SCALABLE COMPUTING
Practice and Experience

Special Issue: Foundational Underpinnings for Pragmatic Agent-based Systems

Editors: Marcin Paprzycki, Niranjan Suri

Volume 8, Number 1, March 2007

ISSN 1895-1767

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 8, Number 1, March 2007

TABLE OF CONTENTS

Software Review:

Book Review:

Scalable Computing: Practice and Experience
Volume 8, Number 1, p. i. http://www.scpe.org
ISSN 1895-1767

EDITORIAL: VIETNAM ON ITS WAY TO HIGH PERFORMANCE COMPUTING

In March 2007 I had the pleasure to participate in the International Workshop on Advanced Computing and Applications (ACOMP 2007) held in Ho Chi Minh City, the former Saigon in Vietnam. Neither was it the first workshop there nor was it my first visit to Ho Chi Minh City. The workshop series started in 2001 with follow-ups in 2002, 2004, and 2005. In the years 2003 and 2006 our Vietnamese colleagues organised a conference held in Hanoi, the capital of Vietnam. While this conference was more focused on mathematical optimization, the Spring workshops have always concentrating on parallel and distributed computing. These conferences are a results from a cooperation between the Interdisciplinary Centre of Scientific Computing (IWR) at the University Heidelberg, Germany, and colleagues in Hanoi and Saigon. For more than a decade Prof. Georg Bock at the IWR and Prof. Hoang Xuan Phu (Hanoi) and Prof. Nguyen Thanh Son (HCMC) have been the driving force behind this fruitful cooperation.

While attending the workshops in 2001 and 2002 I was impressed by the creativity and focused hard work of the researchers in Vietnam. ăThis year's contributions to the workshop highlight a new trend in Vietnamese computer science, i.e. its orientation of research towards Grid Computing.

Remember, it is only since 1975 that the country lives in peace after the Vietnam war. In 1986 the governing Communist Party made a shift to an open market and installed the so-called Doi Moi (renovation) politics. The country developed rapidly, making Vietnam one of the fastest growing economies world-wide. In the 1980's gifted scientists studied in foreign countries and conducted research there in order to later go back and build up their home country. Many of them went to socialist brother countries, in particular also to the former German Democratic Republic.

The faculty for computer science at the Ho Chi Minh City University of Technology (HCMUT) was established in 1986. The university itself has its foundation in the 50s however was named after Ho Chi Minh only after the end of the war. It is now the leading university in teaching and research activities of Vietnam. However, Southern HCMC University after the end of the war received a high number of teaching and management staff from the Northern Hanoi University of Technology. Remember that the communist North Vietnam won the war and the South was reunified with the North in 1976. Being German I see parallels to our academic life after the reunification of West and East Germany in 1990.

Although Hanoi University of Technology also conducts education and research in computer science, the focus on high performance computing is a speciality of the HCMC University. Having 25.000 students in total, the computer science faculty with its 1.500 students plays an important role on the campus. It embraces 7 research groups on different fields like e.g. chip design and data mining. Parallel processing and network computing is headed by Dr. Nguyen Thanh Son, who is now the Vice Rector of the University of Technology and Dr. Nam Thoai.

At the workshop they presented their work in Grid computing in five talks. This was complemented by about the same number of invited keynote talks given by speakers who play an important role in this field like Satoshi Sekiguchi, Dieter Kranzlmueller and others. Under the guidance of HCMUT and with financial support by the Vietnamese Ministery for Science and Technology, the Vietnamese researchers plan to set up a national Grid infrastructure (VN-Grid Initiative).

The EDAGrid-project at HCMUT aims at providing the necessary software components and organizational concepts. Based on the Globus Toolkit 4.x it defines a middleware for service-centric applications. As the telecommunication infrastructure of Vietnam is not yet that powerful as in other countries, the first step will be to define so-called fat Grid nodes at the major universities of Vietnam. HCMUT contributes the Supernode II cluster, which comprises 64 nodes with two processors each. The software is well-known to us: GT 4.x, PBS, LSF and others. Grid based projects at HCMUT focus on data management and on data mining. They run cooperations with researchers from e.g. civil engineering, chemistry and aerospace technology.

So Vietnam is quickly catching up with the international Grid community—and not only with this one. We are looking forward to fruitful discussions and cooperations with our Vietnamese colleagues and of course also hope to see submissions to SCPE.

http://www.hcmut.edu.vn/
http://www.cse.hcmut.edu.vn/
http://www.cse.hcmut.edu.vn/ACOMP2007/

Thomas Ludwig,
Universität Heidelberg.

Scalable Computing: Practice and Experience
Volume 8, Number 1, p. iii. http://www.scpe.org
ISSN 1895-1767

GUEST EDITOR'S INTRODUCTION.

This issue is the second of a two issue collection of selected papers from the AIMS (Agents, Interactions, Mobility, and Systems) conference track. AIMS began in 2002 as part of the ACM SAC (Symposium on Applied Computing) and continued for five years. The first conference was held in Madrid (Spain). Subsequent conferences were held in Melbourne (Florida, USA), Nicosia (Cyprus), Santa Fe (New Mexico, USA), and Dijon (France). The track was primarily created to provide a venue for applied topics in software agents, but became the only venue for papers on mobile agents, as the IEEE conference on Mobile Agents was discontinued after 2002. The first issue focused on papers related to mobile agents. This second issue focuses on software agents and contains eight papers.

In the first paper, Alberti and colleagues address the problem of verifying agent interaction protocols that dictate how agents communicate with each other in a multi-agent system. They propose a system based on Prolog that enforces Social Integrity Constraints—that govern how agents interact with other agents. They also apply their approach to the standard FIPA Contract-Net protocol.

In the second paper, Meneguzzi, Zorzo, Costa Móra, and Luck discuss how to incorporate planning into the Belief, Desires, Intentions (BDI) model based agent systems. Their approach attempts to supplement the BDI model with a planning approach in order to provide efficient means-end reasoning. A hybrid system with a blend of programming platforms integrates reasoning and graphplan generation. This integration addresses the long awaited requirement for BDI pragmatism and provides a novel technological framework.

The third paper by Albayrak, Wollny, Lommatzsch, and Milosevic describes an application of agent technology to information filtering. Their system uses information agents to retrieve content from a number of diverse sources including the web. This information is then filtered for individual users via personal agents, based on the user profiles. They also describe their implementation to support browsing information via PDAs and cellphones.

In the fourth paper, Hexmoor and Mclaughlan address the issue of adjustable autonomy in the context of the Personal Satellite Assistant (PSA)—a softball sized flying robot onboard the space station. The authors propose a computational approach to adjustable autonomy, which considers the tradeoffs between human intervention and guidance to an agent versus the agent's own autonomous behavior.

The fifth paper by Peña and colleagues address the problem of protocol design for multi-agent systems. Unlike the logic-based approach adopted by the first paper, this approach proposes a top-down mechanism for designing the protocols. They model the protocols with FSAs that are successively refined until they are reduced to simple message sequences.

In the sixth paper, Zhang proposes an approach to proactive communication to improve performance of multi-agent teamwork. The goal is to allow agents cooperating in a team to anticipate each other's information needs in a proactive manner and to communicate the information to other agents.

In the seventh paper, Lomonosov and Sitharam discuss the tradeoff between stability, optimality, and complexity for network games. Their approach is based on the Nash equilibrium, with stability being defined as the ability to reach a Nash equilibrium and optimality being defined as the distance between to the equilibrium solution and an optimal solution.

The paper by Carlsson and Jönsson discusses cooperative strategies and their application to the iterated prisoner's dilemma and the chicken game.

Decision support systems for resource management in the corporate world require sophisticated administration and management. Multiagent systems approach to this topic is the tenet of the paper by Symeonidis, et. al., in this issue. Customer management as well as resource tracking and recommendation are core issued discussed.

Marcin Paprzycki,
Niranjan Suri.

Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 1–13. http://www.scpe.org
ISSN 1895-1767

SPECIFICATION AND VERIFICATION OF AGENT INTERACTION PROTOCOLS IN A LOGIC-BASED SYSTEM*

MARCO ALBERTI, FEDERICO CHESANI, DAVIDE DAOLIO, MARCO GAVANELLI, EVELINA LAMMA, PAOLA MELLO AND PAOLO TORRONI

Abstract.
A number of information systems can be described as a set of interacting entities, which must follow interaction protocols. These protocols determine the behaviour and the properties of the overall system, hence it is of the uttermost importance that the entities behave in a conformant manner.

A typical case is that of multi-agent systems, composed of a plurality of agents without a centralized control. Compliance to protocols can be hardwired in agent programs; however, this requires that only "certified" agents interact. In open systems, composed of autonomous and heterogeneous entities whose internal structure is, in general, not accessible (open agent societies being, again, a prominent example) interaction protocols should be specified in terms of the *observable* behaviour, and compliance should be verified by an external entity.

In this paper, we propose a Java-Prolog-*CHR* system for verification of compliance of computational entities to protocols specified in a logic-based formalism (*Social Integrity Constraints*). We also show the application of the formalism and the system to the specification and verification of three different scenarios: two specifications show the feasibility of our approach in the context of Multi Agent Systems (FIPA Contract-Net Protocol and Semi-Open societies), while a third specification applies to the specification of a lower level protocol (Open-Connection phase of the TCP protocol).

1. Introduction. Many information systems can be described as a set of mutually independent, interacting entities. A typical example is that of multi-agent systems. In such a scenario the interaction is usually subject to some kind of interaction protocols, which the agents should respect when interacting. This raises the obvious problem of verifying that interaction protocols are actually followed.

It is possible to design agents so that they will "spontaneously" comply to protocols, and, if possible, formally verify that at design time. For instance, in [13], Endriss et al. propose an approach where protocols are "imported" into individual agent policies.

However, this approach is not viable in open[1] agent societies, where interacting agents are autonomous and heterogeneous and, in general, their internal structure cannot be accessed. In this case, agents should be checked for compliance to interaction protocols based on their *observable* behaviour, by a trusted external entity.

In previous work [5], we proposed a computational logic-based formalism (based upon *Social Integrity Constraints*, SICs) to specify interaction protocols. Social Integrity Constraints are meant to constrain the agent observable behaviour rather than agents' internal (mental) state or policies. In other words, this approach does not restrict an agent's access to societies based on its internal structure; regardless of its policies, any agent can successfully interact in a society ruled by SICs, as long as its behaviour is compliant. The formal semantics of Social Integrity Constraints [4] is based on abductive logic programming [18].

The purpose of this paper is to demonstrate the viability of Social Integrity Constraints as a formalism to specify interaction between computational entities, including, but not limited to, agents in open societies. We will use a modified version of Social Integrity Constraints, which better fits our needs in terms of both simplicity of presentation, and expressiveness.

The paper is structured as follows. In Sect. 2, we introduce the version of Social Integrity Constraints used in this work, giving their syntax and an informal explanation of their semantics.

In Sect. 3 we specify in terms of SICs a contract net-based protocol for resource allocation and negotiation in multi-agent systems, called FIPA CNP, and in Sect. 4 we specify a protocol for entering "semi-open" societies, i. e., virtual environments characterized by the presence of a "gatekeeper" agent and a protocol that governs the agents' access to the society. In Sect. 5 we demonstrate the usage of SICs to specify a network communication protocol, namely the three-way handshake opening of the TCP Internet Protocol.

The article ends with the presentation of the compliance verification system (Sect. 6), and some notes about its Java+Prolog implementation.

*This article is an extended version of the one by Alberti, Daolio, Gavanelli, Lamma, Mello, and Torroni, published in Haddad, Omicini, and Wainwright, eds., *Proceedings of the 19th ACM Symposium on Applied Computing, SAC 2004, Special Track on Agents, Interactions, Mobility, and Systems (AIMS). Nicosa, Cyprus, March 14-17, 2004.* pp. 72-78. ACM Press (2004).

[1]We intend *openness* in societies of agents as Artikis, Pitt and Sergot [7], where agents can be heterogeneous and possibly non-cooperative.

2. Social Integrity Constraints. We distinguish between actual behaviour (*happened events*) and desired behaviour (*expectations*), since in non-ideal situations they do not always coincide. In this section, we let the reader get acquainted with our representation of events and we introduce Social Integrity Constraints (SICs) as a formalism used to express which expectations are generated as consequence of happened events.

Happened Events and Expectations. Happened events are in the form

$$\mathbf{H}(\mathit{Description}, \mathit{Time})$$

where *Description* is a term (as intended in logic programming, see [20]) representing the event that has happened, and *Time* is an integer number representing the time at which the event has happened. For example,

$$\mathbf{H}(\mathit{request}(a_i, a_j, \mathit{give}(10\$), d_1), 7)$$

represents the fact that agent a_i requested agent a_j to give 10\$, in the context of interaction d_1 (dialogue identifier) at time 7.

All happened events form the history of a society. Given the history of a society at a given time, some events will have to happen in order for interaction protocols to be satisfied: we represent such events by means of *expectations*, which can be *positive* or *negative*. Positive expectations are of the form

$$\mathbf{E}(\mathit{Description}, \mathit{Time})$$

and represent an event that is expected to happen (typically, an action that an agent is expected to take). Negative expectations are of the form

$$\mathbf{EN}(\mathit{Description}, \mathit{Time})$$

and represent the fact that an event is expected *not* to happen.

Expectations may (and, typically, will) contain variables, to reflect the fact that the expected event is not fully specified; however, CLP [17] constraints can be imposed on variables to restrict their domain. For instance,

$$\mathbf{E}(\mathit{accept}(a_k, a_j, \mathit{give}(M), d_2), T_\mathrm{a}) : M \geq 10, T_\mathrm{a} \leq 15 \tag{2.1}$$

represents the expectation for agent a_k to *accept* giving agent a_j an amount M of money, in the context of interaction d_2 at time T_a; moreover, M is expected to be at least 10\$, and T_a to be at most 15.

Since we impose no restrictions on the *Description* term of an expectation, expectations can regard any kind of event that can be expressed by a Prolog-like term. However, expectations only regard point-time events; thus it is not possible to express concisely that some *proposition* is expected to be true along a given time interval.

Since we make no assumptions about the agents' internal structure or policies, their behaviour may or may not satisfy expectations. We represent these two cases by means of the notions of *fulfillment* and *violation*. We say that an event *matches* an expectation if and only if:

- their contents unify (à la Prolog);
- all relevant CLP constraints on variables (if any) are satisfied.

A positive expectation can get *fulfilled* by a matching event, whereas a negative expectation can get *violated* by a matching event.

For instance, event

$$\mathbf{H}(\mathit{accept}(a_k, a_j, \mathit{give}(20), d_2), 15)$$

fulfills expectation (2.1); the same event would, instead, violate a negative expectations with the same content and CLP constraints.

If we assume at some point that no more events will ever occur, we say that the history is *closed*. In that case, all positive expectations that are not fulfilled are violated, and all negative expectations that are not violated are fulfilled.

TABLE 2.1
BNF syntax of Social Integrity Constraints

SIC	::= $\chi \rightarrow \phi$
χ	::= EventLiteral [$\wedge$ EventLiteral]* [:CList]
ϕ	::= PriorityLevel [$\Rightarrow$ PriorityLevel]*
PriorityLevel	::= HeadDisjunct [$\vee$ HeadDisjunct]*, P
EventLiteral	::= H(Term,T)
HeadDisjunct	::= Expectation [$\wedge$ Expectation]* [:CList]
Expectation	::= E(Term,T) | EN(Term,T)

Social Integrity Constraints. The way expectations are generated, given a (partial) history of a society, is specified by *Social Integrity Constraints* (SICs). In this article, we adopt a modified version of the SICs introduced in [2] (we discuss and motivate such modifications in Sect. 7).

Table 2.1 reports the BNF syntax of SICs. *Term* is a logic programming term [20], P is an integer number and T is a variable symbol or integer number. *CList* is a conjunction of CLP constraints on variables.

SICs are a kind of forward rules, stating what expectations should be generated on the basis of happened events. By means of SICs, it is possible to express that conjunctions of expectations (*HeadDisjuncts* in Table 2.1) are alternative, and it is also possible to assign a priority, represented by an integer number, to each list of alternatives (*PriorityLevel*s in Table 2.1).

For instance, the following SIC:

$$\begin{aligned} &\mathbf{H}(e_0, T_0) \wedge \mathbf{H}(e_1, T_1) : T_0 < T_1 \\ &\rightarrow \mathbf{E}(e_2, T_2) : T_2 < T_1 \vee \mathbf{EN}(e_3, T_3) : T_3 < T_0, 1 \\ &\Rightarrow \mathbf{E}(e_4, T_4) : T_4 < T_0, 2 \end{aligned} \tag{2.2}$$

means that, if e_0 happens before e_1, then either of the two cases below hold:

- e_2 should have happened before e_1 or e_3 should not have happened before e_0,
- e_4 should have happened before e_0;

and the first case has higher priority than (or is preferred to) the second one. Intuitively, a SIC means that, when a set of events matching its body happens, then at least one of the "priority levels" in its conclusion should be satisfied (the higher the priority, the better). In this case, we say that the SIC is *fulfilled*; otherwise, it is *violated*. While priorities have no effect upon the fulfillment status of the society, they could instead be used by a possible computational entity representing the society to guide its members' behaviour towards some preferred state. This can be useful when expectations are accounted for by agents deliberating about future actions. At each point in time there are in general several equally fulfilled sets of expectations. But if some are more preferred to others, an imaginary "social reasoner" which produces expectations based on events could then evaluate and choose which sets of expectations better fit its goals, and transmit only them to the society members. If such members take expectations into account, the whole society could evolve towards preferred states.

The expectations in SIC (2.2) regard events that should have (or have not) happened before the time of the event that raises them: we call this kind of expectations *backward*. Expectations that regard events that are expected to happen (or not to happen) after the event that raises them are named *forward*. We restrict the possible SICs by requiring that they contain only either backward expectations or forward expectations: in the first case, we will call the SIC *backward*, in the second case *forward*. We discuss this restriction in Sect. 7.

3. Specification of the FIPA Contract-Net. FIPA-CNP [1] is a protocol based on FIPA-ACL [14] defined for regulating transactions between entities by negotiation. The protocol flow, represented as an AUML [21] diagram in Fig. 3.1, starts with an Initiator which issues a request for a resource (*cfp*, standing for *call for proposals*) to other Participants. The Participants can reply by proposing a price that satisfies the request (*propose*), or by refusing the request altogether (*refuse*). The Initiator must accept (*accept-proposal*) or reject (*reject-proposal*) the received proposals. A Participant whose proposal has been accepted must, by a given deadline, inform the Initiator that it has provided the resource (by sending an *inform-done* message, or a more informative *inform-result* message) or that it has failed to provide it (*failure*).

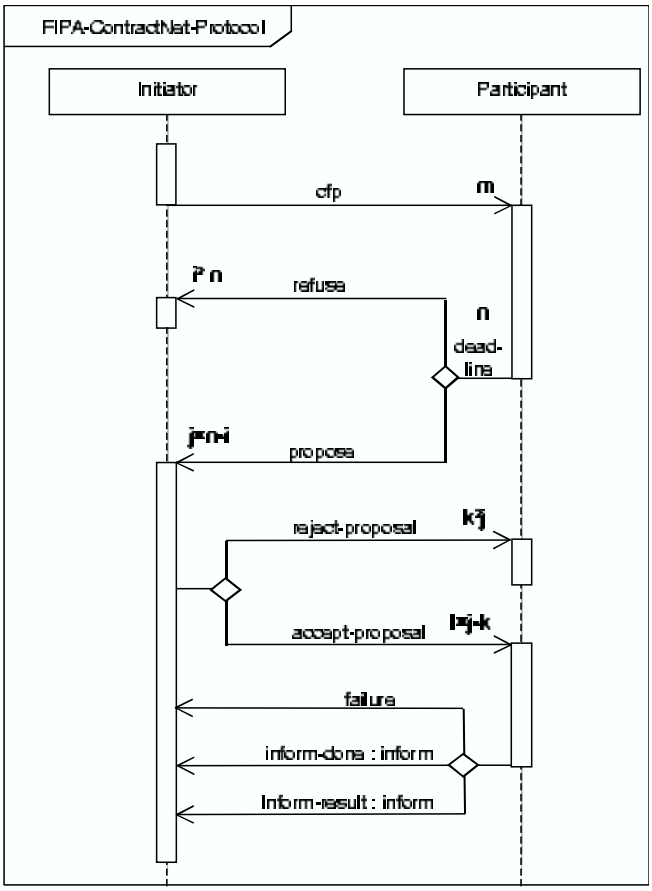


FIG. 3.1. *FIPA-Contract-Net Interaction Protocol (AUML Diagram)*

3.1. Definition by Social Integrity Constraints. The whole set of SICs used to define FIPA-CN is composed of 14 backward SICs and 3 forward SICs. This choice of SICs is obviously not the only possibility. We are currently investigating a general mapping of AUML protocol diagrams and other graphical formalisms to SICs, so as to allow for an automatic translation. Some progress in this sense has been done with the GOSpel graphic language [10] in the health care application domain.

