Auction NetBill Delivery** This protocol formalizes the combinatorial auction adding the payment at the end; when a bidder wins, it must deliver the list of involving items. Then, the delivered list must be payed: here we use the NetBill protocol.

  – **English Auction** The English auction is a type of sequential auction in which an auctioneer directs participants to beat the current, standing bid. New bids must increase the current bid by a predefined increment. The protocol considers a deadline, in terms of a time interval starting from the last bid: agents can place bids within this deadline. After the *close-auction* event, the auctioneer must send an answer to all the bidders. All the bidders, except the one that bids latest, must loose; the last bidder wins only if its bid overcomes a predefined reservation price (otherwise, it looses too).

    Finally, the protocol forbids that an agent places a bid (or closes the auction) for an auction that has not been opened.

- **Borda Count** [30] The Borda Count interaction protocol aims to find a consensus choice, which gives a better representation of what people really want, in a group's election. All voters have to rank their preferences among a collection of N alternatives (except if the Borda Count calculator decides otherwise): they allocate N points to the most preferred strategy, N-1 to the next best, and so on. A central entity ranks the scores given to each strategy and selects the most voted strategy as the winner.

- **TCP/IP Opening Phase**[54] The Transmission Control Protocol over IP is a famous and well-established protocol used in the majority of the modern Internet services. It allows bi-directional communication between peers using the Internet Protocol for routing services; it is connection oriented and it supports limited de-synchronization between sender and receiver through a dynamic adaptive sync window. The protocol here reported represents the opening phase in the "three hand-shake" form, where three consecutive messages are exchanged in order to establish a connection. Particular attention is given to the timeout mechanism for re-transmitting the first "syn" message: the timeout interval is specified as a parameter in the SOKB, and the protocol addresses the problem of early re-transmission.

- **Example of medical guideline** This protocol formalizes a fragment of a microbiological clinical guideline applied in hospitals for microbiological infections' treatment. The guideline may be structured in seven macro phases: patient's arrival at the hospital emergency room; patient's examination at the emergency room; possible admission in a specific hospital ward and first therapy prescription made by the ward's physician; request of a microbiological test (consisting of many sub-phases, involving both human and artificial actors); return of the microbiological test report to the ward physician, who has to decide the final therapy; management of drugs by nurses; evaluation of patient's health and, in case of symptoms' persistence, new prescription of microbiological test.

9 Evaluation and self-assessment

The evaluation of work-package WP6 is done in terms of the principle set out in deliverable D3 [44], and follows the analogous evaluation already done for work-packages WP1-WP4 in deliverable D3 [17].

Recall that the central aim of work-package WP6 is to experiment with individual computees and societies of computees, through a series of controlled experiments by using the PROSOCS platform developed by WP4, and also to test some of the desirable and undesirable properties of both individual computees and societies of computees, as identified within WP5.

Work-packages WP5 and WP6 have been respectively dedicated to the formal verification and to the observation of characteristics of (societies of) computees. WP5 concentrated on properties for which our approach lends itself better to a formal verification, whereas WP6 concentrated on characteristics emerging from the animation of (societies of) computees in order to capture possibly new insights for WP5, amongst other things. To this purpose, in work-package WP6 we tested the models defined in WP1, WP2 and WP3, by exploiting their implementation in PROSOCS, the outcome of WP4. Experimentation provided a feedback to all work-packages, and a better tuning of the models and single components developed for computees and societies of computees.

Work-package WP6 has evaluated and validated the framework developed by the SOCS project through a series of controlled experiments, confirmed some of the properties of (societies of) computees proven within WP5 and described in deliverable D3 [44], also experimented with a proof theoretic approach for automatically prove or disprove protocol properties.

More specifically, as stated in Annex 1 and deliverable D3, WP6 was aimed at:

- testing the theoretical (computational-logic based) model for single computees and their inner behavior, defined in WP1, by exploiting its implementation as resulting from WP3 and WP4;
- testing the theoretical (computational-logic based) model for societies of computees, defined in WP2, by exploiting its implementation as resulting from WP3 and WP4;
- testing some of the desirable and undesirable properties of both computees and their societies, in particular some among those identified in WP5;
- testing predictions, arising from the theoretical results of WP1-WP5, by experimenting with the scenarios identified in the different phases of the project;
- possibly identifying novel characteristics of the system (as emerging properties which can be observed dynamically, rather than formally proved).

Beside these issues, along work-package WP6, we have also experimented the g-$\mathcal{S}$CIFF proof procedure for:

- automatically proving or disproving protocol properties, by experimenting with well-known protocols whose properties have been documented in the literature.

The evaluation in this Section is hence the evaluation of WP6 as it is at the end of the third year.

9.1 Evaluation stages

The evaluation and self-assessment process for WP6 basically spans over the last year project, and is tightly coupled with work-package WP5, in particular, and previous work done in the project, more generally.

Experimentation in WP6 aims at (*i*) testing the formal models developed for the single computees and societies of computees and their implementation, (*ii*) confirming or disconfirming properties formally proved in WP5, and (*iii*) possibly identifying novel (emerging) characteristics of the system.

We have therefore separated the evaluation of WP6 into a first stage that evaluates it in relation with the theoretical models of computees and societies of computees (better with work-packages WP1-WP4) (item (*i*) above), and a second stage that evaluates it in relation with work-package WP5 (items (*ii*) and (*iii*) above).

The first stage of evaluation of WP6 has concentrated on the following steps:

Parameter identification: Along work-package WP4, some notable parameters have been already identified, and shown in the prototype user interface. WP6 has given further insights to WP4. The experiments have varied some notable parameters of the framework, such as: reasoning capabilities of the individual computees, inner behavior of computees, mode of communication and interaction (society protocols), scale of density of computees in a society.

Develop specific experiments: For each evaluation criterion of the first group given for work-packages WP1, WP2 and WP5, specific experiments on (societies of) computees and scenarios have been identified in order to test and control how the identified parameters impact on the overall system, i.e., the PROSOCS platform itself or on its components.

Evaluate experiments: For each of the experiments set out, the behavior of the system has been then examined, with the aim to prove/disprove that the system exhibits the desired/undesired features of the evaluation. Experiments have been also analyzed with the aim of discovering novel, relevant characteristics and provide insights to other work-packages, W3 and WP4 in particular.