In the SICs in the remainder of this section, I will represent the initiator, P a participant, R the resource, Q the price, D the dialogue identifier, S the explanation of a result, and $T, T_1, \ldots$ the time. We will not use priority levels.

Backward SICs. Backward SICs are used to express that an action is only allowed if some other events have (not) occurred before.

SICs (3.1) and (3.2) state that *propose* and *refuse* are only allowed in reply to a *cfp*.

$$\begin{aligned}&\mathbf{H}(tell(P, I, propose(R, Q), D), T) \rightarrow \\ &\mathbf{E}(tell(I, P, cfp(R), D), T_1) : T_1 < T\end{aligned} \tag{3.1}$$

$$\begin{aligned}&\mathbf{H}(tell(P, I, refuse(R), D), T) \rightarrow \\ &\mathbf{E}(tell(I, P, cfp(R), D), T_1) : T_1 < T\end{aligned} \tag{3.2}$$

SICs (3.3) and (3.4) express mutual exclusion between *propose* and *refuse*.

$$\begin{aligned}&\mathbf{H}(tell(P, I, propose(R, Q), D), T) \rightarrow \\ &\mathbf{EN}(tell(P, I, refuse(R), D), T_1) : T_1 \leq T\end{aligned} \tag{3.3}$$

$$\begin{aligned}&\mathbf{H}(tell(P, I, refuse(R), D), T) \rightarrow \\ &\mathbf{EN}(tell(P, I, propose(R, Q), D), T_1) : T_1 \leq T\end{aligned} \tag{3.4}$$

SICs (3.5) and (3.6) state that *accept-proposal* and *reject-proposal* are only allowed in reply to a *propose*.

$$\begin{aligned}&\mathbf{H}(tell(I, P, accept\text{-}proposal(R, Q), D), T) \rightarrow \\ &\mathbf{E}(tell(P, I, propose(R, Q), D), T_1) : T_1 < T\end{aligned} \tag{3.5}$$

$$\begin{aligned}&\mathbf{H}(tell(I, P, reject\text{-}proposal(R, Q), D), T) \rightarrow \\ &\mathbf{E}(tell(P, I, propose(R, Q), D), T_1) : T_1 < T\end{aligned} \tag{3.6}$$

SICs (3.7) and (3.8) express mutual exclusion between *accept-proposal* and *reject-proposal*.

$$\begin{aligned}&\mathbf{H}(tell(I, P, accept\text{-}proposal(R, Q), D), T) \rightarrow \\ &\mathbf{EN}(tell(I, P, reject\text{-}proposal(R, Q), D), T_1) : T_1 \leq T\end{aligned} \tag{3.7}$$

$$\begin{aligned}&\mathbf{H}(tell(I, P, reject\text{-}proposal(R, Q), D), T) \rightarrow \\ &\mathbf{EN}(tell(I, P, accept\text{-}proposal(R, Q), D), T_1) : T_1 \leq T\end{aligned} \tag{3.8}$$

SICs (3.9), (3.10) and (3.11) say that *inform-done*, *inform-result* and *failure* are only allowed in reply to an *accept-proposal*.

$$\begin{aligned}&\mathbf{H}(tell(P, I, inform\text{-}done(R), D), T) \rightarrow \\ &\mathbf{E}(tell(I, P, accept\text{-}proposal(R, Q), D), T_1) : T_1 < T\end{aligned} \tag{3.9}$$

$$\begin{aligned}&\mathbf{H}(tell(P, I, inform\text{-}result(R, S), D), T) \rightarrow \\ &\mathbf{E}(tell(I, P, accept\text{-}proposal(R, Q), D), T_1) : T_1 < T\end{aligned} \tag{3.10}$$

$$\begin{aligned}&\mathbf{H}(tell(P, I, failure(R), D), T) \rightarrow \\ &\mathbf{E}(tell(I, P, accept\text{-}proposal(R, Q), D), T_1) : T_1 < T\end{aligned} \tag{3.11}$$

SICs (3.12), (3.13) and (3.14) express mutual exclusion between *inform-done*, *inform-result* and *failure*.

$$\begin{aligned}&\mathbf{H}(tell(P, I, inform\text{-}done(R), D), T) \rightarrow \\ &\mathbf{EN}(tell(P, I, failure(R), D), T_1) : T_1 \leq T \wedge \\ &\mathbf{EN}(tell(P, I, inform\text{-}result(R, S), D), T_1) : T_1 \leq T\end{aligned} \tag{3.12}$$

$$\begin{aligned}&\mathbf{H}(tell(P, I, inform\text{-}result(R, S), D), T) \rightarrow \\ &\mathbf{EN}(tell(P, I, failure(R), D), T_1) : T_1 \leq T \wedge \\ &\mathbf{EN}(tell(P, I, inform\text{-}done(R), D), T_1) : T_1 \leq T\end{aligned} \tag{3.13}$$

$$\begin{aligned}&\mathbf{H}(tell(P, I, failure(R), D), T) \rightarrow \\ &\mathbf{EN}(tell(P, I, inform\text{-}done(R), D), T_1) : T_1 \leq T \wedge \\ &\mathbf{EN}(tell(P, I, inform\text{-}result(R, S), D), T_1) : T_1 \leq T\end{aligned} \tag{3.14}$$

Forward SICs. SIC (3.15) says that, after receiving a *cfp*, a Participant is expected to issue a *propose* or a *refuse* by 200 time units.[2]

$$\begin{aligned}&\mathbf{H}(tell(I,P,cfp(R),D),T)\rightarrow\\&\mathbf{E}(tell(P,I,propose(R,Q),D),T_1):T_1<T+200\vee\\&\mathbf{E}(tell(P,I,refuse(R),D),T_2):T_2<T+200\end{aligned}\tag{3.15}$$

SIC (3.16) states that the Initiator is expected to reply to a *propose* with an *accept-proposal* or a *reject-proposal* by 200 clock ticks.

$$\begin{aligned}&\mathbf{H}(tell(P,I,propose(R,Q),D),T)\rightarrow\\&\mathbf{E}(tell(I,P,accept\text{-}proposal(R,Q),D),T_1):T_1<T+200\vee\\&\mathbf{E}(tell(I,P,reject\text{-}proposal(R,Q),D),T_2):T_2<T+200\end{aligned}\tag{3.16}$$

SIC (3.17) states that a Participant is expected to reply to an *accept-proposal* with an *inform-done*, an *inform-result* or a *failure* by 200 clock ticks.

$$\begin{aligned}&\mathbf{H}(tell(I,P,accept\text{-}proposal(R,Q),D),T)\rightarrow\\&\mathbf{E}(tell(P,I,inform\text{-}done(R),D),T_1):T_1<T+200\vee\\&\mathbf{E}(tell(P,I,inform\text{-}result(R,S),D),T_2):T_2<T+200\vee\\&\mathbf{E}(tell(P,I,failure(R),D),T_2):T_2<T+200\end{aligned}\tag{3.17}$$

Note that, in all the three cases, backward SICs make the alternative expectations mutually exclusive.

4. Specification of a semi-open society access protocol. According to [11], societies can be classified into 4 groups, each characterized by a different degree of openness. In the following, we give an example of how our framework can model a semi-open society, i. e., a society that can be joined by an agent executing an access protocol. In this example we imagine that a special *gatekeeper* agent is in charge of receiving joining requests, and it requests agents willing to enter to fill in some registration form.

The access protocol is defined by the following SICs, in which C represents the name of an agent willing to join in:

$$\begin{aligned}&\mathbf{H}(tell(C,gatekeeper,ask(register),D),T)\rightarrow\\&\mathbf{E}(tell(gatekeeper,C,ask(form),D),T_1):T_1<T+10\end{aligned}\tag{4.1}$$

$$\begin{aligned}&\mathbf{H}(tell(C,gatekeeper,ask(register),D),T)\wedge\\&\mathbf{H}(tell(gatekeeper,C,ask(form),D),T_1)\wedge T<T_1\rightarrow\\&\mathbf{E}(tell(C,gatekeeper,send(form,F),D),T_2):T_2<T_1+10\end{aligned}\tag{4.2}$$

$$\begin{aligned}&\mathbf{H}(tell(gatekeeper,C,ask(form),D),T_1)\wedge\\&\mathbf{H}(tell(C,gatekeeper,send(form,F),D),T_2)\wedge T_1<T_2\rightarrow\\&\mathbf{E}(tell(gatekeeper,C,accept(register),D),T_3):T_3<T_2+10\vee\\&\mathbf{E}(tell(gatekeeper,C,reject(register),D),T_3):T_3<T_2+10\end{aligned}\tag{4.3}$$

SIC (4.1) says: if C asks *gatekeeper* to join the society (*register*), then the *gatekeeper* should ask for a registration $form$; SIC (4.2) imposes that, after the first two messages, the agent should provide the $form$; and SIC (4.3) says that, after receiving the form, the *gatekeeper* should either *accept* or *reject* the registration request.

[2] Time unit is an abstract concept, whose instantiation actually depends on the application. A time unit may represent for example a clock tick, or a transaction time.

For the sake of simplicity, in the sequel we assume that member agents do not leave the society. Then, the presence in the history of an event of type:

$$H(tell(gatekeeper, C, accept(register), D), T)$$

can be regarded as C's "formal" act of "membership", and it can be used in SICs as a condition for generating expectations.

For instance, SIC (3.15) from the FIPA-CNP (Sect. 3.1) could be modified as follows to take membership into account:

$$\begin{aligned}&\mathbf{H}(tell(gatekeeper, I, accept(register), D), T_I) \wedge \\ &\mathbf{H}(tell(I, P, cfp(R), D), T) \rightarrow \\ &\mathbf{E}(tell(P, I, propose(R, Q), D), T_1) : T_1 < T + 200 \vee \\ &\mathbf{E}(tell(P, I, refuse(R), D), T_2) : T_2 < T + 200\end{aligned} \tag{4.4}$$

5. Specification of the TCP protocol opening phase. In this section, we present a specification of the open-connection phase of the TCP protocol. We will focus on the well known "three-way handshake" opening, summarized below:

1. a peer A sends to another peer B a *syn* segment;[3]
2. B replies by acknowledging (with an *ack* segment) A's *syn* segment, and by sending a *syn* segment in turn;
3. A acknowledges B's *syn* segment with a *ack* segment, and starts sending data.

The following two integrity constraints describe such a protocol:

$$\begin{aligned}&\mathbf{H}(tell(A, B, tcp(syn, null, NSynA, AckNumber), D), T1) \rightarrow \\ &\mathbf{E}(tell(B, A, tcp(syn, ack, NSynB, NSynAAck), D), T2) : \\ &NSynAAck = NSynA + 1 \wedge T2 > T1.\end{aligned} \tag{5.1}$$

SIC 5.1 says that if A sends to B a *syn* segment, whose sequence number is $NSynA$, then B is expected to send to A an *ack* segment, whose acknowledgment number is $NSynA + 1$, at a later time.

$$\begin{aligned}&\mathbf{H}(tell(A, B, tcp(syn, null, NSynA, AckNumber), D), T1) \\ &\wedge \mathbf{H}(tell(B, A, tcp(syn, ack, NSynB, NSynAAck), D), T2) : \\ &T2 > T1 \wedge NSynAAck = NSynA + 1 \rightarrow \\ &\mathbf{E}(tell(A, B, tcp(null, ack, NSynAAck, NSynBAck), D), T3) : \\ &T3 > T2 \wedge NSynBAck = NSynB + 1.\end{aligned} \tag{5.2}$$

SIC 5.2 says that, if the previous two messages have been exchanged, then A is expected to send to B an *ack* segment acknowledging B's *syn* segment, and with acknowledgement number is $NSynB+1$, where $NSynB$ is the sequence number of B's *syn*.

A third integrity constraint has been added, to verify the interaction between peers with different response time. A faster peer in fact could not wait enough for the acknowledge message, and try to resend a syn message to a slower peer. This situation can lead to several problems in the slower peer, whose queue of the incoming messages could easily get saturated by requests.

$$\begin{aligned}&\mathbf{H}(tell(A, B, tcp(syn, null, NSynA, ANY), D), T1) \\ &\wedge ta(TA) \rightarrow \\ &\mathbf{EN}(tell(A, B, tcp(syn, null, NSynA, ANY), D), T2) : \\ &T2 < T1 \wedge T2 > T1 - TA.\end{aligned} \tag{5.3}$$

SIC 5.3 says that, if A has sent to B a *syn* segment to open a connection, then A is expected not to send another *syn* segment before TA time units, where TA is an application-specific constant, defined by the $ta/1$ predicate.

The above specification has been used to check the interaction between experimental mobile phones and a server.

[3] The term "segment" is used in the TCP specification to indicate bit configuration or streams.

TABLE 6.1
State of an expectation

Type	Verified	Expired	State
E	yes		fulfilled
E	no	no	wait
E	no	yes	violated
EN	yes		violated
EN	no	no	wait
EN	no	yes	fulfilled

6. Verification System. In this section, we describe a prototypical system that we have developed to verify the compliance of the agent behaviour to interaction protocols specified by means of SICs.

The system checks for compliance by accomplishing two main tasks:

1. it fires (*activates*) SICs whose conditions become true as relevant events occurs;
2. it decides whether *activated* SICs are fulfilled or violated.

The system is designed to work during the evolution of the society, so it will only have, at each instant, a partial history available, and it must take into account that new events may happen in the future. For instance, let us consider again the sample expectation in Sect. 2:

$$\mathbf{E}(accept(a_k, a_j, give(M), d_2), T_\mathrm{a}) : M \geq 10, T_\mathrm{a} \leq 15.$$

Let us now suppose that, at time 12, no matching event has yet occurred. So, while this expectation has not been fulfilled, neither it has (yet) been violated: since a matching event could still happen at time 13, 14 or 15. It will actually be violated instead, in case a matching event fails to occur by time 15, because the CLP constraint on the time variable becomes unsatisfiable as of time 16.

More generally, it may not be possible to state whether a SIC is fulfilled or violated at the same time it fires; thus, we identify three possible states for an activated SIC:

- *fulfilled*, if the SIC is fulfilled;
- *violated*, if the SIC is violated;
- *wait*, if the SIC is still neither fulfilled nor violated.

The initial state for an activated SIC is wait; happening events will eventually change its state to fulfilled or violated.

If we process events in the correct order in time, in the case of backward SICs, the transition from a wait state to a fulfilled or violated state is immediate, because expectations in a backward SIC regard events that should have (not) happened in the past and, thus, they can be immediately checked for fulfillment.

6.1. Runtime identification of the state of a SIC. In the following, we explain how the state of a SIC changes at runtime.

The activation of a SIC causes the creation of an instance of its "head" (organized in priority levels, each being a disjunction of conjunction of expectations, as explained in Sect. 2). Afterwards, the state of each single expectation is defined, followed by the state of the priority levels, and finally by the state of the SIC.

State of an expectation. An expectation is called "verified" if there exists a matching event in the society history. The state of a verified positive expectation is *fulfilled*; the state of a verified negative expectation is *violated*.

An expectation is called "expired" if CLP constraints over its time variable cannot be any longer satisfied (typically, this is the case with constraints representing deadlines which have expired). The state of an expired and not verified expectation is *violated* if the expectation is positive and *fulfilled* if the expectation is negative; the state of a not expired and not verified expectation is instead *wait*.

Table 6.1 summarises all these cases.

State of a conjunction of expectations. The state of a conjunction of expectations is defined by the following rules:

1. if the state of at least one expectation in the conjunction is *violated*, then the state of the conjunction is *violated*;
2. if the state of all expectations in the conjunction is *fulfilled*, the state of the conjunction is *fulfilled*;
3. otherwise, the state is *wait*.

State of a priority level. A priority level is a disjunction of conjunctions of expectations. The state of a priority level is then defined by the following rules:

1. if the state of at least one of the disjuncts is *fulfilled*, then the state of the priority level is *fulfilled*;
2. if the state of all of the disjuncts is *violated*, then the state of the priority level is *violated*;
3. otherwise, the state is *wait*.

State of a SIC. If all the priority levels of a SIC are violated, then the SIC is *violated*; otherwise, the state of the highest non-violated priority level of the SIC defines the state of the SIC.

6.2. Verification of Compliance. As shown in Sect. 3.1 in relation to the FIPA CNP, backward SICs can express that events are only allowed if some other events have (not) happened before; since their state can be immediately resolved to *fulfilled* or *violated*, backward SICs can be used to verify that an event is allowed as soon as it occurs. In designing our system, we made a choice to ignore the events that are not allowed. However, the system captures the violation: in a richer social model, we can imagine some authority to react to the violation.

The set of forward SICs associated with a legal action is then used to generate expectations about the future events in the society (i. e., the heads of associated forward SICs will be checked for fulfillment).

In order to verify the fulfillment of SICs, we have defined two different phases: the *Event Driven* phase and the *Clock Driven* phase.

Event-driven phase. An event-driven phase starts each time a new event occurs. The system activates all backward SICs associated with the event; if they are all fulfilled, then the event is recognized to be allowed and thus marked as "legal" and added to the history of the interaction. If some of the backward SICs are violated, then the event is marked as "illegal", since it is not allowed, and it is not recorded in the history of the society.

If the event is marked legal, the system processes the new updated history by activating the forward SICs associated with the new event. Forward (activated) SICs define the expected future behaviour of the society, and they will be checked for fulfillment.

Clock-driven phase. The clock-driven phase starts whenever a special event called "clock," or "current time," is registered by the society. The system processes the set of activated forward SICs identifying the state of each one. If the state of a SIC is fulfilled, the SIC is removed from the list of pending (waiting) SICs. If the state of a SIC is violated, the SIC is removed but a violation is raised. If the state is wait, the SIC is kept pending until the next clock-driven phase or the next event-driven phase. Note that the time associated to events and the "current time" event which fires a clock-driven phase must synchronize.

6.3. Implementation. The verification system has been implemented on top of SICStus Prolog's *Constraint Handling Rules* (*CHR*) library [22].

CHR[16] are essentially a committed-choice language consisting of guarded rules that rewrite constraints in a store into simpler ones until they are solved. *CHR* define both *simplification* (replacing constraints by simpler constraints while preserving logical equivalence) and *propagation* (adding new, logically redundant but computationally useful, constraints) over user-defined constraints.

6.3.1. Activation of SICs. Each event happened in the system is represented by the *CHR* constraint `h/2`, where the arguments are a Prolog ground term representing the happened event and an integer number representing the time.

Positive (resp. negative) expectations are represented by the Prolog term `e` (resp. `en`). Its arguments are: a Prolog term describing the event expected to happen (resp. not to happen), the time (typically non ground), and a list of CLP constraints over the variables in the description.

A *PriorityLevel* is represented by the Prolog term `pr`, whose arguments are the list of alternative *HeadDisjuncts* of the priority level and the integer number representing the priority (the lower the number, the higher the priority). Priority levels generated by a SIC are collected as the list argument of a `plist` term.

The argument of the *CHR* constraint `le/1` is the list of all activated `plists` (one for each activated SIC).

Each SIC is represented by a *simpagation CHR*. In general, simpagation rules have the form

$$H_1, \ldots, H_l \backslash H_{l+1}, \ldots, H_i \Leftrightarrow G_1, \ldots, G_j | B_1, \ldots, B_k \tag{6.1}$$

where $l > 0$, $i > l$, $j \geq 0$, $k \geq 0$ and where the multi-head $H_1, \ldots, H_i$ is a nonempty sequence of *CHR* constraints, the guard $G_1, \ldots, G_j$ is a sequence of built-in constraints, and the body $B_1, \ldots, B_k$ is a sequence of built-in and *CHR* constraints. Operationally, when the constraints in the head are in the constraint store and

the guard is true, $H_1, \ldots, H_l$ remain in the store, and $H_{l+1}, \ldots, H_i$ are substituted by $B_1, \ldots, B_k$. For instance, the following *CHR* implements SIC (2.2):

```
h(event0,T0), h(event1,T1) \ le(LExp) <=> T0<T1 &
append(LExp,
 [plist([
    pr([
      and([ e(event2,T2,[min(T2,T1)]) ]),
      and([ en(event3,T3,[min(T3,T0)]) ])
    ],1),
    pr([
      and([ e(event4,T4,[min(T4,T0)]) ])
    ],2)
  ],id1)], LExp1)
| le(LExp1).
```

If `event0` and `event1` have occurred and are part of the "history," the two *CHR* constraints `h(event0,T)` and `h(event1,T1)` are in the constraint store; if the guard `T<T1` is true, then the rule is activated. The store (the `LExp` list) of the heads of activated SICs is updated appending a new *plist()*, which contains the list of priority levels (two in this example) in the head of the SIC. The *CHR* constraint `le/1`, which contained the old `LExp` before the activation of the rule, is removed by simpagation and replaced by the same constraint with the new list `LExp1` as argument.

Note that two different symbols are used to represent the CLP constraint $<$: `<` if its arguments are the times of two happened events[4], and `min` if they are instead the times of two expectations.

The translation of a SIC into a simpagation *CHR* is rather straightforward, which makes it easy to implement new protocols.

As further examples, we report below the *CHR* implementation of SIC (3.1) and SIC (3.15):

```
h(tell(P,I,propose(R,Q),D),T) \
le(LExp) <=>
true &
append(LExp,
[plist([
     pr([
         and([
     e(tell(I,P,cfp(R),D),T1,[min(T1,T)])
             ])
        ],1)
])], LExp1) | le(LExp1).
```

```
h(tell(I,P,cfp(R),D),T) \
 le(LEv,LExp) <=>
 Td is T+200 &
 append(LExp,
 [plist([
     pr([
         and([
     e(tell(P,I,propose(R,Q),D),T1,[min(T1,Td)])
             ]),
         and([
     e(tell(P,I,refuse(R),D),T2,[min(T2,Td)])
             ])
        ],1)
   ])],
LExp1) | le(LExp1).
```

[4] In this case, the times are certainly ground and the Prolog predefined predicate can be applied to them.

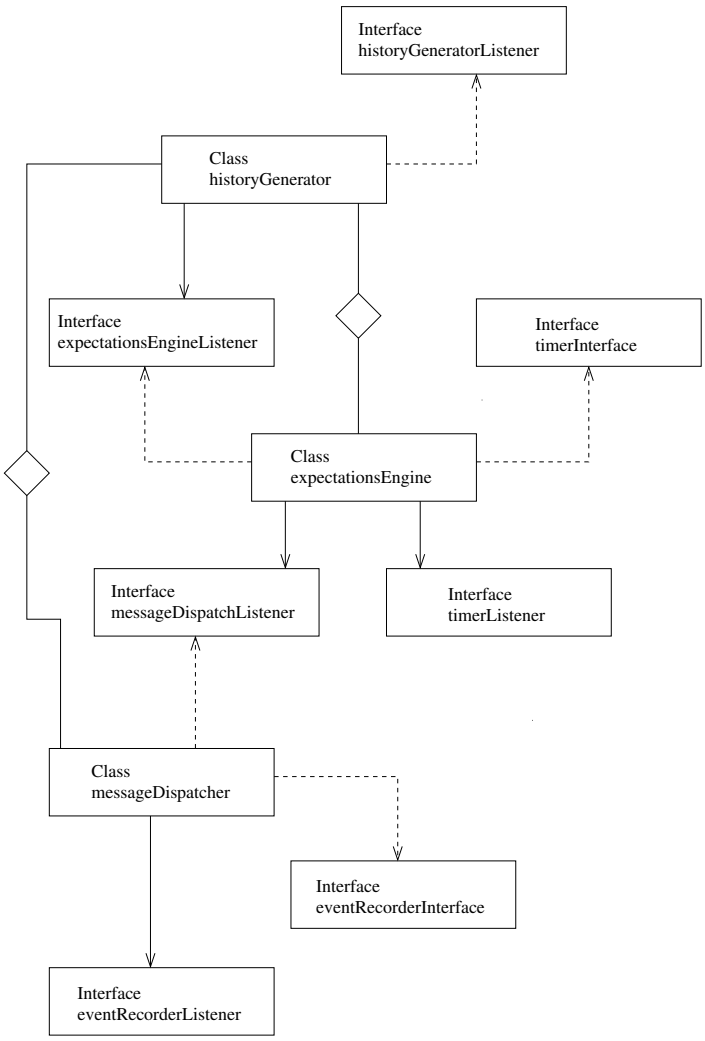


FIG. 6.1. *UML diagram*

6.3.2. Identification of the state of SICs. The identification of the state of a SIC is coded in standard Prolog. The system performs all the steps described in Sect. 6.1. It analyses all its stored `plists`, thus implementing the event-driven and clock-driven phases described above.

6.3.3. Interface to the verification system. In order to use the system in concrete case studies, a Java package (using the SICStus Prolog's Jasper library [22]) has been implemented. This package has been developed to be used as a Java wrapper for the verification system.

The UML diagram of the system is represented in Fig. 6.1. To use the system the user must create a *historyGenerator* object giving as parameter the path to a (compiled) Prolog file containing the protocol definition expressed by SICs. The Java system implements the Event Driven phase receiving messages from the *eventRecorderListener* interface and the clock-driven phase receiving "current time" events from the *timerListener* interface. The rest of the system implements the Java-Prolog interface.

7. Discussion and related work. The syntax of Social Integrity Constraints proposed in this paper is a modified version of that proposed in [2] and in [5]. The modifications have been made in order to tackle both expressiveness and implementation issues. Specifically:

- we added priority levels to SICs (see Sect. 2). This allows for a more flexible specification of protocols, enabling the protocol designer to devise alternative protocol flows while being able to specify preferences among them;

- we imposed the restriction of having only either backward or forward expectation in a SIC (see Sect. 2). While this improves efficiency, on the downside it prevents from writing SICs such as

$$\begin{aligned} &\mathbf{H}(a, T_a) \\ \rightarrow &\mathbf{E}(b, T_b) : T_b < T_a, 1 \\ \Rightarrow &\mathbf{E}(c, T_c) : T_c \leq T_a + \tau, 2 \end{aligned} \tag{7.1}$$

  which one might want to use to express that an event (b) that does not fulfill a backward expectation can, with lower priority, still be allowed, provided that certain "backup" event (c) occur at some point in the future. However, in our experience, SICs such as (7.1) are generally not necessary to express protocols of common use.

In [4] we have defined an abductive semantics for SICs, in the context of agent societies, and a more general framework, in which the verification procedure is performed by an abductive proof procedure [6], whose implementation has been integrated into a software component [3], interfaced to several multi-agent platforms such as Jade [8], PROSOCS [9], and tuProlog [12]. Other authors have proposed alternative approaches to the specification and in some cases animation of interaction among agents. Notably, in [7], Artikis et al. present a theoretical framework for providing executable specifications of particular kinds of multi-agent systems, called open computational societies, and they present a formal framework for specifying and animating systems where the behaviour of the members and their interactions cannot be predicted in advance, and for reasoning about and verifying the properties of such systems. A noteworthy difference with [7] is that we do not explicitly represent the institutional power of the members and the concept of valid action. Permitted are all social events that do not determine a violation, i. e., all events that are not explicitly forbidden are allowed.

In [24], Yolum and Singh apply a variant of Event Calculus [19] to commitment-based protocol specification. The semantics of messages (i. e., their effect on commitments) is described by a set of *operations* whose semantics, in turn, is described by *predicates* on *events* and *fluents*; in addition, commitments can evolve, independently of communicative acts, in relation to *events* and *fluents* as prescribed by a set of *postulates*. Such a way of specifying protocols is more flexible than traditional approaches based on action sequences in that it prescribes no initial and final states or transitions explicitly, but it only restricts the agent interaction in that, at the end of a protocol run, no commitment must be pending. Agents with reasoning capabilities can themselves plan an execution path suitable for their purposes (which, in that work, is implemented by an abductive event calculus planner). Our notion of expectation is more general than that of commitment found in [24] or in other commitment-based works, such as [15]: it represents the necessity of a (past or future) event, and is not bound to have a debtor or a creditor, or to be brought about by an agent.

8. Conclusions. We have presented a framework for the specification and runtime verification of compliance of agent interaction to protocols. The specification at a social level of interaction protocols constrains the agent observable behaviour from the outside, rather than its internal state or structure. This is a characteristic of social approaches to agent protocol specification, and it is particularly suited for usage in open agent societies. Protocol specifications use a computational logic-based formalism called social integrity constraints. The system's Java-Prolog-*CHR*based implementation has been tested on different types of protocols [23]. In this article, we have demonstrated the usage of SICs in three cases: the FIPA CNP, taken from the agent literature, a made up protocol for joining semi-open societies, and the well known three-way handshake phase of the TCP IP protocol for connection establishment. The verification system, implemented in Prolog and *CHR*, can be used as a module in a Java-based system, thanks to the Java-Prolog interface of SICStus Prolog. The modular structure of the system makes it (hopefully) easy to adapt it to new applications.

9. Acknowledgments. This research has been partially supported by the National MIUR PRIN 2005 projects No 2005-011293, *Specification and verification of agent interaction protocols*, [5], and No 2005-015491, *Vincoli e preferenze come formalismo unificante per l'analisi di sistemi informatici e la soluzione di problemi reali*, [6] and by the National FIRB project *TOCAI.IT*[7].

[5] http://www.ricercaitaliana.it/prin/dettaglio_completo_prin_en-2005011293.htm
[6] http://www.sci.unich.it/~bista/projects/prin2006/
[7] http://www.dis.uniroma1.it/~tocai/

REFERENCES

[1] *FIPA Contract Net Interaction Protocol*, Tech. Report SC00029H, Foundation for Intelligent Physical Agents, 2002. Available at `http://www.fipa.org`

[2] M. Alberti, A. Ciampolini, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni, *A social ACL semantics by deontic constraints*, in Multi-Agent Systems and Applications III. Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003, V. Mařík, J. Müller, and M. Pěchouček, eds., vol. 2691 of Lecture Notes in Artificial Intelligence, Prague, Czech Republic, June 16–18 2003, Springer-Verlag, pp. 204–213.

[3] M. Alberti, M. Gavanelli, E. Lamma, F. Chesani, P. Mello, and P. Torroni, *Compliance verification of agent interaction: a logic-based software tool.*, Applied Artificial Intelligence, 20 (2006), pp. 133–157.

[4] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni, *An Abductive Interpretation for Open Societies*, in AI*IA 2003: Advances in Artificial Intelligence, Proceedings of the 8th Congress of the Italian Association for Artificial Intelligence, Pisa, A. Cappelli and F. Turini, eds., vol. 2829 of Lecture Notes in Artificial Intelligence, Springer-Verlag, Sept. 23–26 2003, pp. 287–299.

[5] ———, *Specification and Verification of Agent Interactions using Social Integrity Constraints*, Electronic Notes in Theoretical Computer Science, 85 (2003).

[6] ———, *The $\mathcal{SCIFF}$ abductive proof-procedure*, in Proceedings of the 9th National Congress on Artificial Intelligence, AI*IA 2005, vol. 3673 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 2005, pp. 135–147.

[7] A. Artikis, J. Pitt, and M. Sergot, *Animated specifications of computational societies*, in Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part III, C. Castelfranchi and W. Lewis Johnson, eds., Bologna, Italy, July 15–19 2002, ACM Press, pp. 1053–1061.

[8] F. Bellifemine, F. Bergenti, G. Caire, and A. Poggi, *Jade - a java agent development framework*, in Multi-Agent Programming: Languages, Platforms and Applications, R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, eds., vol. 15 of Multiagent Systems, Artificial Societies, and Simulated Organizations, Springer-Verlag, 2005, pp. 125–147.

[9] A. Bracciali, U. Endriss, N. Demetriou, A. C. Kakas, W. Lu, and K. Stathis, *Crafting the mind of prosocs agents*, Applied Artificial Intelligence, 20 (2006), pp. 105–131.

[10] F. Chesani, A. Ciampolini, P. Mello, M. Montali, and S. Storari, *Testing guidelines conformance by translating a graphical language to computational logic*, in ECAI 2006 Workshop on AI techniques in healthcare:evidence based guidelines and protocols, Riva del Garda, Italy, August 2006. `http://www.openclinical.org/cgp2006_2.html`

[11] P. Davidsson, *Categories of artificial societies*, in Engineering Societies in the Agents World II, A. Omicini, P. Petta, and R. Tolksdorf, eds., vol. 2203 of Lecture Notes in Artificial Intelligence, Springer-Verlag, Dec. 2001, pp. 1–9. 2nd International Workshop (ESAW'01), Prague, Czech Republic, July 7, 2001, Revised Papers.

[12] E. Denti, A. Omicini, and A. Ricci, *Multi-paradigm Java-Prolog integration in* tu*Prolog*, Science of Computer Programming, 57 (2005), pp. 217–250.

[13] U. Endriss, N. Maudet, F. Sadri, and F. Toni, *Protocol conformance for logic-based agents*, in Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico (IJCAI-03), G. Gottlob and T. Walsh, eds., Morgan Kaufmann Publishers, Aug. 2003.

[14] *FIPA: Foundation for Intelligent Physical Agents.* Home Page: `http://www.fipa.org/`

[15] N. Fornara and M. Colombetti, *Operational specification of a commitment-based agent communication language*, in Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II, C. Castelfranchi and W. Lewis Johnson, eds., Bologna, Italy, July 15–19 2002, ACM Press, pp. 535–542.

[16] T. Frühwirth, *Theory and practice of constraint handling rules*, Journal of Logic Programming, 37 (1998), pp. 95–138.

[17] J. Jaffar and M. Maher, *Constraint logic programming: a survey*, Journal of Logic Programming, 19-20 (1994), pp. 503–582.

[18] A. C. Kakas, R. A. Kowalski, and F. Toni, *The role of abduction in logic programming*, in Handbook of Logic in Artificial Intelligence and Logic Programming, D. M. Gabbay, C. J. Hogger, and J. A. Robinson, eds., vol. 5, Oxford University Press, 1998, pp. 235–324.

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

[20] J. W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, 2nd extended ed., 1987.

[21] J. Muller and J. Odell, *Agent UML: A formalism for specifying multiagent software systems*, International Journal of Software Engineering and Knowledge Engineering, 11(3) (2001), pp. 207–230.

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

[23] *The SOCS protocol repository*, 2005. Available at `http://edu59.deis.unibo.it:8079/SOCSProtocolsRepository/jsp/index.jsp`

[24] P. Yolum and M. Singh, *Flexible protocol specification and execution: applying event calculus planning using commitments*, in Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II, C. Castelfranchi and W. Lewis Johnson, eds., Bologna, Italy, July 15–19 2002, ACM Press, pp. 527–534.

Edited by: Marcin Paprzycki, Niranjan Suri
Received: October 1, 2006
Accepted: December 10, 2006

Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 15–28. http://www.scpe.org
ISSN 1895-1767

INCORPORATING PLANNING INTO BDI SYSTEMS

FELIPE RECH MENEGUZZI, AVELINO FRANCISCO ZORZO,
MICHAEL DA COSTA MÓRA AND MICHAEL LUCK

Abstract. Many architectures of autonomous agent have been proposed throughout AI research. The most common architectures, BDI, are procedural in that they do no planning, seriously curtailing an agent's ability to cope with unforeseen events. In this paper, we explore the relationship between propositional planning systems and the process of means-ends reasoning used by BDI agents and define a mapping from BDI mental states to propositional planning problems and from propositional plans back to mental states. In order to test the viability of such a mapping, we have implemented it in an extension of a BDI agent model through the use of Graphplan as the propositional planning algorithm. The implemented prototype was applied to model a case study of an agent controlled production cell.

Key words. Propositional Planning, Agent Models and Architectures, BDI, X-BDI

1. Introduction. Development of autonomous rational agents has been one of the main drivers of artificial intelligence research for some time [37]. Initial efforts focused on disembodied means-ends reasoning with the development of problem-solving systems and generic planning systems, such as STRIPS [15], later evolving into the idea of embodied problem solving entities (*i.e.* agents) [37]. In this line of research, one of the most widely studied models of autonomous agents has been that supported by the mental states of beliefs, desires and intentions [7], or the BDI model. While efforts towards defining BDI architectures have been sustained and significant, resulting in both theoretical [34] and practical architectures [14], they have also led to a disconnect between them.

In particular, theories of autonomous BDI agents often rely on logic models that assume infinite computational power, while architectures defined for runtime efficiency have curtailed an agent's autonomy by forcing the agent to rely on a pre-compiled plan library. Although simple selection of plans from a plan library is computationally efficient, at compile time an agent is bound to the plans provided by the designer, limiting an agent's ability to cope with situations not foreseen at design time. Moreover, even if a designer is able to define plans for every conceivable situation in which an agent finds itself, such a description is likely to be very extensive, offsetting some of the efficiency benefits from the plan library approach. The absence of planning capabilities thus seriously curtails the abilities of autonomous agents. In consequence, we argue that planning is an important capability of any autonomous agent architecture in order to allow the agent to cope at runtime with unforeseen situations.

Though the efficiency of planning algorithms has been a major obstacle to their deployment in time-critical applications, many advances have been achieved in planning [43], and developments are ongoing [2]. Considering that planning is an enabler of agent flexibility, and that there have been significant advances in planning techniques, it is valuable and important for autonomous agent architectures to employ planning to allow an agent to cope with situations that the designer was not able to foresee. This article describes and demonstrates one such architecture, which integrates propositional planning with BDI, allowing agents to take advantage of the practical reasoning capabilities (*i.e.* selecting and prioritising goals) of the BDI model, and replacing the BDI means-ends reasoning (*i.e.* selecting a course of action to achieve goals) with the flexibility of generic planning. Our approach is underpinned by a mapping among BDI mental states and propositional planning formalisms, allowing any algorithm based on a similar formalism to be used as a means-ends reasoning process for a BDI agent. In order to demonstrate the viability of such an approach we take a specific BDI agent model, namely the X-BDI model [27], and modify it to use propositional planning algorithms to perform means-ends reasoning [30].

The paper is organised as follows: Section 2 contains an overview of the related work and main concepts used throughout this paper; Section 3 describes X-BDI and the extensions that allow it to use an external planning algorithm; Section 4 contains a case study used to demonstrate the implemented prototype; finally, Section 5 contains concluding remarks about the results obtained in this work.

2. Agents and Planners. In this section we review background work on agents and planning systems, and conclude with a discussion of the integration of these technologies in an agent architecture, laying the groundwork for the remainder of this article. Section 2.1 provides an overview of computer agents and the BDI model, used in the agent architecture described later in this article; Section 2.2 introduces generic planning

algorithms and problem representation; Section 2.3 describes the particular planning algorithm used in the prototype described in Section 3; finally, we discuss how these technologies can be pieced together in order to address some of their individual limitations.