The second stage of evaluation of the experimentation in WP6 is linked to WP5, and has been structured into the following steps. WP5 aims at verifying formal properties of the models define in WP1 and WP2, and their operational counterpart as defined in WP3. The project has set out in the internal document "A list of verifiable properties", a preliminary list of formal properties of the behavior of the computees and societies of computees that are to be examined within WP5, and updated it in deliverable D12 and D13 [5, 2].

Develop specific experiments for properties: Experimental verification of formally proved properties is necessary because the theoretical framework must make assumptions which may not be satisfied in real world applications. For some identified (and formally proved) properties, specific experiments as societies of computees and scenarios have been set out in order to experimentally confirm (or disconfirm) them. These experiments concerns both *properties* of the model of computees (as set out in WP1, WP3 and WP5), and *properties* of the society model referring to interaction amongst computees (as set out in WP2, WP3 and WP5). Furthermore, specific experiments have been set out in order to test the proof theoretic approach proposed in WP5 for automatically proving or disproving protocol properties.

Evaluate experiments for properties: For each of the experiments set out, the behavior of the system have been then examined. Experiments have been also analyzed with the aim of discovering novel, relevant characteristics. The discovery of novel characteristics have a strong feedback on WP5, WP3, WP1 and WP2 (e.g., this lead to the need for improving the various underlying proof procedures).

Test scalability of prototype: The various experiments, when increasing in size, has helped to test the scalability of the platform resulting from WP4, and identify its limits.

9.2 Evaluation of WP6 and Methodology used

The evaluation criteria set out in deliverable D3 [44] were divided into success, extension (marked by +) and future direction (marked by *).

For this work-package, the criteria considered are as follows. For each of them, we report about evaluation of results achieved within WP6.

SYSTEM TESTING – After the settle of the underlying models, and the implementation of their operational counterpart and prototype developed, we identified how to test such a system. We have identified a number of system parameters of the SOCS framework and PROSOCS platform, and three main classes of experiments, in order to test computees and their societies.

Parameter identification – In order to test PROSOCS and experiment societies of computees on concrete scenarios and well-known interaction protocols, we have identified parameters to be varied, sufficient to capture the main features of the single computee model and society model, and the basic components in PROSOCS. For instance, we have varied the reasoning capabilities of the individual computees, in terms of underlying proof procedure implementation, the proof procedure for the society infrastructure, the scale of density of computees (and interactions) in a society, the behavior profile of individual computees, the kind of society protocol for the same scenario.

Parameter relation with GC criteria/features – Some of the identified parameters are related with the main features of the single computee model and society model, in a GC vision. For instance, the adoption of different profiles of behaviour and their experimentation (still ongoing work) is strictly related with heterogeneity of members, even if they are based upon the KGP model. Openness of members is also guaranteed, since we experimented with the society component as a verifier of interactions, not affecting members inner policies. Nonetheless, the link of system parameters to GC features is strictly related to the link between the KGP model and GC features and the link between the society model and GC features. We therefore refer to evaluation and self assessment of previous Workpackages (WP1 and WP2, but also WP3 and WP4) for this.

EFFICIENCY AND COMPUTATIONAL LIMITS – Experimentation concerning PROSOCS has tested the performance, suitability and computational limits of different implementation of the underlying proof procedures. Also, further experiments for properties and concrete scenarios has shown how to construct specific types of computees (e.g., the focussed behaviour profile computees) for specific types of societies. Nonetheless, a further experimentation and tuning of PROSOCS is required to deliver the platform.

DETECTING EXPECTED/UNDESIRABLE/EMERGING BEHAVIOUR – As behaviors are predicted and confirmed/disconfirmed experimentally or observed as emerging, we can establish a more general classification scheme according to the class of computees, their societies, and the variation of their system parameters. During WP6, we have experimented only part of the features available in the KGP model and in the social model developed within SOCS. Further activity is needed to have a complete test of all capabilities, and different profiles, as well to test all the possible protocols and social knowledge bases supported by the society model.

Disconfirm of properties – Given the catalogue of properties identified in WP5, we experimented with three notable profiles for computees, and also with two well-known protocols. For behavior profiling, we achieved the expected results. For the considered protocols, we have re-obtained results found in the literature pertaining their properties, but with a different approach.

Emergence of patterns^{+} – The focus is mainly on the experimental observation of simple characteristics, originated by interaction patterns among agents. The identification of notable patterns of interest was not achieved.

Relation of emerged patterns to system parameters* – The observation of emerging notable patterns of interest, can be related to identified system parameters. The identification of relations between emerged patterns and identified parameters was not achieved.

MODULAR AND PARAMETRIC SYSTEM – The high-level models for computees and societies are mapped into proof procedures and single components, then implemented in PROSOCS. Testing done along WP6 had also the effect of better tuning the single components (and their underlying proof procedures) of the PROSOCS platform. Furthermore, an extension of the platform itself was required, by introducing distributed objects in it. Modular profiles of computees were provided and added as library to the platform. Experimentation has required to identify first-class parameters influencing the overall behavior, which were tuned as work proceeded.

ANALYSIS AND REASONING OVER SYSTEM BEHAVIOR* – For the experimentation on concrete scenarios (the Combinatorial Auction, in particular) we had the chance of analyzing experimental results, and choose which architecture with specific parameters (e.g., basically reactive computees and the improved version of underlying proof procedure for the society infrastructure) was able to meet the required, and expected behavior.

RELATED WORK – We have compared (part of) the documented experiments with those documented in related work, when available. For some of the experiments, the related literature is mentioned as well, or references were given.

10 Published Papers/Demos pertaining WP6

In this section, we provide a list of publications and demos, contributing to workpackage WP6 of the project. For each publication or demo paper we reproduce the abstract and briefly comment on its contribution to the SOCS workpackage and WP6.

Most relevant papers to work done in the context of WP6 (mentioned in the documented experiments) are collected at the Web page: `http://www-lia.deis.unibo.it/Research/Projects/SOCS/guests/publications/D14.html`. Some of them are also common to work done in the context of WP5 (see Deliverable D13).

10.1 A Logic-based Approach to Reasoning with Beliefs about Trust (ICSTM)

[58] F, Sadri and F. Toni. "A Logic-based Approach to Reasoning with Beliefs about Trust". Proceedings of the IJCAR Workshop on Automated Reasoning for Security Protocols Analysis (ARSPA-2004), 2004.

Abstract. We adopt the abductive logic programming framework used for the knowledge representation and reasoning of SOCS computees to endow agents interacting within multi-agent systems with the capability to build up beliefs about the trustworthiness of other agents and to reason with these beliefs, e.g., to favor trustworthy agents when choosing who to negotiate with regarding a task or a resource. We illustrate our approach by means of several examples.

Contribution to WP6. The knowledge representation techniques suggested in this paper could be used to allow computees to reason about trustworthiness of other computees and thus allow trust-based interaction to be modelled within SOCS. It could also be used to support experiments, and to formalize properties related to trust. Thus, the work contributes to WP1, WP2, and potentially to WP5 and WP6.

10.2 Abductive Logic Programming with CIFF: System Description (ICSTM-DIPISA)

[25] U. Endriss, P. Mancarella, F. Sadri, G. Terreni, and F. Toni. "Abductive Logic Programming with CIFF: System Description". Proceedings of the 9th European Conference on Logic in Artificial Intelligence (JELIA-2004), 2004.

Abstract. We describe a system implementing CIFF, a novel extension of the IFF proof procedure for abductive logic programming, dealing with constraint predicates and with non-allowed abductive logic programs. Application areas of CIFF include various reasoning tasks underlying intelligent agents, including planning, reactivity, high-level communication, trust-mediated interaction and (various forms of) negotiation.

Contribution to WP6. This paper is relevant to WP6 as it describes the latest version of our implementation of CIFF, which has been guided by the results of our experiments with different variants of the procedure.

10.3 Compliance Verification of Agent Interaction: A Logic-based Software Tool (UNIBO-DIFERRARA)

[47] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello and P. Torroni. "Compliance Verification of Agent Interaction: A Logic-based Software Tool". Applied Artificial Intelligence, Taylor Francis. To appear., 2005.

Abstract. In open societies of agents, where agents are autonomous and heterogeneous, it is not realistic to assume that agents will always act so as to comply to interaction protocols. Thus, the need arises for a formalism to specify constraints on agent interaction, and for a tool able to observe and check for agent compliance to interaction protocols. In this paper we present a JAVA-PROLOG software component built on logic programming technology, which can be used to verify compliance of agent interaction to protocols, and that has been integrated with the PROSOCS platform.

Contribution to WP6. The focus of the paper is on the implementation of the social infrastructure of the PROSOCS demonstrator, which has been improved and used to perform experiments in WP6. A previous version of this work was presented at the Fourth International Symposium "From Agent Theory to Agent Implementation" (AT2AI-2004), 2004.

10.4 Learning Techniques for Automatic Algorithm Portfolio Selection (UNIBO)

[37] A. Guerri and M. Milano. "Learning Techniques for Automatic Algorithm Portfolio Selection". Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004), 2004.

Abstract. The purpose of this paper is to show that a well known machine learning technique based on Decision Trees can be effectively used to select the best approach (in terms of efficiency) in an algorithm portfolio for a particular case study: the Bid Evaluation Problem (BEP) in Combinatorial Auctions. In particular, we are interested in deciding when to use a Constraint Programming (CP) approach and when an Integer Programming (IP) approach, on the basis of the structure of the instance considered. Different instances of the same problem present a different structure, and one aspect (e.g. feasibility or optimality) can prevail on the other. We have extracted from a set of BEP instances, a number of parameters representing the instance structure. Some of them (few indeed) precisely identify the best strategy and its corresponding tuning to be used to face that instance. We will show that this approach is very promising, since it identifies the most efficient algorithm in 90% of the cases.

Contribution to WP6. This paper proposes the portfolio selection algorithm for the auction solver used in the experiments on combinatorial auctions carried out within WP6. Given a combinatorial auction instance to be solved, the portfolio selection algorithm is able to chose the best approach to face the instance in 90% of the cases.

10.5 A Demonstration of SOCS-SI (UNIBO-DIFERRARA)

[65] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, P. Torroni. “A Demonstration of SOCS-SI”. Demo Sessions of AAMAS-2004 and CILC-2004, 2004.

Abstract. The paper describes the demonstration of the SOCS-SI tool that has been given in the demo session of AAMAS-2004 and CILC-2004. The demonstration shows the concrete example of a “first price sealed bid” auction.

Contribution to WP6. The demonstration focuses on the SOCS-SI tool, which was developed as the verifier of compliance of computee interactions to interaction protocols, and has been improved and used in WP6 for experimentation.

10.6 Crafting the Mind of a PROSOCS Agent (ICSTM-UCY-CITY-DIPISA)

[16] A. Bracciali and N. Demetriou and U. Endriss and A. Kakas and W. Lu and K. Stathis. “Crafting the Mind of a PROSOCS Agent”. Applied Artificial Intelligence, Taylor Francis. To appear., 2005.

Abstract. PROSOCS agents are software agents that are built according to the KGP model of agency. KGP is used as a model for the mind of the agent, so that the agent can act autonomously using a collection of logic theories, providing the minds reasoning functionalities. The behaviour of the agent is controlled by a cycle theory that specifies the agents preferred patterns of operation. The implementation of the minds generic functionality in PROSOCS is worked out in such a way so it can be instantiated by the platform for different agents across applications. In this context, the development of a concrete example illustrates how an agent developer might program the generic functionality of the mind for a simple application.

Contribution to WP6. The focus of the paper is on the implementation of the KGP model in the PROSOCS demonstrator, which has been improved and used to perform experiments in WP6. A previous version of this work was presented at the Fourth International Symposium “From Agent Theory to Agent Implementation” (AT2AI-2004), 2004.

10.7 Ambient Intelligence using KGP Agents (ICSTM-UCY)

[67] F. Toni and K. Stathis. “Ambient Intelligence using KGP Agents”. Proceedings of the 2nd European Symposium for Ambient Intelligence, LNCS, pages 351-362, Springer-Verlag., 2005.

Abstract. We investigate the application of a logical model of agency, known as the KGP model, to develop agents for ambient intelligence applications. Using a concrete scenario, we illustrate how the logical formalism employed by a KGP agent allows a person to access the surrounding ambient through the agent in a transparent manner. We evaluate our claims by implementing the resulting interactions in PROSOCS, a prototype multi-agent systems platform that allows KGP agents to be deployed as components of ambient intelligence applications.