2.1. Agents. The growing complexity of computer systems has led to the development of increasingly more advanced abstractions for their representation. An abstraction of growing popularity for representing parts of complex computer systems is the notion of *computer agents* [13], so far as to be proposed as an alternative to the Turing Machine as an abstraction for the notion of computation [19, 42]. Although there is a variety of definitions for computer agents, most researchers agree with Jennings' definition of an agent as *encapsulated* computer system, *situated* in some environment, and capable of *flexible*, *autonomous* action in that environment in order to meet its *design objectives* [19].

In the context of multi-agent systems research, one of the most widely known and studied models of deliberative agents uses *beliefs*, *desires* and *intentions* (BDI) as abstractions for the description of a system's behaviour. The BDI model originated from a philosophical model of human practical reasoning [6], later formalised [11] and improved towards a more complete computational theory [34, 44]. Though other approaches to the design of autonomous agents have been proposed [16], the BDI model or variations of it are used in many new architectures of autonomous agents [13, 31, 4, 40]. More specifically, the components that characterise the BDI model can be briefly described as follows [28]:

- **beliefs** represent an agent's expectation regarding the current world state or the possibility that a given course of action will lead to a given world state;
- **desires** represent a set of possibly inconsistent preferences an agent has regarding a set of world states; and
- **intentions** represent an agent's commitment regarding a given course of action, constraining the consideration of new objectives.

The operation of a generic BDI interpreter can be seen as a process that starts with an agent considering its sensor input and updating its belief base. With this updated belief base, a set of goals from the agent's desires is then selected, and the agent commits itself to achieving these goals. In turn, plans are selected as the means to achieve the goals through intentions which represent the commitment. Finally, these intentions are carried out through concrete actions contained in the instantiated plans (or intentions). This process is illustrated in the activity diagram of Figure 2.1, which shows the components of an agent that are used in each of the main processes of BDI reasoning, namely: obtaining sensor input and updating beliefs; selecting a goal from among the desires; and adopting intentions to carry out the actions required to achieve the selected goal.

This last process of selecting and adopting intentions to achieve a goal is one of the most important processes of the BDI model, since it affects not only the actions an agent chooses, but also the selection of goals, as an agent must drop goals deemed impossible. This problem of determining whether an agent is capable of satisfying its objectives through some sequence of actions given an environment and a set of objectives is sometimes characterised as the *agent design problem* [45]. The most widely known BDI agent implementations [18, 33, 14] bypass this problem through the use of plan libraries in which the courses of action for every possible objective an agent might have are stored as encapsulated procedures. Agents using these approaches are said to pursue *procedural goals*. However, the *theories* commonly used to underpin the creation of new plans of action at runtime assume an agent with unlimited resources, thus making their actual implementation impossible [37, 34]. When an agent selects target world-states and then uses some process at runtime to determine its course of action, it is said to pursue *declarative goals*. Recent efforts seek to deal with this problem in various ways, for instance by defining alternate proof systems [27, 31] or using model checking in order to validate the agent's plan library [5]. An alternative approach to solving the problem is the use of planning algorithms to perform means-ends reasoning at runtime [37, 26, 47].

2.2. Planning Algorithms. Means-ends reasoning is a fundamental component of any *rational* agent [6] and is useful in the resolution of problems in a number of different areas, such as scheduling [38], military strategy [39], and multi-agent coordination [12]. Indeed, the development of planning algorithms has been one of the main goals of AI research [35]. In more detail, a planning problem is generically defined by three components [43]:

- a formal description of the start state;
- a formal description of the intended goals; and
- a formal description of the actions that may be performed.

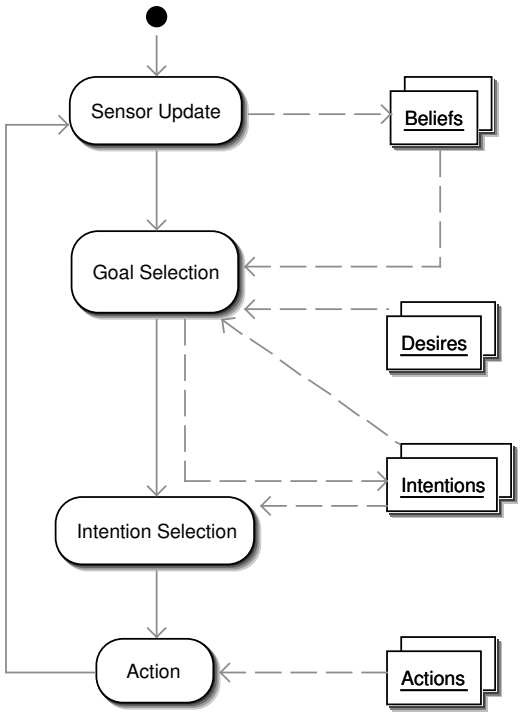


FIG. 2.1. *Activities of a generic BDI interpreter.*

A planning system takes these components and generates a set of actions ordered by some relation which, when applied to the world in which the initial state description is true, makes the goals' description true. Despite the high complexity proven for the general case of planning problems[1], recent advances in planning research have led to the creation of planning algorithms that perform significantly better than previous approaches to solving various problem classes [43, 2]. These new algorithms make use of two main techniques, either combined or separately:

- expansion and search in a planning graph [3]; and
- compilation of the planning problem into a logical formula to be tested for satisfiability (SAT) [20].

One such planning algorithm is Graphplan, which we consider in more detail below.

2.3. Graphplan. Graphplan [3] is a planning algorithm based on the first of these techniques, expansion and search in a graph. It is considered to be one of the most efficient planning algorithms created recently [43, 38, 17], having been refined into a series of other algorithms, such as IPP (Interference Progression Planner) [22] and STAN (STate ANalysis) [24]. The efficiency of Graphplan was empirically demonstrated through the very significant results obtained by instances of Graphplan in the planning competitions of the AIPS (International Conference on AI Planning and Scheduling) [21, 25].

Planning in Graphplan is based on the concept of a graph data structure called the *planning graph*, in which information regarding the planning problem is stored in such a way that the search for a solution can be accelerated. Planning graph construction is efficient, having polynomial complexity in graph size and construction time with regard to problem size [3]. A plan in the planning graph is essentially a flow, in the sense of a network flow, and the search for a solution to the planning problem is performed by the planner using data stored in the graph to speed up the process. The basic Graphplan algorithm (*i.e.* without the optimisations proposed by other researchers [21, 25]) is divided into *graph expansion* and *solution extraction*, which take place alternately until either a solution is found or the algorithm can prove that no solution exists. The way these two parts of Graphplan are used throughout planning is summarised in the activity diagram of Figure 2.2, and explained below.

Since a plan is composed of temporally ordered actions and, in between these actions there are world states, graph levels are divided into alternating proposition and action levels, making it a directed and levelled graph,

[1]Planning is known to be undecidable [10] and planning problems, in the general case, have PSPACE complexity [9].

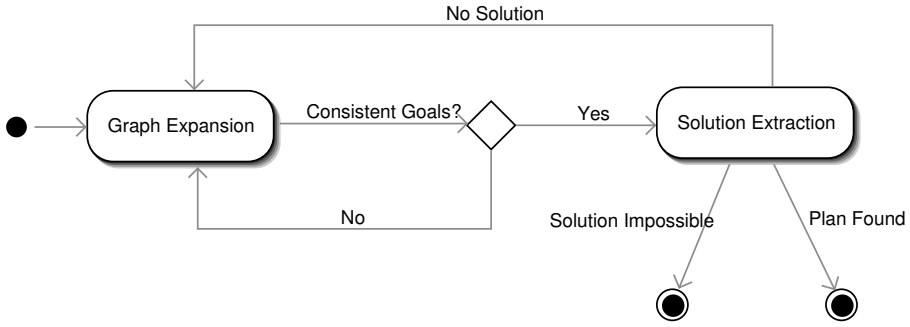


Fig. 2.2. *Graphplan algorithm overview.*

as shown in Figure 2.3. Proposition levels are composed of proposition nodes labelled with propositions, and connected to the actions in the subsequent action level through pre-condition arcs. Here, action nodes are labelled with operators and are connected to the nodes in the subsequent proposition nodes by effect arcs.

Every proposition level denotes literals that are possibly true at a given moment, so that the first proposition level represents the literals that are possibly true at time 1, the next proposition level represents the literals that are possibly true at time 2 and so forth. Similarly, action levels denote operators that can be executed at a given moment in time in such a way that the first action level represents the operators that may be executed at time 1, the second action level represents the operators that may be executed at time 2 and so forth. The graph also contains mutual exclusion relations (*mutex*) between nodes (at the same graph level) so that they cannot be simultaneously present at the same graph level for the same solution. This gives them a fundamental role in algorithm efficiency, as they allow the search for a solution to completely ignore a large number of possible flows in the graph.

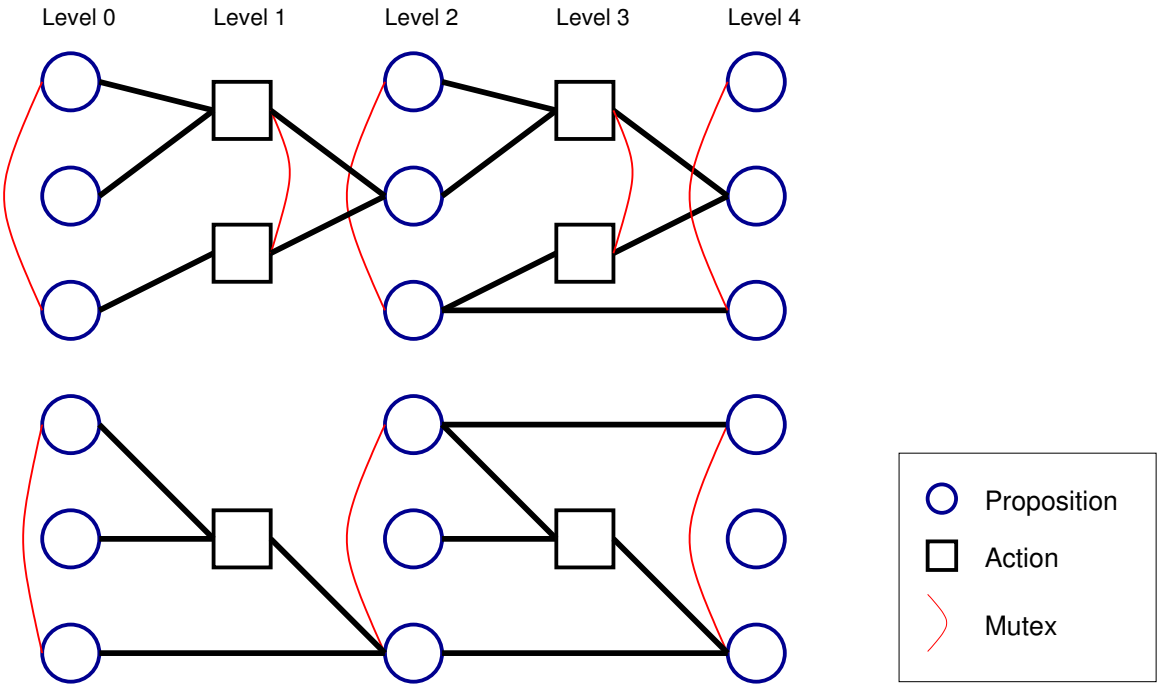


Fig. 2.3. *A planning graph example.*

After graph expansion, the graph is analysed by the solution extraction part of the algorithm, which uses a backward chaining strategy to traverse the graph, level by level, trying to find a flow starting from the goals and leading to the initial conditions. An important optimising factor in this phase is never to search for a solution

unless all the *goal* propositions are present and consistent, since *they cannot be mutually exclusive at the last graph level*. Fundamental to Graphplan is its assurance that, whenever a plan for the proposed problem exists, the algorithm will find it, otherwise the algorithm will determine that the proposed problem is unsolvable [3].

2.4. Discussion. When one considers how BDI reasoning operates, it is straightforward to perceive that propositional planning can be used as a means-ends reasoning component. From a representational point of view, BDI mental states can be converted to planning problems without complication: beliefs translate into an initial state specification, actions and capabilities translate into operator specifications and selected goals translate into a goal state specification. At this simple level, the delegation of means-ends reasoning to an external planning process can improve the runtime efficiency of existing BDI interpreters by leveraging advances in planning algorithms research.

3. Introducing procedural planning into X-BDI.

3.1. Introduction. Given the shortcomings of traditional BDI architectures in terms of runtime flexibility, and the performance problems of alternative architectures, we define an extended version of the X-BDI agent model [27], modified to accommodate the use of an external planning component. Here, we focus on STRIPS-like (STanford Research Institute Problem Solver) formalisms [15]. Our formalism is based on the one introduced by Nebel [30], and, according to the author, is a $\mathcal{S}_{\mathcal{IL}}$ formalism, *i.e.* the basic STRIPS plus the possibility to use incomplete specifications and literals in the description of world states. It is important to point out that the formalism defined by Nebel [30] is more general, but since we do not aim to provide a detailed study of planning formalisms, we use a simpler version. In particular, we use a propositional logical language with variables only in the specification of operators, and with operators not being allowed to have conditional effects. In Nebel's description of the the STRIPS formalism, one can notice that it deals only with atoms. Nevertheless, within this paper more expressivity is desirable, in particular, the possibility to use first order ground literals. It is possible to avoid these limitations through the use of syntactic transformations so that planners can operate over first order ground literals. The main contribution of our work lies in the efficiency improvement of a *declarative* agent architecture. The fact that this type of agent architecture has traditionally been notoriously inefficient highlights the relevance of this efficiency gain.

3.2. X-BDI. An X-BDI agent has the traditional components of a BDI agent, *i.e.* a set of *beliefs*, *desires* and *intentions*. The agent model was originally defined in terms of the *Extended Logic Programming with explicit negation* (ELP) formalism created by Alferes and Pereira [1], which includes a revision procedure responsible for maintaining logic consistency. We do not provide a description of the formalism here, though we assume the presence of its revision procedure in our description of X-BDI. Given its extended logic definition, X-BDI also has a set of time axioms defined through a variation of the *Event Calculus* [27, 23].

The set of beliefs is simply a formalisation of facts in ELP, individualised for a specific agent. From the agent's point of view, it is assumed that its beliefs are not always consistent, and whenever an event makes the beliefs inconsistent, they must be revised. The details of this process are unimportant in the understanding of the overall agent architecture, but can be found in [1]. The belief revision process in X-BDI is the result of the program revision process performed in ELP.

Every desire in an X-BDI agent is conditioned to the body of a logic rule, which is a conjunction of literals called *Body*. Thus, *Body* specifies the pre-conditions that must be satisfied in order for an agent to desire a property. When *Body* is an empty conjunction, some property P is unconditionally desired. Desires may be temporally situated, *i.e.* can be desired at a specific moment, or whenever their pre-conditions are valid. Moreover, a desire specification contains a priority value used in the formation of an order relation among desire sets.

There are two possible types of intentions: *primary intentions*, which refer to the intended properties, and *relative intentions*, which refer to actions able to bring about these properties. An agent may not intend something in the past or that is already true, and intentions must in principle be possible, *i.e.* there must be at least one plan available whose result is a world state where the intended property is true.

Now, we diverge from the original X-BDI architecture in several respects. First, the original reasoning process verified the possibility of a property through the abduction of an event calculus theory to validate the property. In brief, the logic representation of desires in the original X-BDI included clauses specifically marked for revision in such a way that sequences of actions (whose preconditions and effects were described in event calculus) could be found true in the process of revising these clauses. This abduction process was necessary

for the implementation of X-BDI planning framework in extended logic, but the implementation of the logic interpreter was notably inefficient for abductive reasoning. In this work, the planning process is abstracted out from the operational definition of X-BDI, allowing any planning component that satisfies the conditions of Section 2.2 to be invoked by the agent. Thus, the notion of possibility of a desire is associated with the existence of a plan to satisfy it.

The reasoning process performed by X-BDI begins with the selection of *eligible desires*, which represent unsatisfied desires whose pre-conditions are valid, though the elements of this set of desires are not necessarily consistent among themselves. A set of eligible desires that are both consistent and possible is then selected as *candidate desires*, to which the agent commits itself to achieving by adopting them as *primary intentions*. In order to achieve the primary intentions, the planning process generates a sequence of temporally ordered actions that constitute the *relative intentions*. This process is summarised in Figure 3.1.

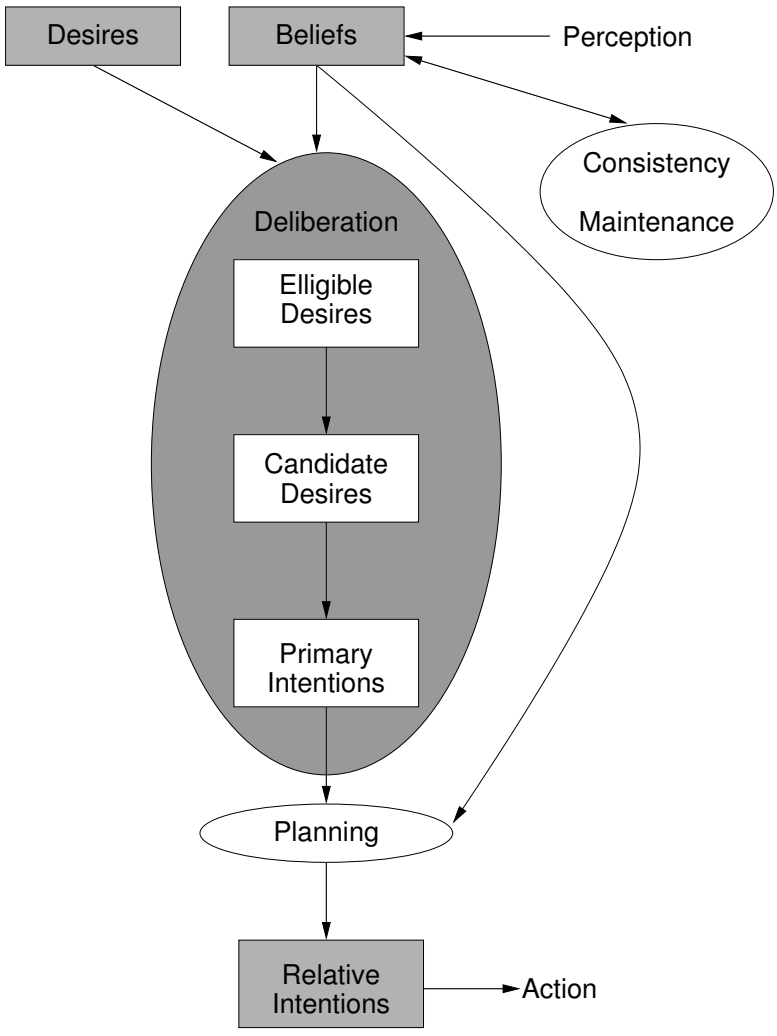


Fig. 3.1. *X-BDI operation overview.*

Eligible desires have rationality constraints that are similar to those imposed by Bratman [6] over intentions in the sense that an agent will not desire something in the past or something the agent believes will happen without its interference. Agent beliefs must also support the pre-conditions defined in the desire *Body*. Within the agent's reasoning process these desires give rise to a set of mutually consistent subsets ordered by a partial order relation.

The process of selecting candidate desires seeks to choose from the eligible desires one subset that contains only desires that are internally consistent and possible. A possible desire in this sense is one that has a property

P that can be satisfied through a sequence of actions. In order to choose among multiple sets of candidate desires, the original X-BDI uses ELP constructs that allow desires to be prioritised in the revision process. Although we depart from the original abduction theory, we still use these priority values to define a desire preference relation. Through this preference relation, a desire preference graph that relates all subsets of eligible desires is generated.