Contribution to WP6. The focus of the paper is on application of the KGP model and its implementation in PROSOCS demonstrator for the development of ambient intelligence applications.

10.8 Planning Partially for Situated Agents (DIPISA-ICSTM)

[48] Paolo Mancarella and Fariba Sadri and Giacomo Terreni and Francesca Toni. "Planning Partially for Situated Agents". http://www.doc.ic.ac.uk/ ft/PAPERS/clima04-planning.pdf, 2004.

Abstract. In recent years, within the planning literature there has been a departure from approaches computing *total plans* for given goals, in favor of approaches computing *partial plans*. Total plans can be seen as (partially ordered) sets of actions which, if executed successfully, would *lead* to the achievement of the goals. Partial plans, instead, can be seen as (partially ordered) sets of actions which, if executed successfully, would *contribute* to the achievement of the goals, subject to the achievement of further *sub-goals*. Planning partially (namely computing partial plans for goals) is useful (or even necessary) for a number of reasons: (i) because the planning agent is resource-bounded, (ii) because the agent has incomplete and possibly incorrect knowledge of the environment in which it is situated, (iii) because this environment is highly dynamic. In this paper, we propose a framework to design situated agents capable of planning partially. The framework is based upon the specification of planning problems via an abductive variant of the event calculus.

Contribution to WP6. This work provides a variant of the computational model of a planning computee. In particular it emphasizes the possibility of planning partially by means of a different treatment of preconditions of actions with respect to the D8 model and it proposes a new revision transition which, in practice, merges the Goal Revision and the Plan Revision transitions leading to a new State Revision transition as seen in D14.

10.9 Specification and Verification of Agent Interactions using Social Integrity Constraints (UNIBO-DIFERRARA)

[13] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, P. Torroni. "Specification and Verification of Agent Interactions using Social Integrity Constraints". Electronic Notes in Theoretical Computer Science, Vol. 85, n. 2. Elsevier Science., 2004.

Abstract. In this paper we propose a logic-based social approach to the specification and verification of agent interaction. We firstly introduce integrity constraints about social acts (called Social Integrity Constraints) as a formalism to express interaction protocols and to give a social semantics to the behavior of agents, focusing on communicative acts. Then, we discuss several possible kinds of verification of agent interaction, and we show how social integrity constraints can be used to verify some properties in this respect. We focus our attention on static verification of compliance of agent specifications to interaction protocols, and on run-time verification, based on agents' observable behavior. We adopt as a running example the NetBill security transaction protocol for the selling and delivery of information goods.

Contribution to WP6. The paper is relevant to WP6 in that it shows the application of an early version of the $\mathcal{S}$CIFF proof procedure to the on-the-fly verification of the NetBill security protocol.

10.10 The CHR-based Implementation of a System for Generation and Confirmation of Hypotheses (DIFERRARA-UNIBO)

[9] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma. "The CHR-based Implementation of a System for Generation and Confirmation of Hypotheses". The 19th Workshop on (Constraint) Logic Programming, 2005.

Abstract. Hypothetical reasoning makes it possible to reason with incomplete information in a wide range of knowledge-based applications. It is usually necessary to constrain the generation of hypotheses, so to avoid inconsistent sets or to infer new hypotheses from already made ones. These requirements are met by several abductive frameworks. In order to tackle many practical cases, however, it would also be desirable to support the dynamical acquisition of new facts, which can confirm the hypotheses, or possibly disconfirm them, leading to the generation of alternative sets of hypotheses.

In this paper, we present a system which supports the generation of hypotheses, as well as their confirmation or disconfirmation. We also describe the implementation of an abductive proof procedure, used as a reasoning engine for the generation and (dis)confirmation of hypotheses.

Contribution to WP6. The article is relevant to WP6 in that it describes the implementation of the $\mathcal{S}$CIFF proof procedure used for the experimentation done in the context of WP6.

10.11 Expressing Interaction in Combinatorial Auction through Social Integrity Constraints (DIFERRARA-UNIBO)

[7] Marco Alberti and Federico Chesani and Marco Gavanelli and Alessio Guerri and Evelina Lamma and Paola Mello and Paolo Torroni. "Expressing Interaction in Combinatorial Auction through Social Integrity Constraints". Intelligenza Artificiale. To appear., 2005.

Abstract. Combinatorial Auctions are an attractive application of intelligent agents; their applications are countless and are shown to provide good revenues. On the other hand, one of the issues they raise is the computational complexity of the solving process (the Winner Determination Problem, WDP), that delayed their practical use. Recently, efficient solvers have been applied to the WDP, so the framework starts to be viable.

A second issue, common to many other agent systems, is trust: in order for an agent system to be used, the users must trust both their representative and the other agents inhabiting the society: malicious agents must be found, and their violations discovered. The SOCS project addresses such issues, and provided a language, the social integrity constraints, for defining the allowed interaction moves, together with a proof procedure able to detect violations.

In this paper we show how to write a protocol for the combinatorial auctions by using social integrity constraints. In the devised protocol, the auctioneer interacts with an external solver for the winner determination problem.

Contribution to WP6. This work provides the implementation of a protocol for combinatorial auctions in the $\mathcal{S}$CIFF language. It discusses the protocol itself, and the integration of an external object, namely the auction solver, in the compliance check activity. It discusses experimentation and gives timing results both for the compliance check and for the combinatorial auction solving process.

10.12 A Computational Logic-based approach to security protocols verification, and its application to the Needham-Schroeder Public Key authentication protocol (UNIBO-DIFERRARA)

[11] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, P. Torroni. "A Computational Logic-based approach to security protocols verification, and its application to the Needham-Schroeder Public Key authentication protocol". Technical Report CS-2004-04, ENDIF, University of Ferrara. Available at `http://www.ing.unife.it/informatica/tr/`. Submitted., 2004.

Abstract. In this work we present a proof-theoretic approach to the verification of security protocols. Starting from the perspective of open multi-agent systems, where the internal architecture of the individual system's components may not be completely specified but it is important to infer and prove properties about the overall system behavior, we propose a general framework where several kinds of verification can be applied.

We take a formal approach based on Computational Logic, to tackle verification at two orthogonal levels: 'static' verification of protocol properties, and 'dynamic' verification of compliance of agent communication. The first one could investigate the possibility that while two agents are correctly following a given protocol a malicious agent can learn secrets exchanged by them. The second one could be about automatically checking that a group of agents interacting with each other are indeed following a given protocol.

The main advantages of our approach are (1) from a formal perspective, in the declarative protocol specification language, associated with a rigorous declarative and operational semantics, and (2) from an engineering perspective, in the reduction of the gap between protocol specification, verification and implementation. We study our approach in the domain of security protocols by presenting its application to the well-known Needham-Schroeder Public Key authentication protocol.

Contribution to WP6. The paper is relevant to WP6 as it describes experiments, performed in the context of this workpackage, on-the-fly conformance and automatic verification of protocol properties.

10.13 On the automatic verification of interaction protocols using g-$\mathcal{S}$CIFF (UNIBO-DIFERRARA)

[12] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, P. Torroni. "On the automatic verification of interaction protocols using g-$\mathcal{S}$CIFF". Technical Report LIA-04-2004, DEIS-LIA, University of Bologna. Submitted., 2004.

Abstract. Interaction protocol verification has been in recent years intensively investigated within and outside multi-agent research. Many techniques and models have been proposed based on a number of assumptions and choices, among which are the kind of knowledge available and the kind of properties that are subject of verification.

We focus on interaction protocol verification in open multi-agent systems, where the internal architecture of the individual agents is not known a-priori. In such a setting, while it is not possible to predict the behavior of individual agents, it is still possible to approach the interaction verification problem either by assuming that agents will indeed comply to the protocols, thus focussing on the study of properties of the interaction protocols, or by assuming that the agent behavior, if not predictable, is at least observable, and then focussing on the run-time verification of compliant behavior of interacting agents by reasoning on the messages they exchange with each other.

In this article, we build on previous work on logic-based specification and run-time verification of protocol compliance, and we extend the SOCS social framework towards verification of protocol properties. In doing so, we propose a unified formal and computational framework based on abductive logic programming which can be used to (*i*) design and specify interaction protocols that provably exhibit desirable properties and (*ii*) verify, at run-time, that agent interaction actually complies to the specified interaction protocols.

Contribution to WP6. This paper is relevant to WP6 in that it shows a case study of the automatic verification of protocol properties using the implementation of the g-$\mathcal{S}$CIFF proof procedure.

10.14 Expressing Interaction in Combinatorial Auction through Social Integrity Constraints (DIFERRARA-UNIBO)

[8] Marco Alberti, Federico Chesani, Marco Gavanelli, Alessio Guerri, Evelina Lamma, Michela Milano, and Paolo Torroni. "Expressing Interaction in Combinatorial Auction through Social Integrity Constraints". The 19th Workshop on (Constraint) Logic Programming, 2005.

Abstract. Combinatorial auctions are an interesting application of intelligent agents. They let the user express *complementarity* relations among the items for sale, and let the seller obtain higher revenues. On the other hand, the solving process, the so-called Winner Determination Problem (WDP) is NP-hard. This restricted the practical use of the framework, because of the fear to be, in some WDP instances, unable to meet reasonable deadlines. Recently, however, efficient solvers have been proposed, so the framework starts to be viable.

A second issue, common to many agent systems, is *trust*: in order for an agent system to be used, the users must *trust* both their representative and the other agents inhabiting the society. The SOCS project addresses such issues, and provided a language, the *social integrity constraints*, for defining the allowed interaction moves, and a proof-procedure able to detect violations.

In this paper we show how to write a protocol for combinatorial auctions by using social integrity constraints. In the devised protocol, the auctioneer interacts with an external solver for the winner determination problem. We also suggest extensions of the scenario, with more auctions in a same society, and suggest to verify whether two auctions interact. We also apply the NetBill protocol for the payment and delivery scheme.

Contribution to WP6. This paper presents experiments done in WP6 in the combinatorial auction scenario. Besides formalizing the auction protocol in the $\mathcal{S}$CIFF language, it gives timing results for various classes of experiments: the basic combinatorial auction, the double auction, and for the combinatorial auction with NetBill (where the actual delivery of goods is also considered).

Part IV
Conclusions

The integration of the single computee and society model in an engineered architecture grounded on Computational Logic (as resulting from WP1-WP4) is a promising approach in order to both encapsulate the heterogeneity of (societies of) computees and reason about their properties (WP5). Work-package WP6 has given some insights about benefits arising from this engineered approach, once system parameters have been identified and monitored, via a series of controlled experiments.

The experimentation in WP6 has been designed to shed light on the fundamental issues of research, validate the proposed models and their implementation, confirm or disconfirm their properties as proved in WP5 and, possibly, discover emerging characteristics.

WP6 has given feedback to all earlier work-packages, and in particular to the implementation of single components of PROSOCS platform, within WP4.

Further activity within WP6 for testing properties has been strictly linked with WP5 activity. In particular, in WP6 we have experimented with some properties of both individual computees, by varying their behaviour profile, and societies of computees by automatically proving properties for society protocols.

With respect to the specification and verification of single computees, the experimentation of the KGP model defined in WP1 and WP3 (see deliverable D4 and D8 [40, 34]), and implemented in WP4 (see deliverable D9 [4]), performed in the context of Workpackage WP6, has shown concretely the viability and usefulness of the proposed approach. The KGP model and its implementation have been applied to face planning problems, and to implement computees for the chosen scenarios. Different profiles of behaviour and capabilities (mainly, reactivity goal decision and planning) have been experimented.

With respect to the specification and verification of societies of computees, the experimentation of the society model defined in WP2 and WP3 (see deliverable D5 and D8 [50, 34]), and implemented in WP4 (see deliverable D9 [4]), performed in the context of Workpackage WP6, has confirmed the applicability and usefulness of a tool for on-the-fly verification of heterogenous and open systems, embedded into the $\mathcal{S}$CIFF proof procedure, and also the concrete applicability of a further tool, embedded into the g-$\mathcal{S}$CIFF proof procedure, for the automatic proof of properties of interaction protocols. Various society protocols have been considered in the experimentation, and collected in a Web site, subject to be extended. Furthermore, g-$\mathcal{S}$CIFF was applied to well-known protocols, for proving/disproving their properties.