Candidate desires represent the most significant modification made in this paper regarding the original X-BDI [27]. Originally, X-BDI verified the possibility of a desire through the abduction of an event calculus theory in which the belief in the validity of a desired property P could be true. Such an abduction process is, actually, a form of planning. Since our main objective in this paper is to distinguish the planning process previously hard-coded within X-BDI, the notion of desire possibility must be re-defined. Therefore, we define the set of candidate desires to be the subset of eligible desires with the greater preference value, and whose properties can be satisfied. Satisfiability is verified through the execution of a propositional planner that processes a planning problem in which the initial state contains the properties that the agent believes at the time of planning. The P properties present in the candidate desires are used to generate the set of primary intentions. The modified reasoning process for X-BDI is illustrated in Figure 3.2.

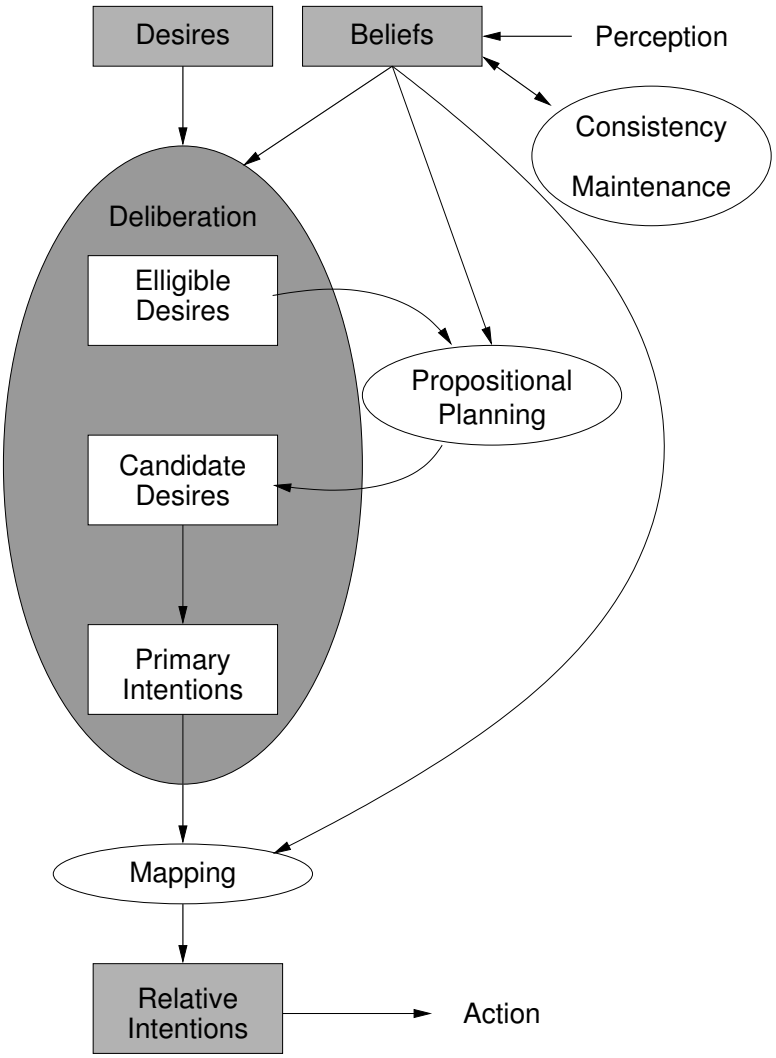


Fig. 3.2. *Modified X-BDI overview.*

Primary intentions can be seen as high-level plans, similar to the intentions in IRMA [7], and representing the agent's commitment to a course of action. These primary intentions are systematically refined up to the point where an agent has a temporally ordered set of actions representing a concrete plan towards the satisfaction of

its goals. Relative intentions then correspond to the temporally ordered steps of the concrete plans generated to satisfy the agent's primary intentions. Thus the notion of agent commitment results from the fact that relative intentions must not contradict or annul primary intentions.

3.3. Intention Revision. The computational effort and the time required to reconsider the whole set of intentions of a resource-bounded agent is generally significant regarding the environment change ratio. Intention reconsideration should therefore not occur constantly, but only when the world changes in such a way as to threaten the plans an agent is executing or when an opportunity to satisfy more important goals is detected. As a consequence, X-BDI uses a set of reconsideration *triggers* generated when intentions are selected, and causes the agent to reconsider its course of action when activated.

These trigger conditions are defined to enforce Bratman's [6] rationality conditions for BDI components, as follows. If all of the agent's primary intentions are satisfied before the time planned for them to be satisfied, the agent restarts the deliberative process, since it has achieved its goals. On the other hand, if one of the primary intentions is not achieved at the time planned for it, the agent must reconsider its intentions because its plans have failed. Moreover, if a desire with a higher priority than the currently selected desires becomes possible, the agent reconsiders its desires in order to take advantage of the new opportunity. Reconsideration is completely based on integrity constraints over beliefs, and since beliefs are revised at every sensoring cycle, it is possible that reconsideration occurs due the *triggering* of a reconsideration restriction.

3.4. Implementation. The prototype implemented for this work is composed of three parts: the X-BDI kernel, implemented in Prolog; a planning system containing a C++ implementation of Graphplan; and a Java graphical interface used to ease the operation of X-BDI and to visualise its interaction with the environment. The architecture is outlined in Figure 3.3.

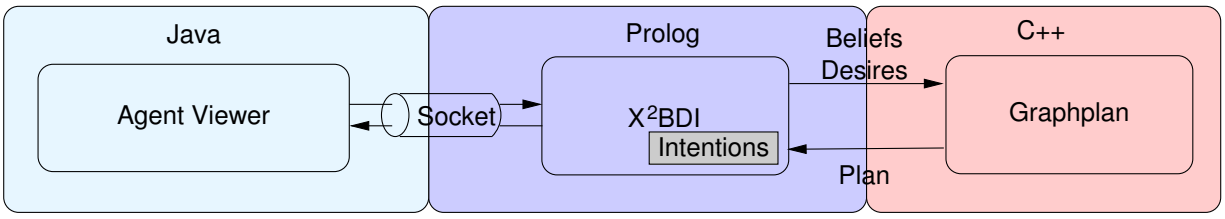


Fig. 3.3. *Solution Architecture*

Here, the *Agent Viewer* interface communicates with X-BDI through sockets by sending the input from the environment in which the agent is embedded and receiving the result of the agent's deliberation. Through the *Agent Viewer* the user can also specify the agent in terms of its desires, actions and initial beliefs. Once X-BDI receives the agent specification, it communicates with the planning module through operating system files and the Prolog/C++ interface. The planner is responsible for generating a set of intentions for the agent. When the agent deliberates, it converts subsets of the agent's desired properties into propositional planning problems and invokes the planning algorithm to solve these problems until either a plan that solves the highest priority desires is found, or the algorithm determines that it is not possible to solve any one of these problems.

4. A BDI Production Cell. In this work we use a BDI agent in order to model a production cell as a case study, and as a means to verify the validity of the architecture described in Section 3. In particular, the rational utilisation of equipment in industrial facilities is a complex problem, especially scheduling its use. This problem is complicated when the facility produces multiple component types, where each type requires a subset of the equipment available. In our test scenario, the proposed production cell [46], illustrated in Figure 4.1, is composed of seven devices: a *feed belt*, a *deposit belt* and four *processing units* upon which components are moved to be processed.

Components enter the production cell for processing through the feed belt and, once processed by all the appropriate processing units, they are removed from the cell through the deposit belt. Every processing unit is responsible for performing a different kind of operation on the component being processed, and can hold only one component at a given moment. Each component introduced into the cell can be processed by one or more processing units, determined by the type of component being processed, and different component types have different processing priorities. The control of the production cell is entrusted to a BDI agent implemented using

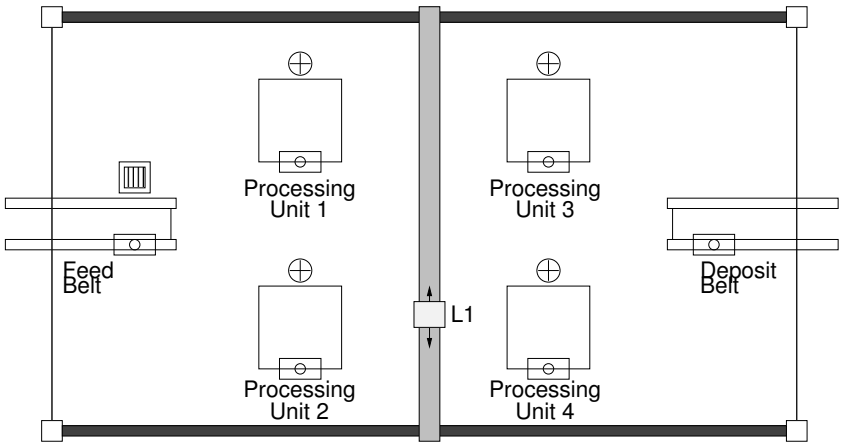


FIG. 4.1. *A BDI Production Cell.*

X-BDI, which should schedule the work of the production cell in relation to its beliefs, desires and intentions, re-scheduling whenever some change in the system occurs.

The first step in modelling any problem using a STRIPS-like formalism is the choice of the predicates used to represent the problem's object-types and its states. We define the following predicates representing objects in the cell:

- `component(C)` denotes that `C` is a *component* to be processed;
- `procUnit(P)` denotes that `P` is a *processing unit*, which is also a device;
- `device(D)` denotes that `D` is a *device*;
- `feedBelt` represents the *feed belt*;
- `depositBelt` represents the *deposit belt*.

Similarly, we have the following predicates representing system states:

- `over(C,D)` denotes that component `C` is over device `D`;
- `empty(P)` denotes that processing unit `P` is empty, *i.e.* has no component over it;
- `processed(C,P)` denotes that component `C` has already been processed by processing unit `P`;
- `finished(C)` denotes that component `C` has already been processed by all appropriate processing units and has been removed from the production cell;

Next, we define the actions the agent is capable of performing in the context of the proposed problem, these are summarised in Table 4.1. Informally, action `process(C,P)` represents the processing that a processing unit `P` performs on a component `C` over it; `consume(C)` represents the removal of component `C` from the production cell through the deposit belt; and `move(C,D1,D2)` represents the motion of component `C` from device `D1` to device `D2`.

TABLE 4.1
Action specification for the production cell agent.

Action	Preconditions	Effects

`process(C,P)`	`procUnit(P)`
	`component(C)`
	`over(C,P)`
		`processed(C,P)`

`consume(C)`	`component(C)`
	`over(C,depositBelt)`
		¬`over(C,depositBelt)`
		`empty(depositBelt)`
		`finished(C)`

`move(C,D1,D2)`	`over(C,D1)`
	`empty(D2)`
	`component(C)`
	`device(D1)`
	`device(D2)`
		`over(C, D2)`
		¬`over(C,D1)`
		¬`empty(D2)`
		`empty(D1)`

The processing requirements of components and their priorities are modelled through desires. Thus, we can model an agent, `pCell`, which needs to process component `comp1` by processing units `procUnit1`, `procUnit2` and `procUnit3` as soon as this component is inserted into the production cell using the specification of Listing 1[2].

Listing 1
Specification of desires related to processing `comp1`.

```
des(pCell,finished(comp1),Tf,[0.7])
 if bel(pCell, component(comp1)),
    bel(pCell, processed(comp1,procUnit1)),
    bel(pCell, processed(comp1,procUnit2)),
    bel(pCell, processed(comp1,procUnit3)),
    bel(pCell, -finished(comp1)).

des(pCell,processed(comp1,procUnit1),Tf,[0.6])
 if bel(pCell, component(comp1)),
    bel(pCell, -processed(comp1,procUnit1)).

des(pCell,processed(comp1,procUnit2),Tf,[0.6])
 if bel(pCell, component(comp1)),
    bel(pCell, -processed(comp1,procUnit2)).

des(pCell,processed(comp1,procUnit3),Tf,[0.6])
 if bel(pCell, component(comp1)),
    bel(pCell, -processed(comp1,procUnit3)).
```

Similarly, we can model the agent's need to process component `bloc2` by processing unit `procUnit3` and `procUnit4` by adding to the agent specification the desires of Listing 2.

Listing 2
Specification of desires related to processing `comp2`.

```
des(pCell,finished(comp2),Tf,[0.6])
 if bel(pCell, component(comp2)),
    bel(pCell, processed(comp2,procUnit3)),
    bel(pCell, processed(comp2,procUnit4)),
    bel(pCell, -finished(comp2)).

des(pCell,processed(comp2,procUnit3),Tf,[0.5])
 if bel(pCell, component(comp2)),
    bel(pCell, -processed(comp2,procUnit3)).

des(pCell,processed(comp2,procUnit4),Tf,[0.5])
 if bel(pCell, component(comp2)),
    bel(pCell, -processed(comp2,procUnit4)).
```

Finally, we model the agent's static knowledge regarding the problem domain, in particular the object's classes and the initial world-state with the beliefs specified in Listing 3.

The arrival of a new component in the production cell is signalled by the sensors through the inclusion of `component(comp1)` and `over(comp1,feedBelt)` in the agent's belief database, activating the agent's reconsideration process. Given the desire's pre-conditions previously defined, only the desires related to the following properties become eligible:

[2] `Tf` is the time at which the desire is valid, and the values `0.7` and `0.6` are the desires priorities.

Listing 3
Domain knowledge for the production cell.

```
bel(pCell, procUnit(procUnit1)).
bel(pCell, procUnit(procUnit2)).
bel(pCell, procUnit(procUnit3)).
bel(pCell, procUnit(procUnit4)).
bel(pCell, device(procUnit1)).
bel(pCell, device(procUnit2)).
bel(pCell, device(procUnit3)).
bel(pCell, device(procUnit4)).
bel(pCell, device(depositBelt)).
bel(pCell, device(feedBelt)).
bel(pCell, empty(procUnit1)).
bel(pCell, empty(procUnit2)).
bel(pCell, empty(procUnit3)).
bel(pCell, empty(procUnit4)).
bel(pCell, empty(depositBelt)).
```

- `processed(comp1,procUnit1)`;
- `processed(comp1,procUnit2)`;
- `processed(comp1,procUnit3)`;

These desires are then analysed by the process of selecting candidate desires. In this process, the agent's eligible desires and beliefs are used in the creation of planning problems that are sent to Graphplan for resolution. The result of this processing is a plan that satisfies all the eligible desires, with the following steps:

1. `move(comp1,feedBelt,procUnit2)`
2. `process(comp1,procUnit2)`
3. `move(comp1,procUnit2,procUnit1)`
4. `process(comp1,procUnit1)`
5. `move(comp1,procUnit1,procUnit3)`
6. `process(comp1,procUnit3)`

The existence of this plan indicates to X-BDI that the specified set of eligible desires is possible, thus turning the previous set of desires into candidate desires, which generate primary intentions representing the agent's commitment. Next, relative intentions are generated using the steps in the recently created plan, with one intention for each step of the plan. These lead the agent to perform the appropriate actions. Once the actions are executed, the candidate desires from the previous deliberation are satisfied. Moreover, the pre-condition of the desire to accomplish `finished(comp1)` becomes true, reactivating the agent's deliberative process and generating the following plan:

1. `move(comp1,procUnit3,depositBelt)`
2. `consume(comp1)`

Once more, this plan brings about some intentions and, eventually, leads the agent to act. Now, suppose that during the agent's operation, a new component in the production cell arrives. If this occurred immediately after the deliberation that created the first plan, it would be signaled by the agent's sensors through the inclusion of `component(comp2)` and `over(comp2,feedBelt)` in the beliefs database, which would modify the eligible desires chosen in the second deliberation cycle to:

- `finished(comp1)`;
- `processed(comp2,procUnit3)`;
- `processed(comp2,procUnit4)`;

These desires become candidate desires because Graphplan is capable of generating a plan that satisfies all the desires. The new plan is:

1. `move(comp1,procUnit3,depositBelt)`
2. `move(comp2,feedBelt,procUnit4)`
3. `consume(comp1)`

4. `process(comp2,procUnit4)`
5. `move(comp2,procUnit4,procUnit3)`
6. `process(comp2,procUnit3)`
7. `move(comp2,procUnit3,depositBelt)`
8. `consume(comp2)`

The steps of this plan thus generate relative intentions, eventually leading the agent to the execution of its actions.

5. Conclusions. In this paper, we have discussed the relationship between propositional planning algorithms and means-end reasoning in BDI agents. To test the viability of using propositional planners to perform means-ends reasoning in a BDI architecture, we have described a modification to the X-BDI agent model. Throughout this modification, new definitions of desires and intentions were created in order for the agent model to maintain the theoretical properties present in its original version, especially regarding the definition of desires and intentions impossibility. Moreover, it was necessary to define a mapping between the structural components of a BDI agent and propositional planning problems. The result of implementing these definitions in a prototype can be seen in the case study of Section 4, which represents a problem that the means-end reasoning process of the original X-BDI could not solve.

Considering that most implementations of BDI agents use a plan library for means-end reasoning in order to bypass the inherent complexity of performing planning at runtime, X-BDI offers an innovative way of implementing more flexible agents through its fully declarative specification. However, its planning mechanism is notably inefficient. For example, the case study described in Section 4 was not tractable in the original X-BDI planning process. Thus, the main contribution of our work consists in addressing this limitation through the definition of a mapping from BDI means-end reasoning to fast planning algorithms. Moreover, such an approach enables the agent architecture to be extended with any propositional planning algorithm that uses a formalism compatible with the proposed mapping, thus allowing an agent to use more powerful planners as they become available, or to use more suitable planning strategies for different problem classes.

Other approaches to performing runtime planning have also been proposed, the most notable recent ones by Sardina *et al.* [36] and Walczak *et al.* [41]. Sardina proposes the tight integration of the JACK agent framework [8] with the SHOP hierarchical planner [29]. This approach relies on new constructs added to an otherwise procedural agent representation and takes advantage of the similarity of *hierarchical task network* (HTN) planning to BDI reasoning. The work of Walczak proposes the integration of JADEX [32]ike]What is JADEX? to a customised knowledge-based planner operating in parallel to agent execution, using a similar process of agent-state conversion to work of Meneguzzi *et al.* [26, 47], as well as the one presented in this paper.

Some ramifications of this work are foreseen as future work, in particular, the incorporation of the various Graphplan improvements, as well as the conduction of tests using other propositional planning algorithms, SAT being an example. It is clear that other agent architectures can benefit from the usage of planning components to allow agents to cope with unforeseen events at runtime, as demonstrated by recent efforts in planning agents [36, 41]. Therefore, investigating how to integrate planning capabilities to AgentSpeak-based agents could create agents that can take advantage of both the fast response of pre-compiled plans and the flexibility of being able to plan at runtime to cope with unforeseen situations.

Acknowledgments. We wish to acknowledge X, Y and Z for the support in this work.

REFERENCES

[1] J. J. Alferes and L. M. Pereira, *Reasoning with Logic Programming*, Springer Verlag, 1996.
[2] S. Biundo, K. L. Myers, and K. Rajan, eds., *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA*, AAAI, 2005.
[3] A. L. Blum and M. L. Furst, *Fast planning through planning graph analysis*, Artificial Intelligence, 90 (1997), pp. 281–300.
[4] R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, *Multi-Agent Programming: Languages, Platforms and Applications*, vol. 15 of Multiagent Systems, Artificial Societies, and Simulated Organizations, Springer, 2005.
[5] R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge, *Model checking AgentSpeak*, in Proceedings of the 2nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS-03), Melbourne, Australia, July 2003, ACM Press, pp. 409–416.
[6] M. E. Bratman, *Intention, Plans and Practical Reason*, Harvard University Press, Cambridge, MA, 1987.
[7] M. E. Bratman, D. J. Israel, and M. E. Pollack, *Plans and resource-bounded practical reasoning*, Computational Intelligence, 4 (1988), pp. 349–355.

[8] P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas, *Jack intelligent agents—components for intelligent agents in java.* AgentLink Newsletter, January 1999. White paper, `http://www.agent-software.com.au`

[9] T. Bylander, *The computational complexity of propositional STRIPS planning*, Artificial Intelligence, 69 (1994), pp. 165–204.

[10] D. Chapman, *Planning for conjunctive goals*, Artificial Intelligence, 32 (1987), pp. 333–377.