With the GC vision in mind, we found it important to demonstrate experimentally that the models and tools developed in WP1-WP4 are flexible enough to address aspects such as heterogeneity, resource achievement, and scalability, often present in open environment. This was one of our main criteria when we had to choose experimental settings and scenarios. Combinatorial auctions are an example. They can be used to facilitate resource achievement, and can be tested by varying a number of parameters significant in the GC vision, like profiles and number of computees involved in the resource exchange process. Besides, the auction mechanism can itself be considered as a service, and the possibility to define different protocols for resource exchange is a demonstration of programmability of services. The library of behaviour profiles added to PROSOCS and experimented in WP6 allows one to develop his own computees by adopting one of them and taking advantage of these predefined, modular components. The rich library of protocols that we designed includes various auction mechanisms (like English auction, combinatorial auction and others) as well as various one-to-one resource exchange protocols (like FIPA Request Information protocol). The intuitive and easily readable specifications of such protocols demonstrates the flexibility of a fully declaratively programmable tool. This, together with the guaranteed verifiability of interaction in all our test cases, demonstrates that exporting the SOCS models into a universal scale is something feasible. The diversity of tested protocols demonstrates the SOCS model's ability to help the integration of an open and global computing reality, where entities can live with pre-existing interaction mechanisms and a new associated social semantics. Incrementally, diverse (and possibly incompatible) implementations can in this way move towards integration and global communication.

Nonetheless, experimentation has also shed the light on problems pertaining the various components of PROSOCS, and their integrated working in a single platform. As experimentation proceeded for testing scalability and performances, we have ameliorated the implementation of the underlying proof procedures (CIFF and $\mathcal{S}$CIFF). However, runs for integrated examples are still limited in size, and further work is required before a real deliver of the platform.

As concerns the demo that will be given at the next review meeting, examples to be demonstrated are available at the Web site `http://www.lia.deis.unibo.it/Research/Projects/SOCS/partners/experimentation/`. We will demonstrate examples for the Combinatorial Auction scenario, and also show runs with different behaviour profiles.

A Annexed documents

The following two documents are annexed to this deliverable.

A.1 Experiments on Combinatorial Auctions (UNIBO-DIFERRARA-CITY-ICSTM)

[6] M. Alberti, M. Alberti, F. Chesani, U. Endriss, M. Gavanelli, A. Guerri, E. Lamma, W. Lu, P. Mello, M. Milano, K. Stathis, F. Toni, P. Torroni. "Experiments on Combinatorial Auctions". D14 Annex, SOCS Consortium, 2005.

Abstract. This paper contains the description of experiments done so far in the combinatorial auction scenario. We describe the computee side and the society side. In this context we introduce also *external objects* providing services to the computees themselves, and in particular implementing a complex optimization algorithm. Experimental results are given for a traditional combinatorial auction and for two variants of it: a double auction and an auction plus the successive payment with Netbill. Also, some experiments on non combinatorial auctions will be provided in this document.

Contribution to WP6. The paper documents the experiments on several auction scenarios that have been performed in WP6.

A.2 Computee Profiles in PROSOCS (CITY-UCY)

[1] A. Yip, A. Kakas and K. Stathis. "Computee Profiles in PROSOCS". D14 Annex, SOCS Consortium, 2005.

Abstract. This document describes the methodology used to develop profiles of behaviour, and the enhancement to the PROSOCS platform in order to support the profiles. A profile is simply a purpose built Cycle Theory incorporating a specific behaviour, differing from the Normal Cycle. A full collection of implemented profiles is presented. Typically a behaviour profile is defined as an extension to the Normal Cycle. The extension is then defined as a collection of desired behaviour in logic.

Contribution to WP6. The paper documents the implementation of different profiles, experimented in WP6.

References

[1] A. K. A. Yip and K. Stathis. Computee profiles in prosocs. Discussion note, SOCS Consortium, Feb. 2005. `http://lia.deis.unibo.it/Research/Projects/SOCS/`.

[2] M. Alberti, F. Athienitou, A. Bracciali, F. Chesani, U. Endriss, M. Gavanelli, A. Kakas, E. Lamma, W. Lu, P. Mancarella, P. Mello, F. Sadri, K. Stathis, F. Toni, and P. Torroni. Verifiable properties of societies of computees. Technical report, SOCS Consortium, 2005. Deliverable D13.

[3] M. Alberti, A. Bracciali, F. Chesani, N. Demetriou, U. Endriss, W. Lu, F. Sadri, A. Kakas, E. Lamma, P. Mello, M. Milano, K. Stathis, F. Toni, and P. Torroni. Examples of the functioning of computees and their societies. Discussion Note IST3250/ICSTM//DN/I/a2, SOCS Consortium, Dec. 2003. `http://lia.deis.unibo.it/Research/Projects/SOCS/`.

[4] M. Alberti, A. Bracciali, F. Chesani, U. Endriss, M. Gavanelli, W. Lu, K. Stathis, and P. Torroni. SOCS prototype. Technical report, SOCS Consortium, 2003. Deliverable D9.

[5] M. Alberti, A. Bracciali, F. Chesani, U. E. M. Gavanelli, A. Guerri, A. Kakas, E. Lamma, P. Mancarella, P. Mello, M. Milano, F. Riguzzi, F. Sadri, K. Stathis, G. Terreni, F. Toni, and P. Torroni. Update report on wp1–wp6. Technical report, SOCS Consortium, 2004. Deliverable D12.

[6] M. Alberti, F. Chesani, U. Endriss, M. Gavanelli, A. Guerri, E. Lamma, W. Lu, P. Mello, M. Milano, K. Stathis, F. Toni, and P. Torroni. Experiments on (combinatorial) auctions. Discussion Note IST32530/UNIBO/250/DN/I/a1, SOCS Consortium, Dec. 2005. `http://lia.deis.unibo.it/Research/Projects/SOCS/`.

[7] M. Alberti, F. Chesani, M. Gavanelli, A. Guerri, E. Lamma, P. Mello, and P. Torroni. Expressing interaction in combinatorial auction through social integrity constraints. *Intelligenza Artificiale*, 2005. To appear.

[8] M. Alberti, F. Chesani, M. Gavanelli, A. Guerri, E. Lamma, M. Milano, and P. Torroni. Expressing interaction in combinatorial auction through social integrity constraints. In Wolf et al. [69], pages 53–64.

[9] M. Alberti, F. Chesani, M. Gavanelli, and E. Lamma. The CHR-based Implementation of a System for Generation and Confirmation of Hypotheses. In Wolf et al. [69], pages 111–122.

[10] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Compliance verification of agent interaction: a logic-based tool. In Trappl [68], pages 570–575. Extended version to appear in a special issue of Applied Artificial Intelligence, Taylor & Francis, 2005.

[11] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. A Computational Logic-based approach to security protocols verification, and its application to the Needham-Schroeder Public Key authentication protocol. Technical Report CS-2004-04, ENDIF, Università degli Studi di Ferrara, December 2004. Available at `http://www.ing.unife.it/informatica/tr/`.

[12] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. On the automatic verification of interaction protocols using g-$\mathcal{S}$CIFF. Technical Report DEIS-LIA-04-004, University of Bologna (Italy), 2005. LIA Series no. 72.

[13] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Specification and verification of agent interactions using social integrity constraints. *Electronic Notes in Theoretical Computer Science*, 85(2), 2003.

[14] J. J. Alferes and J. A. Leite, editors. *JELIA*, volume 3229 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2004.

[15] L. Amgoud, S. Parsons, and N. Maudet. Arguments, dialogue and negotiation. In W. Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence*. IOS Press, Aug. 2000.

[16] A. Bracciali, N. Demetriou, U. Endriss, A. Kakas, W. Lu, and K. Stathis. Crafting the Mind of a PROSOCS Agent. *Applied Artificial Intelligence*, 2005. To appear.

[17] A. Bracciali, A. C. Kakas, E. Lamma, P. Mello, K. Stathis, F. Toni, and P. Torroni. D11: Evaluation and self assessment. Technical report, SOCS Consortium, 2003. Deliverable D11.

[18] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.

[19] J. Collins and M. Gini. An integer programming formulation of the bid evaluation problem for coordinated tasks. In B. Dietrich and R. V. Vohra, editors, *Mathematics of the Internet: E-Auction and Markets*, volume 127 of *IMA Volumes in Mathematics and its Applications*, pages 59–74. Springer-Verlag, New York, 2001.

[20] B. Cox, J. Tygar, and M. Sirbu. Netbill security and transaction protocol. In *Proceedings of the First USENIX Workshop on Electronic Commerce*, New York, July 1995.

[21] A. C. Y. D. Dolev. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(29):198–208, 1983.

[22] S. de Vries and R. V. Vohra. Combinatorial auctions: A survey. *INFORMS Journal on Computing*, (3):284–309, 2003.

[23] N. Demetriou, A. Kakas, and P. Torroni. Further examples for the functioning of computees. Discussion Note IST32530/UCY//DN/I/a2, SOCS Consortium, Jan. 2004. `http://lia.deis.unibo.it/Research/Projects/SOCS/`.

[24] C. Dixon, M.-C. Fernández Gago, M. Fisher, and W. van der Hoek. Using temporal logics of knowledge in the formal verification of security protocols. In *Procedings of the Eleventh International Workshop on Temporal Representation and Reasoning (TIME'04)*, 2004.

[25] U. Endriss, P. Mancarella, F. Sadri, G. Terreni, and F. Toni. Abductive logic programming with CIFF: System description. In Alferes and Leite [14], pages 680–684.

[26] U. Endriss, P. Mancarella, F. Sadri, G. Terreni, and F. Toni. The CIFF proof procedure: Definition and soundness results. Technical Report 2004/2, Department of Computing, Imperial College London, May 2004.

[27] U. Endriss, P. Mancarella, F. Sadri, G. Terreni, and F. Toni. The CIFF proof procedure for abductive logic programming with constraints. In Alferes and Leite [14], pages 31–43.

[28] M. Esteva, D. de la Cruz, and C. Sierra. ISLANDER: an electronic institutions editor. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part III*, pages 1045–1052, Bologna, Italy, July 15–19 2002. ACM Press. `http://portal.acm.org/ft_gateway.cfm?id=545069&type=pdf&dl=GUIDE&dl=ACM&CFID=4415868&CFTOKEN=57395936`.

[29] J. G. Fanjul. Verification and simulation of the netbill protocol using spin. In *SPIN98 - Papers from the 4th International SPIN Workshop*, 1998.

[30] Proposal for borda count interaction protocol. Home page:http://www.fipa.org/docs/input/f-in-00092/f-in-00092.htm.

[31] T. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming*, 37(1-3):95–138, Oct. 1998.

[32] Y. Fujishima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, volume 1, pages 548–553. Morgan Kaufmann Publishers, 1999.

[33] T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 33(2):151–165, Nov. 1997.

[34] M. Gavanelli, E. Lamma, P. Torroni, P. Mello, K. Stathis, P. Moraïtis, A. C. Kakas, N. Demetriou, G. Terreni, P. Mancarella, A. Bracciali, F. Toni, F. Sadri, and U. Endriss. Computational model for computees and societies of computees. Technical report, SOCS Consortium, 2003. Deliverable D8.

[35] F. Guerin and J. Pitt. Proving properties of open agent systems. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II*, pages 557–558, Bologna, Italy, July 15–19 2002. ACM Press. `http://portal.acm.org/ft_gateway.cfm?id=544873&type=pdf&dl=GUIDE&dl=ACM&CFID=4415868&CFTOKEN=57395936`.

[36] A. Guerri and M. Milano. Exploring CP-IP based techniques for the bid evaluation in combinatorial auctions. In F. Rossi, editor, *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 863–867. Springer-Verlag, 2003.

[37] A. Guerri and M. Milano. Learning techniques for automatic algorithm portfolio selection. In R. Lopez de Mantaras and L. Saitta, editors, *Proceedings of the 16th European Conference on Artificial Intelligence*. IOS Press, Aug. 2004. to appear.

[38] ILOG S.A., France. *ILOG Solver*, 5.0 edition, 2003.

[39] N. R. Jennings. Automated haggling: Building artificial negotiators (invited talk). In *AISB'01 Convention, York, UK*, March 2001. Electronically available, `http://www.ecs.soton.ac.uk/~nrj/download-files/negotiation.pdf`.

[40] A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. A logic-based approach to model computees. Technical report, SOCS Consortium, 2003. Deliverable D4.