[11] P. R. Cohen and H. J. Levesque, *Intention is choice with commitment*, Artificial Intelligence, 42 (1990), pp. 213–261.

[12] J. S. Cox, E. H. Durfee, and T. Bartold, *A distributed framework for solving the multiagent plan coordination problem*, in AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, 2005, ACM Press, pp. 821–827.

[13] W. V. der Hoek and M. Wooldridge, *Towards a logic of rational agency*, Logic Journal of the IGPL, 11 (2003), pp. 133–157.

[14] M. d'Inverno and M. Luck, *Engineering AgentSpeak(L): A formal computational model*, Journal of Logic and Computation, 8 (1998), pp. 233–260.

[15] R. Fikes and N. Nilsson, *STRIPS: A new approach to the application of theorem proving to problem solving*, Artificial Intelligence, 2 (1971), pp. 189–208.

[16] M. Georgeff, B. Pell, M. E. Pollack, M. Tambe, and M. Wooldridge, *The belief-desire-intention model of agency*, in Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98), J. Müller, M. P. Singh, and A. S. Rao, eds., vol. 1555, Springer-Verlag: Heidelberg, Germany, 1999, pp. 1–10.

[17] J. Hoffmann and B. Nebel, *The FF planning system: Fast plan generation through heuristic search*, Journal of Artificial Intelligence Research (JAIR), 14 (2001), pp. 253–302.

[18] F. F. Ingrand, M. P. Georgeff, and A. S. Rao, *An architecture for real-time reasoning and system control*, IEEE Expert, Knowledge-Based Diagnosis in Process Engineering, 7 (1992), pp. 33–44.

[19] N. R. Jennings, *On agent-based software engineering*, Artificial Intelligence, 117 (2000), pp. 277–296.

[20] H. Kautz and B. Selman, *Planning as satisfiability*, in Proceedings of the Tenth European Conference on Artificial Intelligence, Chichester, UK, 1992, Wiley, pp. 359–363.

[21] J. Köhler, *Solving complex planning tasks through extraction of subproblems*, in Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, R. Simmons, M. Veloso, and S. Smith, eds., Pittsburg, 1998, AAAI Press, pp. 62–69.

[22] J. Köhler, B. Nebel, J. Hoffmann, and Y. Dimopoulos, *Extending planning graphs to an ADL subset*, in Proceedings of the 4th European Conference on Planning, S. Steel, ed., vol. 1348 of Lecture Notes in Computer Science, Springer Verlag, Germany, 1997, pp. 273–285.

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

[24] D. Long and M. Fox, *Efficient implementation of the plan graph in STAN*, Journal of Artificial Intelligence Research (JAIR), 10 (1999), pp. 87–115.

[25] D. Long and M. Fox„ *Automatic synthesis and use of generic types in planning*, in Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, Colorado, 2000, AAAI Press, pp. 196–205.

[26] F. R. Meneguzzi, A. F. Zorzo, and M. D. C. Móra, *Propositional planning in BDI agents*, in Proceedings of the 2004 ACM Symposium on Applied Computing, Nicosia, Cyprus, 2004, ACM Press, pp. 58–63.

[27] M. d. C. Móra, J. G. P. Lopes, R. M. Vicari, and H. Coelho, *BDI models and systems: Bridging the gap.*, in Intelligent Agents V, Agent Theories, Architectures, and Languages, Fifth International Workshop, ATAL '98, vol. 1555 of LNCS, Springer, Paris, France, 1999, pp. 11–27.

[28] J. P. Müller, *The design of intelligent agents: A layered approach*, in The Design of Intelligent Agents: A Layered Approach, vol. 1177 of Lecture Notes in Computer Science, Springer Verlag, Germany, 1996.

[29] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila, *Shop: Simple hierarchical ordered planner.*, in IJCAI, 1999, pp. 968–975.

[30] B. Nebel, *On the compilability and expressive power of propositional planning formalisms*, Journal of Artificial Intelligence Research (JAIR), 12 (2000), pp. 271–315.

[31] N. Nide and S. Takata, *Deduction systems for BDI logics using sequent calculus*, in Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, ACM Press, 2002, pp. 928–935.

[32] A. Pokahr, L. Braubach, and W. Lamersdorf, *Jadex: Implementing a bdi-infrastructure for jade agents*, EXP—in search of innovation (Special Issue on JADE), 3 (2003), pp. 76–85.

[33] A. S. Rao, *AgentSpeak(L): BDI agents speak out in a logical computable language*, in Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, W. V. de Velde and J. W. Perram, eds., vol. 1038 of LNCS, Springer, Eindhoven, The Netherlands, 1996, pp. 42–55.

[34] A. S. Rao and M. P. Georgeff, *Formal models and decision procedures for multi-agent systems*, Tech. Report 61, Australian Artificial Intelligence Institute, 171 La Trobe Street, Melbourne, Australia, 1995. Technical Note.

[35] S. J. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, New Jersey, 1994.

[36] S. Sardina, L. de Silva, and L. Padgham, *Hierarchical Planning in BDI Agent Programming Languages: A Formal Approach*, in AAMAS '06: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan, 2006, ACM Press, pp. 1001–1008.

[37] M. Schut and M. Wooldridge, *The control of reasoning in resource-bounded agents*, The Knowledge Engineering Review, 16 (2001).

[38] D. E. Smith and D. S. Weld, *Temporal planning with mutual exclusion reasoning*, in Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI), 1999, pp. 326–337.

[39] J. R. Surdu, J. M. D. Hill, and U. W. Pooch, *Anticipatory planning support system*, in WSC '00: Proceedings of the 32nd conference on Winter simulation, San Diego, CA, USA, 2000, Society for Computer Simulation International, pp. 950–957.

[40] B. van Riemsdijk, M. Dastani, F. Dignum, and J.-J. C. Meyer, *Dynamics of declarative goals in agent programming.*, in LNCS, vol. 3476, 2004, pp. 1–18.
[41] A. Walczak, L. Braubach, A. Pokahr, and W. Lamersdorf, *Augmenting BDI Agents with Deliberative Planning Techniques*, in The Fifth International Workshop on Programming Multiagent Systems (PROMAS-2006), 2006.
[42] P. Wegner, *Why interaction is more powerful than algorithms*, Communications of the ACM, 40 (1997), pp. 80–91.
[43] D. S. Weld, *Recent Advances in AI Planning*, AI Magazine, 20 (1999), pp. 93–123.
[44] M. Wooldridge, *Reasoning about Rational Agents*, The MIT Press, 2000.
[45] M. Wooldridge, *The Computational Complexity of Agent Design Problems*, in Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS 2000), E. Durfee, ed., IEEE Press, 2000, pp. 341–348.
[46] A. F. Zorzo, L. A. Cassol, A. L. Nodari, L. A. Oliveira, and L. R. Morais, *Scheduling safety-critical systems using a partial order planning algorithm*, in Advances in Logic, Artificial Intelligence and Robotics, São Paulo, Brazil, 2002, IOS Press, pp. 33–40.
[47] A. F. Zorzo and F. R. Meneguzzi, *An agent model for fault-tolerant systems*, in SAC '05: Proceedings of the 2005 ACM symposium on Applied computing, New York, NY, USA, 2005, ACM Press, pp. 60–65.

Edited by: Marcin Paprzycki, Niranjan Suri
Received: October 1, 2006
Accepted: December 10, 2006

Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 29–40. http://www.scpe.org
ISSN 1895-1767

AGENT TECHNOLOGY FOR PERSONALIZED INFORMATION FILTERING: THE PIA SYSTEM

SAHIN ALBAYRAK*, STEFAN WOLLNY†, ANDREAS LOMMATZSCH‡, AND DRAGAN MILOSEVIC‡

Abstract. As today the amount of accessible information is overwhelming, the intelligent and personalized filtering of available information is a main challenge. Additionally, there is a growing need for the seamless mobile and multi-modal system usage throughout the whole day to meet the requirements of the modern society (anytime, anywhere, anyhow). A personal information agent that is delivering the right information at the right time by accessing, filtering and presenting information in a situation-aware matter is needed. Applying Agent-technology is promising, because the inherent capabilities of agents like autonomy, pro- and reactiveness offer an adequate approach. We developed an agent-based personal information system called PIA for collecting, filtering, and integrating information at a common point, offering access to the information by WWW, e-mail, SMS, MMS, and J2ME clients. Push and pull techniques are combined allowing the user to search explicitly for specific information on the one hand and to be informed automatically about relevant information divided in pre-, work and recreation slots on the other hand. In the core of the PIA system advanced filtering techniques are deployed through multiple filtering agent communities for content-based and collaborative filtering. Information-extracting agents are constantly gathering new relevant information from a variety of selected sources (internet, files, databases, web-services etc.). A personal agent for each user is managing the individual information provisioning, tailored to the needs of this specific user, knowing the profile, the current situation and learning from feedback.

Key words. intelligent and personalized filtering, ubiquitous access, recommendation systems, agents and complex systems, agent-based deployed applications, evolution, adaptation and learning.

1. Introduction. Nowadays, desired information often remains unfound, because it is hidden in a huge amount of unnecessary and irrelevant data. On the Internet there are well maintained search engines that are highly useful if you want to do full-text keyword-search [1], but they are not able to support you in a personalized way and typically do not offer any push-services or in other words no information will be sent to you when you are not active. Also, as they normally do not adapt themselves to mobile devices, they cannot be used throughout a whole day because you are not sitting in front of a standard browser all the time and when you return, these systems will treat you in the very same way as if you have never been there before (no personalization, no learning). Users who are not familiar with domain-specific keywords won't be able to do successful research, because no support is offered. Predefined or auto-generated keywords for the search-domains are needed to fill that gap. As information demands are continuously increasing today and the gathering of information is time-consuming, there is a growing need for a personalized support (Figure 1.1). Labor-saving information is needed to increase productivity at work and also there is an increasing aspiration for a personalized offer of general information, specific domain knowledge, entertainment, shopping, fitness, lifestyle and health information. Existing commercial personalized systems are far away from that functionality, as they usually do not offer much more than allowing to choose the kind of the layout or collecting some of the offered information channels (and the information within is not personalized).

To overcome that situation you need a personal information agent (PIA) who knows the way of your thinking and can really support you throughout the whole day by accessing, filtering and presenting information to you in a situation-aware matter (Figure 1.1). Some systems exist (Fab [2], Amalthaea [3], WAIR [4], P-Tango [5], Trip-Matcher [6], PIAgent [7], Letizia [8], Let's Browse [9], Newt [10], WebWatcher [11], PEA [12], BAZAR [13]) that implement advanced algorithmic technology, but did not become widely accepted, maybe because of real world requirements like availability, scalability and adaptation to current and future standards and mobile devices.

In this paper we present an agent-based approach for the efficient, seamless and tailored provisioning of relevant information on the basis of end-users' daily routine. As we assume the reader will be familiar with agent-technology (see [14, 15] for a good introduction), we will concentrate on the practical usage and the real-world advantages of agent-technology. After briefly describing the existing systems from which the scientific publications are available, we describe the design and architecture and afterwards depict the system in Section 4.

2. Related work. The following paragraphs are going to briefly present some of the already mentioned systems (Fab, Amalthaea, WAIR, P-Tango, PIAgent, Letizia, Let's Browse, Newt, WebWatcher and PEA), for which we believe that are related to our work.

*DAI-Labor, Technical University Berlin, Germany (sahin@dai-lab.de)
†High Performance Computing, Zuse Institute Berlin, Germany (wollny@zib.de)
‡DAI-Labor, Technical University Berlin, Germany ({andreas, dragan}@dai-lab.de)

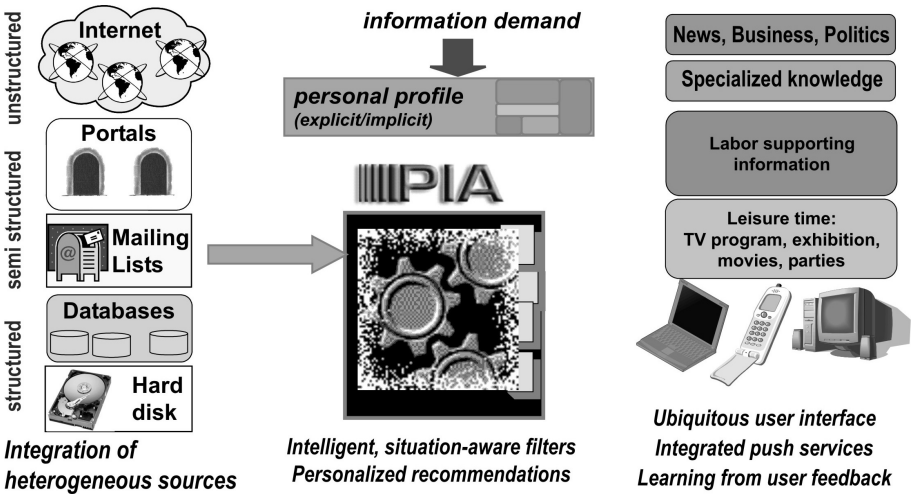


Fig. 1.1. *Demand for a personal information agent*

Fab [2] is an automatic recommendation service for information retrieval, which is able to over time adapt to its users, who consequently receive increasingly personalized documents. By maintaining both collection and selection agents, Fab system is a good test-bed for trying out different filtering strategies, which either collect documents from the Web that belong to the certain topic, or select some of the collected documents that are suitable for a particular user. The creation of profiles through the content-based analysis, which are afterwards directly compared to find similar users for collaborative recommendations, represents the unique synergy of these two frequently combined filtering techniques. Unfortunately, the usability of the whole system depends on the ability of the content based filtering to generate the usable profiles, being the serious drawback of the Fab system.

Since information discovery and information filtering are proven to be the suitable domains for applying multi-agent technology, a personalized system, named Amalthaea [3] has been developed. It proactively tries to discover from various distributed sources the information that is relevant to a user. The multi-agent technology is applied by maintaining two different types of agents, being information filtering and information discovery ones. The ways how these agents are managing to learn the user's interests and habits, to maintain their competence by adapting to the changes in the user's information needs, and to explore the new domains that may be of interest to a user, depend on evolution programming, being maybe not so applicable for the large-scale information retrieval tasks.

Seeking the state of a user profile, which best represents actual information interests and therefore maximize the expected value of the cumulative user relevance feedback, is formulated in WAIR multi-agent system [4] as the reinforcement learning problem. The insufficiency of explicit user ratings is tried to be overcome by using the classification approach based on the neural network, which exploits different implicit indicators of interests in order to estimate the real relevance feedback values. Unfortunately, the amount of the explicit ratings needed for training that classifier still seems to be too large. This clearly limits the applicability of the WAIR system.

To intelligently deliver a personalized newspaper, which contains only the articles of highest interest that are individually selected everyday for each and every user, P-Tango [5] system proposes a framework for combining different filtering strategies. Although the currently combined strategies are only the content-based and collaborative ones, a proposed framework is significant, by reason of being extendible to any filtering methods. In spite of this extensibility, we believe that the agent-based framework that we propose in this paper, offers better flexibility when the integration and afterwards the usage of new strategies is concerned.

As the information became the one of most significant resources for business and research, both periodically scanning different information sources and pushing the found relevant articles to interested users, have also motivated the development of PIAgent [7]. While a usage of various extractor agents each supporting a particular information source is more or less typical for agent-based filtering systems (and it is also present in our approach), the uniqueness of PIAgent lies in its application of back propagation neural network for separating

relevant articles from others. Such a neural network approach has strength in optimistically providing excellent classification accuracy. Unfortunately, its big weakness in often expensive training that practically makes the PIAgent to be hardly applicable for nowadays information retrieval tasks.

The intelligent assistance to the user, who is browsing the Internet for the interesting information, is provided by the autonomous interface agent, named Letizia [8]. It tracks user behavior and uses various heuristics to anticipate, which hyperlinks may lead to the potentially relevant documents, and which should be ignored by reason of pointing to junk or not existing page. The cornerstone property of the Letizia system is in asking the user neither to explicitly state its interests by defining the query nor to provide the explicit feedback about a real relevance of recommendations. Although this explicit communication with the user can speed up learning, the priority in designing the Letizia system has been given to both letting the user to browse without being interrupted and asking for help only when being unsure which link to follow.

The MIT Media Laboratory has also developed an agent, whose job is to choose, from the links reachable on the current Web page, those that are likely to best satisfy the interests of multiple users. The agent is named Let's Browse [9], by reason of providing the assistance to the group of humans in browsing, by suggesting hyperlinks likely to be of common interests. Although this system demonstrates how documents that are good for the group of users and that are in the neighborhood can be found, it generally does not respond to the challenge of finding the data that is located anywhere on the Internet.

The ability to both specialize to user interests, adapt to preference changes and explore the newer information domains makes the foundation of the NewT [10], being one personalized multi-agent filtering system for news articles. As user information interests are modeled as the population of the competing profiles, the used learning mechanisms are both relevance feedback, as well as the crossover and mutation genetic operators. These recombination genetic operators are mainly responsible for the adaptation and exploration issues by creating more fitted future populations. In the meantime, a user profile also learns through the application of the relevance feedback techniques. Taken together these learning mechanisms make the so-called Baldwin effect [24], saying that a population evolves towards a fitter form much faster, whenever its members are allowed to learn during their lifetime. Although the Baldwin evolution seems to be more powerful than the evolution approach used in Amalthaea, it has the same weaknesses which limit its applicability for large-scale information retrieval.

Users may find it difficult both to create the appropriate queries and to locate the information of interest in the case of having no specific knowledge about the content of the underlying document collection. On the one hand, some systems aim to deploy efficient clustering algorithms, which will dynamically produce the table of contents, needed to facilitate the users' browsing activities. The cornerstone idea is to by some means help a user first to get an overview concerning the available content, and then to accurately express its information needs. On the other hand, WebWatcher [11] acts as the tour guide that provides the assistance, which is similar to the guidance of the human in the real museum. It accompanies users from page to page, suggests appropriate hyperlinks, and learns from the obtained experience to improve its advice giving skills. Such a system unfortunately can only locally assist the user, which brings the same drawbacks being present in Letizia and Let's Browse systems.

Personal Email Assistant (PEA) [12] filters incoming mails and ranks them according to their relevance in order to help nowadays users, who easily end up with large part of their working day being spent with reading emails. PEA maintains the personal user model that consists of several profiles and uses the evolutionary algorithms to move that model constantly closer to the current information needs. By doing that PEA aims at assisting users in dealing more effectively with their daily load of emails so that valuable working time is saved for more productive and creative tasks. Even though the evolution strategies seems to be powerful enough for dealing with emails in the PEA system, their usage in the Internet-like environment still remains to be a great challenge.

3. Design of PIA: The Personal Information Agent. To meet the discussed requirements and to support the user in that matter, we designed a multi-agent system composed of four classes of agents: many information extracting agents, agents that implement different filtering strategies, agents for providing different kinds of presentation and one personal agent for each user. Logically, all this can be seen as a classical three tier application (Figure 3.1). Concerning the information extraction, general search engines on the one hand but also domain-specific portals on the other hand have to be integrated. Additional information sources (Databases, Files, Mailinglists etc.) should also be integrated easily at run-time.

Several agents realize different filtering strategies (content-based and collaborative filtering [16], [5]) that have to be combined in an intelligent matter. Also agents for providing information actively via SMS, MMS, Fax, e-mail (push-services) are needed. A Multi-access service platform has to manage the presentation of the results tailored to the used device and the current situation. The personal agent should constantly improve the knowledge about his user by learning from the given feedback, which is also taken for collaborative filtering, as information that has been rated as highly interesting might be useful for a user with a similar profile as well. As users usually are not very keen on giving explicit feedback (ratings), implicit feedback like the fact that the user stored an article can also be taken into account [18].

A keywordassistant should support the user in defining queries even if he is not familiar with a certain domain. PIA provides three strategies for finding adequate keywords and for optimizing existing requests:

1. Keywords predefined by experts for frequently requested topics (or categories) can help the unexperienced user to find the relevant keywords. The suggestions provided by domain experts are usually a good starting point for individual requests.
2. An alternative method for finding interesting keywords is the extraction of words and phrases from interesting papers. This strategy helps the user to identify the key concepts from a paper that can be useful for finding other relevant documents. In contrast to other approaches (like Googles Find similar documents) the keyword extraction gives the user the opportunity to adapt extracted keywords according to the personal interests and preferences.
3. For optimizing existing queries the PIA system suggests keywords from similar requests. For computing the similarities between user requests the systems analyses the overlapping of user profiles (based on stems) and the corelation between the user ratings. Keywords that frequently occure in the requests of similar users are suggested to the user as potentially relevant search terms.

The whole system is designed to be highly scalable, easy to modify, to adapt and to improve and therefore an agent-based approach that allows to integrate or to remove agents even at run-time is a smart choice. The different filtering techniques are needed to provide accurate results, because the weakness of individual techniques should be compensated by the strengths of others. Documents should be logically clustered by their domains to allow fast access, and for each document several models [19] will be built, all including stemming and stop-word elimination, but some tailored for very efficient retrieval at run-time and others to support advanced filtering algorithms for a high accuracy.

If the system notices that the content-based filtering is not able to offer sufficient results, additional information should be offered by collaborative filtering, i. e. information that was rated as interesting by a user with a similar profile will be presented.

With the push-services, the user can decide to get new integrated relevant information immediately and on a mobile device, but for users who do not want to get new information immediately, a personalized newsletter also has to be offered. This newsletter is collecting new relevant information to be conveniently delivered by e-mails, allowing users to stay informed even if they are not actively using the system for some time.

4. Deployment and evaluation.

4.1. Overview. We implemented the system using Java and standard open source database and web-technology and based on the JIAC IV agent framework [20]. JIAC IV is FIPA 2000 compliant [21], that is it is conforming to the latest standards.

It consists of a communication infrastructure as well as services for administrating and organizing agents (Agent Management Service, AMS and Directory Facilitator, DF). The JIAC IV framework provides a variety of management and security functions, management services including configuration, fault management and event logging, security aspects including authorization, authentication and mechanisms for measuring and ensuring trust and therefore has been a good choice to be used from the outset to the development of a real world application.

Within JIAC IV, agents are arranged on platforms, allowing the arrangement of agents that belong together with the control of at least one manager. A lot of visual tools are offered to deal with administration aspects. Figure 3.2 shows a platform, in this case with agents for the building of different models specialized for different retrieval algorithms.

The prototypical system is currently running on Sun-Fire-880, Sun-Fire-480R and Sun Fire V65x, whereas the main filtering computation, database- and web-server and information-extraction is placed on different machines for performance reasons.

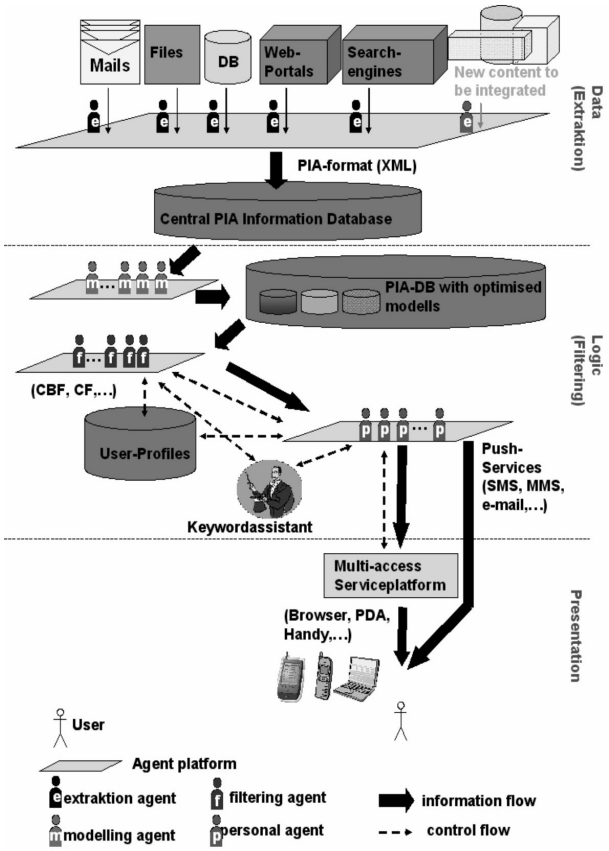


FIG. 3.1. *The PIA System seen as a three tier application*

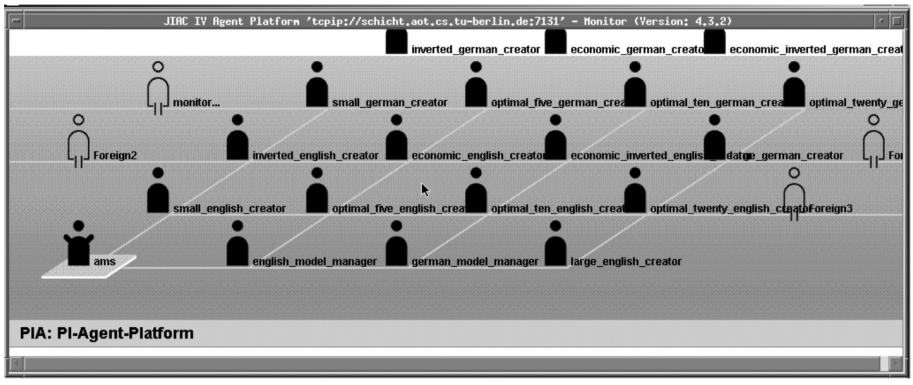


FIG. 3.2. *Several agents are building different models specilised for different retrieval algorithms*

New content is stored, validated and consolidated in a central relational database (update-driven). Information can be accessed by WWW, e-mail, SMS, MMS, and J2ME Clients, where the system adapts the presentation accordingly, using the CC/PP (Preferences Profile) with a tailored layout for a mobile phone and a PDA (see Section 4.6). The personalized newsletter and the push-services are sent via e-mail, SMS or MMS. The user can use self-defined keywords for a request for information or choose a category and therefore the system will use a list of keywords predefined by experts and updated smoothly by learning from collaborative filtering. A combination of both is also possible. The keyword assistant is able to extract the most import keywords of a given article using the term frequency inverse document frequency (TFIDF)-algorithm [22].

4.2. Gathering new information. New information is constantly inserted in the system by information extraction agents, e.g. web-reader agents or agents that are searching specified databases or directories. Additional agents for the collection of new content can easily be integrated even at runtime, as all that is necessary for a new agent is to register himself at the system, store the extracted information at a defined database table and inform the modeling-manager agent about the insertion. As a file reader-agent is constantly observing a special directory, manual insertion of documents can be done simply by drag-and-drop and an e-mail and upload-interface also exists. Source can also be integrated by Web services. New Readers can be created using a easy-to-handle tool and another tool is enabling to conveniently observe the extraction-agents, as this is the interface to the outside that might become critical if for example the data-format of a source is changed.

4.3. Pre-processing for efficient retrieval. The first step of pre processing information for efficient retrieval is the use of distinct tables in the global database for different domains like e.g. news, agent-related papers, etc. Depending on the filtering request, tables with no chance of being relevant can therefore be omitted. The next step is the building of several models for each document. Stemming and stop-word elimination is implemented in every model but different models are built by computing a term importance either based only on local frequencies, or based on term frequency inverse document frequency (TFIDF) approach. Furthermore number of words that should be included in models is different which makes models either more accurate or more efficient. Created models are indexed either on document or word level, which facilitate their efficient retrieval. The manager agent is assigning the appropriate modeling agents to start building their models but might decide (or the human system administrator can tell him) at runtime to delay latest time-consuming modeling activity for a while if system load is critical at a moment. This feature is important for a real-world application, as overloading has been a main reason for the un-usability of advanced academic systems.

4.4. Filtering technology. As the quality of results to a particular filtering request might heavily depend on the information domain (news, scientific papers, conference calls), different filtering communities are implemented. For each domain, there is at least one community which contains agents being tailored to do specific filtering and managing tasks in an efficient way. Instead of having only filtering agents (they will be described in Section 4.5), each and every community has also one so-called manager agent that is mainly responsible for doing coordination of filtering tasks, as well as cooperation with other managers.

The coordination is based on quality, CPU, DB and memory fitness values, which are the measures being associated to each filtering agent [23]. These measures respectively illustrate filtering agent successfulness in the past, its efficiency in using available CPU and DB resources, and the amount of memory being required for filtering. A higher CPU, DB or memory fitness value means that filtering agent needs a particular resource in a smaller extent for performing a filtering task. This further means that an insufficiency of a particular resource has a smaller influence on filtering agents with a higher particular fitness value.

The introduced different fitness values together with the knowledge about the current system runtime performance can make coordination be situation aware (see also [23]) in the way that when a particular resource is highly loaded a priority in coordination should be given to filtering agents for which a particular resource has a minor importance. This situation aware coordination provides a way to balance response time and filtering accuracy, which is needed in overcoming the problem of finding a perfect filtering result after few hours or even few days of an expensive filtering.

Instead of assigning filtering task to the agent with the best combination of fitness values in the current runtime situation, manager is going to employ a proportional selection principle [24] where the probability for the agent to be chosen to do actual filtering is proportional to the mentioned combination of its fitness values. By not always relying only on the most promising agents, but also sometimes offering a job to other agents, manager gives a chance to each and every agent to improve its fitness values. While the adaptation of quality fitness value can be accomplished after the user feedback became available, the other fitness values can be changed immediately after the filtering was finished through the response time analyses. The adaptation scheme has a decreasing learning rate that prevents already learnt fitness values of being destroyed, which further means that proven agents pay smaller penalties for bad jobs than the novice ones [17].

The underlying coordination activities, essentially responsible for the mentioned optimal exploitation of available system resources, are represented on Figure 4.1, giving the simplest possible selection scenario. Under the assumption that everything goes perfectly, the scenario starts with a job creation and ends with a result usage, being done by the User agent (U). The real coordination activities, being performed in a meantime by the chosen Manager (M), are first resource estimation, and afterwards strategy selection. After the selected Filtering

agent (F) that encapsulates the particular searching algorithm (deployed filtering strategies are described in Section 4.5), is produced results, the manager M can adapt fitness values based on the measurement of the response time. The found filtering results are finally returned back to the user agent U, and this simple scenario ends.

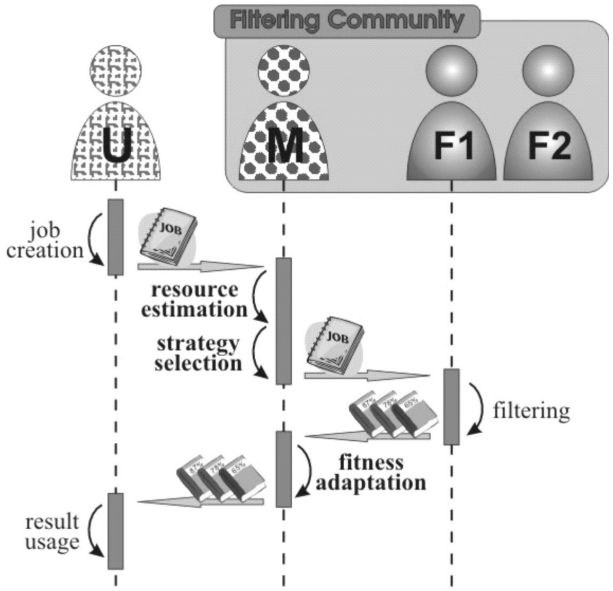


FIG. 4.1. *System architecture illustrating agent communication for resource-aware coordination*

In the case where the received filtering task cannot be successfully locally accomplished usually because of belonging to unsupported information domain, manager agent has to cooperate with other communities. While coordination takes place inside each and every filtering community between manager and filtering agents, cooperation occurs between communities among manager agents (see also Figure 4.2). The cooperation is based on either finding a community which supports given domain or in splitting received task on sub-tasks where for each sub-task a community with good support exists.

The information is usually scattered around many different, more or less dynamic, distributed sources. Two cornerstone challenges therefore become both finding which sources should be consulted for resolving the particular request, as well as putting the found results together. While the challenge of searching for sources is known as the database selection problem, the composing of a final result set is often simply referred as the information fusion. One light cooperation approach, already published in [25], and which is based on the application of the intelligent cooperative agents, is going to be briefly illustrated in the rest of this sub-section.

The fundamental cooperation idea is based on the installation of at least one filtering community around each database, as well as on setting up the sophisticated mechanisms, which enable that these communities can efficiently help each other while processing the incoming requests. Although the filtering request can be sent to any filtering community, the most suitable communities will be autonomously found, and the request will be then dispatched to them. The found results will be finally collected, and only the best ones will be returned to the sender of the filtering request. The most appealing property behind these cooperative processing is that everything is done transparently for the user, being not any more forced to manually think where the request should be sent, and which obtained filtering results are really the best ones.

Example (Coordination & Cooperation) Figure 4.2 presents a high level overview of the filtering framework being composed of three different filtering communities (FC), where each community has one filter manager agent (M) and different number of specialized filtering agents (F). There are two different databases (DB) with information from different domains, and they are accessed at least by one community. On the figure cooperation is illustrated as a circle with arrows which connect manager agents.

4.5. Filtering strategies. The cornerstone of the PIA system is in offering a framework that facilitates the integration of different filtering strategies. Although this paper is not dealing with any particular filtering

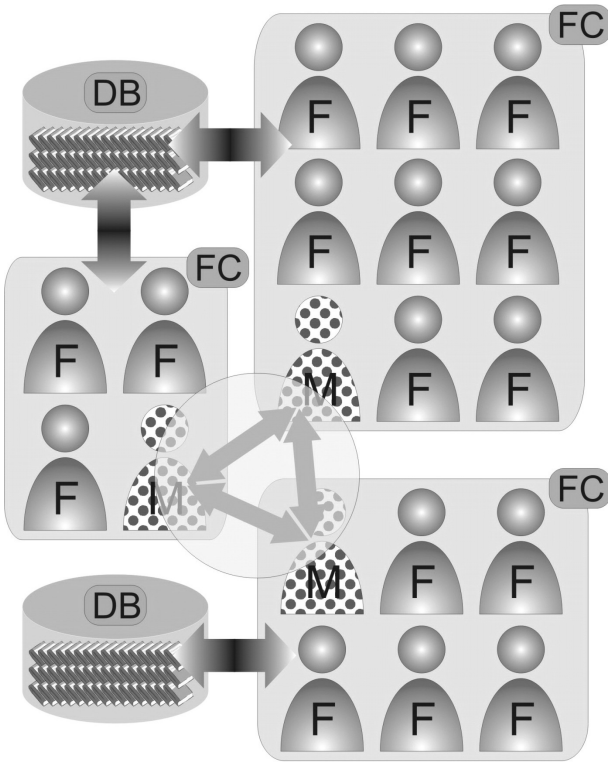


Fig. 4.2. *Filtering framework with three different communities*

strategy, their short descriptions will be given in the following paragraphs in order to make this paper self-contained and to clear the roles of the agents on Figure 3.2.

By using the term frequency inverse document frequency scheme, the importance values of different words can be computed, and each and every document can be modelled by a corresponding weighted vector. While the so-called *Large Filtering Strategy* will always build a model with all words from a document, *Optimal Twenty*, *Optimal Ten*, and *Optimal Five Filtering Strategies* will take into consideration only twenty, ten, and five most important words, respectively. The models, being created by these optimal strategies, are thus smaller, and consequently can be faster both loaded into memory and compared with a filtering request. As they are omitting many words, they might be at the same time potentially less accurate, and the coordination engine has a chance to decide which one in the given situation can be the best solution.

Since the examination of every single document for each request becomes infeasible even for a collection with the modest size, two different indexing filtering strategies have been also implemented. The first one, named *Inverted List Filtering Strategy*, creates for every word the list of documents having that word. The inbuilt simplification, tending to dramatically reduce a size of inverted lists, is made by not storing the positions of words in the corresponding documents. While a strategy due to such a design decision becomes more efficient, it loses its ability to support requests with a phrase. The second *Position Filtering Strategy* will not utilise such a simplification regarding not storing the positions, and thus will be able to accurately find documents with requested phrases. As this second strategy is naturally more expensive, the trade-off, between providing the accurate results and responding quickly, becomes evident and unavoidable for requests with phrases.

The property of fuzzy clustering [24], to assign documents to multiple clusters together with specifying a degree to which a particular article belongs to a given cluster, has been used as the inspiration for a realisation of a dedicated *Fuzzy Filtering Strategy*. While its strength is in keeping short cluster summaries in the high speed memory, its greatest weakness lies in a used simplification to cluster documents in advance fixed clusters. The few different versions of this fuzzy filtering strategy are finally implemented by limiting the amount of a memory that is utilised for cashing the cluster summaries, having as the implication that different trade-offs between the response time and the memory requirements are possible.

Every mentioned filtering strategy is also exploited for creating its appropriate clone, which will take into account only words from a manually created dictionary. By limiting the vocabulary to few thousands instead to more than half a million, underlying models are much smaller, and thus the underlying strategies become more efficient. Unfortunately, the paid price lies in the lost of a support for all requests with words that are not pre-selected, resulting in the potentially lower quality of found filtering results. These clone strategies finally provide even more fruitful playing ground for both cooperation and coordination mechanisms, which should prove their capabilities while resolving the mentioned trade-off problems.

4.6. Presentation. As one of the main design principles has been to support the user throughout the whole day, the PIA system provides several different access methods and adapts its interfaces to the used device (Figure 4.3). To fulfill these requirements an agent platform (Multi Access Service Platform) was developed that optimizes the graphical user interface for the access by Desktop PCs, PDAs and smart phones.

If the user wants to use the PIA system, the request is received by the Multi Access Service Platform (MASP). The MASP delegates the request to an agent, providing the logic for this service. In the PIA system the requests are forwarded either to login agent or the personal agent. The chosen agent performs the service specific actions and sends the MASP an abstract description of the formular that should be presented to the user. For this purpose the XML based Abstract Interaction Description Language (AIDL) has been defined. Based on the abstract description and the features of the used device the MASP generates an optimized interface presented to the user. The conversion is implemented as a XSLT transformation in which the optimal XSLT style sheet is selected based on the CC/PP information about the user's device.

The Multi Access Service Platform provides a generic infrastructure for providing device optimized interfaces for a big number of devices. The basic idea of MASP is to separate the application logic from the concrete interface design. So the application developer does not have to cope with the specific characteristic of the each relevant device and can concentrate on the application specific data flow and interaction logic.

Fig. 4.3. *Information accessed by browser or tailored for presentation on a PDA or a mobile phone*

For defining the interface of an application the XML based Abstract Interaction Description Language (AIDL) has been defined. The definition of a user interaction (typically one web page) is structures as a tree of predefined user interface elements (e.g. label, input field). An exemplary page description is shown in Program 1.

The abstract interface description can be easily transformed into HTML, PDA optimized HTML or WML. If the user wants to have a voice interface, a style sheet for converting the abstract user interface description into VoiceXML has to be added to the MASP. Additional changes at the application are not needed. In general, the support for new devices can be added without changing or shutting down the application.

Program 1 The abstract interaction description of the PIA login page

```
<?xml version="1.0" encoding="UTF-8"?>
<scenario name="loginPage">
  <input>
    <UIElement>
      <list name="rootNode">
        <UIElement>
          <pageSetting name="user_language">
            <value>de</value>
          </pageSetting>
        </UIElement>
        <UIElement>
           <label name="login__piaLoginQXQ25">
             <value>PIA-Login</value>
           </label>
        </UIElement>
        <UIElement>
          <list name="login__data">
            <UIElement>
              <label name="login__userName">
                <value>Benutzername:</value>
              </label>
            </UIElement>
            <UIElement>
              <fieldValue name="login__userName_default">
                  <value>andreas</value>
               </fieldValue>
            </UIElement>
          </list>
        </UIElement>
         ...
        <serviceLink name="createAccountServiceLink">
          <provider address="tcpip://127.0.0.1:7325" name="PIAgent"/>
          <service name="MAPPresent"/>
          <param name="scenario">createAccount</param>
        </serviceLink>
      </list>
    </UIElement>
  </input>
</scenario>
```

The transformation of the abstract interface description is done using Extensible Stylesheet Language Transformations (XSLT). A XSLT transformation is typically written by a designer who is an expert for creating optimized user interfaces for a device considering the preferences of the respective audience. For simplifying the building of XSLT transformations, the MASP provides a set of generic rules for transforming the frequent elements of the abstract user interface descriptions into basic HTML or WML. Based on these rules more complex and device optimized XSLT transformations can be defined.