[41] A. C. Kakas and P. Mancarella. On the relation between Truth Maintenance and Abduction. In T. Fukumura, editor, *Proceedings of the 1st Pacific Rim International Conference on Artificial Intelligence, PRICAI-90, Nagoya, Japan*, pages 438–443. Ohmsha Ltd., 1990.

[42] A. C. Kakas, B. van Nuffelen, and M. Denecker. A-System: Problem solving through abduction. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 591–596, Seattle, Washington, USA, August 2001. Morgan Kaufmann Publishers.

[43] P. Küngas and M. Matskin. Linear logic, partial deduction and cooperative problem solving. In J. A. Leite, A. Omicini, L. Sterling, and P. Torroni, editors, *Declarative Agent Languages and Technologies*, volume 2990 of *Lecture Notes in Artificial Intelligence*, pages 263–279. Springer-Verlag, May 2004. First International Workshop, DALT 2003. Melbourne, Australia, July 2003. Revised Selected and Invited Papers.

[44] E. Lamma, P. Mello, P. Mancarella, A. Kakas, K. Stathis, and F. Toni. Self-assessment: parameters and criteria. Technical report, SOCS Consortium, 2003. Deliverable D3. Distribution restricted to the GC programme.

[45] G. Lowe. Breaking and fixing the Needham-Shroeder public-key protocol using CSP and FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems: Second International Workshop, TACAS'96*, volume 1055 of *Lecture Notes in Artificial Intelligence*, pages 147–166. Springer-Verlag, 1996.

[46] W. Lu and K. Stathis. Incorporating objects in PROSOCS artificial worlds. Technical report, SOCS Consortium, 2004. IST32530/CITY/015/DN/I/b2.

[47] M. G. E. L. P. M. M. Alberti, F. Chesani and P. Torroni. Compliance Verification of Agent Interaction: A Logic-based Software Tool. *Applied Artificial Intelligence*, 2005. To appear.

[48] P. Mancarella, F. Sadri, G. Terreni, and F. Toni. Planning partially for situated agents. In J. Leite and P. Torroni, editors, *5th Workshop on Computational Logic in Multi-Agent Systems (CLIMA V)*, Sept. 29-30 2004.

[49] P. Mello, M. Milano, A. Roli, R. Montanari, M. Gavanelli, E. Lamma, and F. Riguzzi. Combinatorial Auctions. Technical Report IST32530/UNIBO/0XX/IN/I/a2, SOCS Consortium, 2002.

[50] P. Mello, P. Torroni, M. Gavanelli, M. Alberti, A. Ciampolini, M. Milano, A. Roli, E. Lamma, F. Riguzzi, and N. Maudet. A logic-based approach to model interaction amongst computees. Technical report, SOCS Consortium, 2003. Deliverable D5.

[51] S. Merz. Model checking: A tutorial overview. In F. Cassez, C. Jard, B. Rozoy, and M.D.Ryan, editors, *Modeling and Verification of Parallel Processes*, number 2067 in Lecture Notes in Computer Science, pages 3–38. Springer-Verlag, 2001.

[52] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[53] N. Nisan. Bidding and allocation in combinatorial auctions. pages 1–12. ACM Press, 2000.

[54] J. Postel. Transmission control protocol. RFC 793, IETF, September 1981.

[55] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers.* MIT Press, Cambridge, Massachusetts, 1994.

[56] M. Rothkopf, A. Pekec, and R.M.Harstad. Computationally manageable combinational auctions. *Management Science*, 44(8):1131–1147, 1998.

[57] A. Russo, R. Miller, B. Nuseibeh, and J. Kramer. An abductive approach for analysing event-based requirements specifications, 2002.

[58] F. Sadri and F. Toni. A logic-based approach to reasoning with beliefs about trust. In *Proceedings ARSPA04, Workshop on Automated Reasoning for Security Protocols Analysis, affiliated to IJCAR04*. Cork, July 2004. To appear.

[59] F. Sadri, F. Toni, and P. Torroni. Logic agents, dialogues and negotiation: an abductive approach. In *Proceedings AISB'01 Convention, York, UK*, Mar. 2001.

[60] Y. Sakurai, M. Yokoo, and K. Kamei. An efficient approximate algorithm for winner determination in combinatorial auctions. In *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC-00)*, pages 30–37, 2000.

[61] T. Sandholm. Agents in electronic commerce: Component technologies for automated negotiation and coalition formation. *Journal of Autonomous Agents and Multi-Agent Systems (Special Issue on Best of ICMAS 1998)*, 3(1):73–96, 2000.

[62] T. Sandholm. Algorithm for optimal winner determination in combinatorial auction. *Artificial Intelligence*, 135, 2002.

[63] SICStus prolog user manual, release 3.11.0, Oct. 2003. `http://www.sics.se/isl/sicstus/`.

[64] C. Sierra and P. Noriega. Agent-mediated interaction. From auctions to negotiation and argumentation. In M. d'Inverno, M. Luck, M. Fisher, and C. Preist, editors, *Foundations and Applications of Multi-Agent Systems, UKMAS Workshop 1996-2000, Selected Papers*, volume 2403 of *Lecture Notes in Computer Science*, pages 27–48. Springer-Verlag, 2002.

[65] A demonstration of SOCS-$\mathcal{S}$I for AAMAS'04. Demo storyboard available at the URL: `http://lia.deis.unibo.it/Research/aamas2004demo/`.

[66] K. Stathis, A. C. Kakas, W. Lu, N. Demetriou, U. Endriss, and A. Bracciali. PROSOCS: a platform for programming software agents in computational logic. In Trappl [68], pages 523–528. Extended version to appear in a special issue of Applied Artificial Intelligence, Taylor & Francis, 2005.

[67] K. Stathis and F. Toni. Ambient Intelligence using KGP Agents. In *Proceedings of the 2nd European Symposium for Ambient Intelligence*, pages 351–362, Eindhoven, November 2004. LNCS, Springer-Verlag.

[68] R. Trappl, editor. *Proceedings of the 17th European Meeting on Cybernetics and Systems Research, Vol. II, Symposium "From Agent Theory to Agent Implementation" (AT2AI-4)*. Austrian Society for Cybernetic Studies, Vienna, Austria, Apr. 13-16 2004.

[69] A. Wolf, T. T. Frühwirth, and M. Meister, editors. *Proceedings of the 19th Workshop on (Constraint) Logic Programming*, Ulm, Germany, February 2005. University of Ulm.