An important feature of the utilised MASP is the support of Composite Capability/Preference Profiles (CC/PP). Considering the specific features and properties (e.g. screen size, supported css version, supported image formats) the user interface designer can optimize the interfaces to the properties of the respective device. For converting media data into a device adequate format, the MASP provides a component for scaling and converting images and videos.

The components and interfaces of the Multi-Access-Service Platform are shown in Figure 4.4. Users who want to use the PIA service interact with the Multi Access Point. The MAP contains components for interaction with the respective device (e.g. web server or voice server) and components for rendering the application interface optimized for supported devices. Approved rendering components for HTML, WML and VoiceXML based user interfaces exists; components for applet based components are under development. For the device independent interface description the MASP uses the Abstract Interface Description Language (AIDL) that is use as interface between interface designer and application developer. The bridge between the application and the Abstract Interface Description is provided by the Alter Ego Agent that contains the interaction description

and specific representation rules. Additionally the Media Cache component provides the media content as well as connectivity to external media providers.

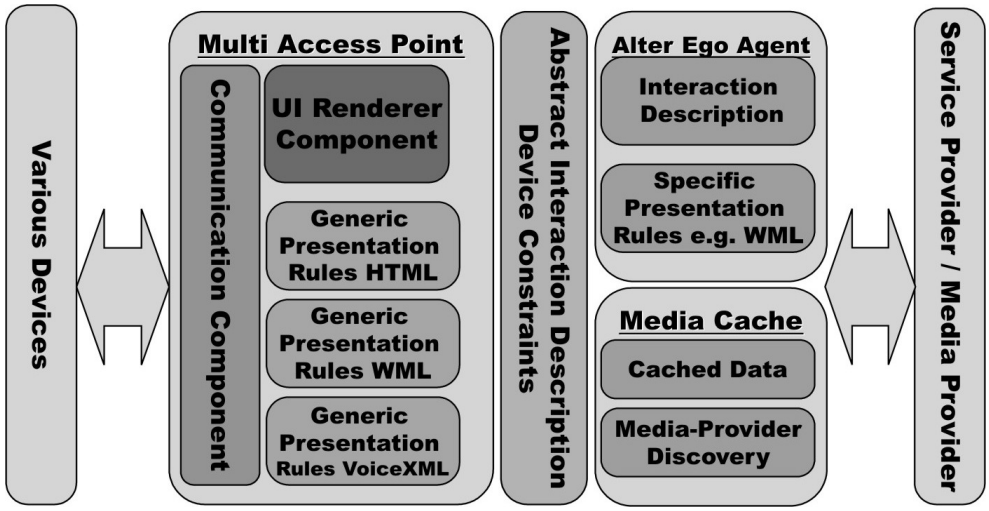


FIG. 4.4. *The architecture of the MASP*

Beside of the features provided by MASP the design of the user interface must create an easy to use system even on devices with a tiny screen and without a keyboard. That is why the PIA interface provides additional navigation elements on complex forms and minimizes the use of text input fields. New results matching a defined request are presented first as a list of short blocks containing only title, abstract and some meta-information (as this is familiar to every user from well-known search-engines). This information is also well readable on PDAs or even mobile phones. Important articles can be stored in a repository. This allows the user to choose the articles on his PDA he wants to read later at his desktop PC.

Depending on the personal options specified by the user, old information found for a specific query may be deleted automatically step by step after a given time, that is, there is always up to date information that is presented to the user (we call this *smart mode*). This is for example convenient for getting personalized filtering news. The other option is to keep that information unlimited (*global mode*) for a query for e.g. basic scientific papers.

For the *push-services* (that is, the system is becoming active and sending the user information without an explicit request), the user specifies his working time (e.g. 9 am to 5 pm). This divides the day in a pre-, work, and a recreation slot, where the PIA system assumes different demands of information. For each slot an adequate delivering technology can be chosen (e-mail, sms, mms, fax or Voice). If you decide to subscribe to the personalized newsletter, new relevant information for you will be collected and sent by e-mail or fax once a day with a similar layout and structure for convenient reading if you have not seen it already by other pull- or push services. Therefore you can also stay informed without having to log into the system and if you do not want to get all new information immediately.

5. Conclusion and future work. The implemented system has an acceptable runtime performance and shows that it is a good choice to develop a personal information system using agent-technology based on a solid agent-framework like JIAC IV. Currently, PIA system supports more than 120 different web sources, grabs daily around 3.000 new semi-structured and unstructured documents, has almost 500.000 already pre-processed articles, and actively helps about fifty scientists related to our laboratory in their information retrieval activities. Their feedback and evaluation is a valuable input for the further improvement of PIA. In the near future we plan to increase the number of users to thousands, and therefore we plan to work on the further optimization of the filtering algorithms to be able to simultaneously respond to multiple filtering requests. Also, we think about integrating additional services for the user that provide information tailored to his geographical position (GPS), a natural speech interface and innovative ways to motivate the user to give precise explicit feedback, as the learning ability of the system is depending on that information.

REFERENCES

[1] S. Brin and L. Page, *The anatomy of a large-scale hyper textual (Web) search engine*, in Proc. 7th International World Wide Web Conference on Computer Networks, 30(1-7), 1998, pp. 107–117.

[2] M. Balabanovic and S. Yoav, *FAB: Content-Based Collaborative Recommendation*, Communication of the ACM, 40(3), 1997, pp. 66–72.

[3] A. Moukas, *Amalthaea: Information Discovery and Filtering using a Multi agent Evolving Ecosystem*, in Practical Application of Intelligent Agents & Multi-Agent Technology, London 1998.

[4] B. Zhang, and Y. Seo, *Personalized Web-Document Filtering Using Reinforcement Learning*, Applied Artificial Intelligence, 15(7), 2001, pp. 665–685.

[5] M. Claypool and A. Gokhale and T. Miranda and P. Murnikov and D. Netes and N. Sartin, *Combining Content-Based and Collaborative Filters in an Online Newspaper*, in Proc. ACM SIGIR Workshop on Recommender Systems, Berkeley, CA, August 19, 1999.

[6] J. Delgado and R. Davidson, *Knowledge bases and user profiling in travel and hospitality recommender systems*, in Proc. of the ENTER 2002 Conference, Innsbruck, Austria, 2002, pp. 1–16.

[7] D. Kuropka and T. Serries, *Personal Information Agent*, in Proc. Informatik Jahrestagung 2001, pp. 940–946.

[8] H. Lieberman, *Letizia: An Agent That Assists Web Browsing*, in Proc. International Joint Conference on Artificial Intelligence, Montreal, August 1995.

[9] H. Lieberman and N. Van Dyke and A. Vivacqua, *Let's Browse: A Collaborative Browsing Agent*, Knowledge-Based Systems, 12(8), 1999, pp. 427–431.

[10] B. Sheth, *A Learning Approach to Personalized Information Filtering*, M.Sc. Thesis, MIT dept, USA, 1994.

[11] T. Joachims and D. Freitag and T. Mitchell, *WebWatcher: A Tour Guide for the World Wide Web*, in Proc. IJCAI (1), 1997, pp. 770–777.

[12] W. Winiwarter, *PEA—A Personal Email Assistant with Evolutionary Adaptation*, International Journal of Information Technology, 5(1), 1999.

[13] C. Thomas and G. Fischer, *Using agents to improve the usability and usefulness of the world wide web*, in Proc. 5th International Conference on User Modelling, 1996, pp. 5–12.

[14] N. Jennings and M. Wooldridge, *Agent-oriented software engineering*, Handbook of Agent Technology (ed. J. Bradshaw), AAAI/MIT Press, 2000.

[15] R. Sesseler and S. Albayrak, *Agent-based Marketplaces for Electronic Commerce*, in Proc. International Conference on Artificial Intelligence, IC-AI 2001.

[16] P. Resnick and J. Neophytos and M. Suchak and P. Bergstrom and J. Riedl, *GroupLens: An open architecture for collaborative filtering of net news*, in Proc. ACM Conference on Computer-Supported Cooperative Work, 1994, pp. 175–186.

[17] S. Albayrak and D. Milosevic, *Self Improving Coordination in Multi Agent Filtering Framework*, in Proc. IEEE/WIC/ACM International Joint Conference on Intelligent Agent technology (IAT 04) and Web Intelligence (WI 04), Beijing, China, September 2004.

[18] D. Nichols, *Implicit Rating and Filtering*, in Proc. 5th DELOS Workshop on Filtering and Collaborative Filtering, Budapest, Hungary, 1997, pp. 31–36.

[19] D. Tauritz, *Adaptive Information Filtering: concepts and algorithms*, Ph.D. dissertation, Leiden University, 2002.

[20] S. Fricke and K. Bsufka and J. Keiser and T. Schmidt and R. Sesseler and S. Albayrak, *Agent-based Telematic Services and Telecom Applications*, Communications of the ACM, 2001.

[21] *Foundation for Intelligent Physical Agents*, `www.fipa.org` 2004.

[22] L. Jing and H. Huang and H. Shi, *Improved Feature Selection Approach TFIDF in Text Mining*, in Proc. 1st International Conference on Machine Learning and Cybernetics, Beijing, 2002.

[23] S. Albayrak and D. Milosevic, *Situation-aware Coordination in Multi Agent Filtering Framework*, in Proc. 19th International Symposium on Computer and Information Sciences (ISCIS 04), Antalya, Turkey, 2004.

[24] M. Zbigniew and D. Fogel, *How to Solve It: Modern Heuristics*, Springer-Verlag New York, Inc., New York, NY, 2000.

[25] S. Albayrak and D. Milosevic, *Cooperative Community Selection in Multi Agent Filtering Framework*, in Proc. IEEE/WIC/ACM International Joint Conference on Intelligent Agent technology (IAT 05) and Web Intelligence (WI 05), Compiegne University of Technology, France, 2005, pp. 527–535.

Edited by: Marcin Paprzycki, Niranjan Suri
Received: October 1, 2006
Accepted: December 10, 2006

Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 41–48. http://www.scpe.org
ISSN 1895-1767

COMPUTATIONALLY ADJUSTABLE AUTONOMY

HENRY HEXMOOR* AND BRIAN MCLAUGHLAN*

Abstract. Reasoning about autonomy is an integral component of collaboration among computational units of distributed systems. This paper introduces an agent-level algorithm that allows an agent to continuously update its autonomy with respect to recurring asynchronous problems with the aim of system-wide collaboration efficiency. The algorithm is demonstrated in a relevant scenario involving NASA space station-based Personal Satellite Assistants, which can handle dynamic situation management that frustrates global collaboration protocols.

Key words. Agents, Autonomy, portable satellite assistant.

1. Introduction. Computer-controlled systems feature prominently in large-scale projects currently under development by the military, commercial, and scientific agencies. Examples of these projects include the US military's Network-Centric Warfare doctrine, IBM's Autonomic Computing initiative, and NASA's space station project. As these systems have increased in complexity, self-governing components have come to feature prominently in their design and control. This change in paradigm from direct human control to indirect human oversight has forced designers to address issues involving the autonomy of these sub-systems.

Autonomy is defined and used in multi-agent system research [6, 7, 11, 12, 13] and other disciplines including sociology [10] and philosophy [14, 15]. It is important in multiagent interactions since it relates the abilities of an agent to its freedoms and choices. The understanding and quantification of an agent's autonomy is required for coherent interagent interaction.

The concept of autonomy is closely related to the concepts of power, control, and dependence [5, 7]. The notion of autonomy has been used in a variety of senses and has been studied in different contexts. It generally presupposes some independence or restricted dependence. However, it can describe many different but related concepts. An agent can be autonomous with respect to another agent if it is beyond the influences of control and power of that agent. It can also be used to describe quality of choice and can even encompass self-imposed "sense of duty" concepts.

While autonomy can be intuitively understood, it unfortunately is a complex topic whose exact definition and implementation is rather elusive. However, by identifying "types" or "subclasses" of autonomy, specific aspects of the concept can be defined and quantified. The multiagent system designer can then utilize these models to focus on the particular attributes of autonomy that would be most beneficial for the particular implementation.

Autonomy is defined in [6] as the agent's degree to which its decisions depend on external sources including other agents. This form of autonomy can be called Cognitive Autonomy. This concept has been explored further in [7]. This paper utilizes this definition of autonomy and promotes the relativistic view introduced in [3, 4].

Adjustable autonomy is a related notion that captures the idea of a human operator intervening and guiding actions of a machine [8]. Another example of the work on adjustable autonomy is [1] with quantitative measure proposed in [2]. In this, the degree of autonomy is defined as an agent's relative voting weight in decision-making. This approach has several advantages including the allowance for explicit representation and adjustment of agent autonomy.

The remainder of this paper presents our work regarding computation and determination of adjustable autonomy levels for collaborative, problem-solving agents in a multi-agent system. Section Two describes our approach, including the generalized algorithm. Section Three portrays an implementation of this algorithm for NASA's PSA program. Experiments performed on this system are chronicled in Section Four. Section Five presents the conclusions drawn from this work.

2. Approach. This paper addresses adjustable autonomy in a distributed system where agents discover, announce, and complete asynchronously occurring tasks. The tasks are generic and require multiple participant collaboration to solve. The collaboration process is facilitated through a four-stage bidding process:

1. Announcement
2. Priority

*Southern Illinois University Carbondale, Illinois, 62901 {hexmoor, brianm}@cs.siu.edu

3. Permission
4. Acceptance

In addition to providing a mechanism for collaboration on tasks, the algorithm must be able to scale well and handle dynamic and complex situations. That is, it must be able to handle multiple, conflicting tasks. It must be able to handle changes to the problem topology such the introduction or removal of key agents or tasks. Ideally, the algorithm will handle variations without excessive setback in its ongoing computations.

Announcement

Upon discovery of a new task, the discovering agent—known here as the originating agent—broadcasts the discovery to the group. Each agent maintains a list of announced tasks. The task data structure is shown in Figure 2.1.

An agent will update the information about a task as it receives relevant information. For simplification, this paper assumes that all agents have some method of hearing announcements and other bidding related information, whether through direct or indirect means. If this simplification is not the case, the algorithm will yield as best a solution as is possible with the information available.

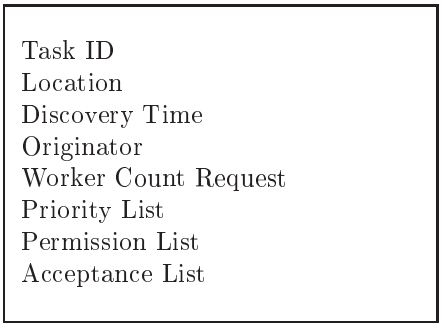


FIG. 2.1. *Task Data*

Priority

Upon receiving and archiving the task announcement, an agent will reason about its objective suitability to address the task. The agent may include several attributes, e.g., necessary skills, energy usage, and the time that the task has been active. It incorporates these factors in assigning some meaningful priority to the task. It is important to note that, at this stage, the agent will not account for alternative tasks. That is, it will not rank a task higher or lower according to its personal preferences. Reasoning along subjective considerations will occur later. Upon determining its priority for the task, the agent will announce the score to the other agents. In the most basic version of this system, only the originator needs to maintain all the priorities. However, as will be described later, some enhancements are possible in which agents can adjust their acceptance based on the priority scores made by other agents.

Permission

The originating agent collects these priority scores and generates a permission list. In its simplest form, the permission list is an ordered list of the priority scores. However, the algorithm utilized by the originating agent can be much more complex, taking into account abstract concepts such as trust and affinity the originating agent has towards particular agents or even known synergies among bidding agents. Ultimately, this permission list contains the bidding agents in the order of most to least desirable for joining the task. Although the originating agent only needs a specific number of agents to perform the task, it will create an ordered list containing all bidding agents in the event that some of the most desirable agents will be unable or unwilling to participate. The originating agent publishes this list to the group.

Acceptance

Unlike many contemporary systems such as online auctions, a bid does not constitute a contract in this system. Each agent is allowed to tentatively accept or reject the permission granted by the originating agent. Additionally, a tentative acceptance is not enforceable. If an agent finds a task for which it is more suitable, it is free to abandon its current task. As will be shown later, it is assumed that the agent has taken into account any disruption its action would make on its current task if it were to act. Thus, the acceptance becomes an announcement of which task the agent is currently considering to perform.

The bidding agent makes her acceptance determination by accounting for several factors including its desire or suitability for this relative to other tasks, the level of permission granted by the originating agent for this and other tasks, and the priority of alternative agents should the agent decline to perform the task.

The bidding agent takes into account competing tasks at this stage rather than in the priority stage so that it can provide benevolence for the system. For example consider an agent X that has placed bids on two tasks, Task 1 that has been announced by agent A and Task 2 that has been announced by agent B. Agent X determines its priority for Task 1 to be quite low, but sees its priority for Task 2 to be high. Both agents A and B have published permission lists in which agent X is among the top choices. If agent X were to take a greedy stance, it would accept the task for which it gave the highest priority, in this case Task 2. However, if it further inspects the permission lists, it may discover that the agents that would be forced to perform Task 1 in agent X's absence are not particularly well-suited for the task and would struggle, while the alternative agents for Task 2 are only slightly less-suitable than agent X and could still perform adequately. To provide for optimal system performance, agent X could choose to accept Task 1 even though it would personally prefer Task 2.

There are three caveats to accepting tasks. First, an agent may only give its acceptance to one task. If it has already accepted a previous task, it must announce its withdrawal from that previous task.

Second, an agent cannot accept a task that has been locked. A task is locked if n higher-ranked agents have accepted the task, where n is the requested number of agents for the task[1].

Third, an agent cannot accept a task where it is not ranked in the first n non-rejecting agents in the permission list where n is the number of agents required to perform the task. That is, if a task needs three agents, and agent X is ranked fourth, it cannot accept the task unless one of the first three decline it. Conversely, any agent may decline a task regardless of its ranking in the permission list. These scenarios are shown in Figure 2.2.

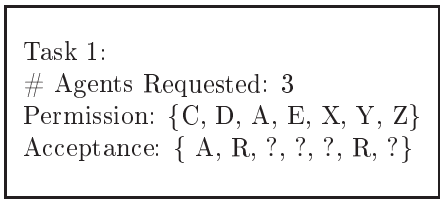


FIG. 2.2. *Agent X cannot accept the task until either agent A or E rejects it.*

Algorithm

An algorithm has been developed to facilitate this bidding scheme. This algorithm is implemented at the agent level and runs continuously. The pseudo code for this algorithm is shown in Algorithm 1.

Some notes regarding this algorithm. In the final If statement, the agent does nothing if its chosen task could be filled by more qualified agents. This forces the agent to wait to see if the desired task will become available. As an alternative, the agent could change this to a rejection and recalculate a "second best choice". Then, if the desired task becomes available due to top-ranked agents rejections, it can change its acceptance back to the original task. This alternative keeps all agents busy, but it may cause additional start-up costs from changing tasks

It is the task originator's responsibility to ensure that the task does not get lost in the shuffle. To this end, the originating agent will periodically broadcast the current state of the task.

Rather than rigidly define the four phases of the bidding process, the algorithm allows each agent to proceed independently. This precludes the need for coordination of phase changes that may be difficult in some environments. However, this could cause the originating agent to publish a permission list before all agents have given their priority scores. With the publication of this list, the agents are free to begin the acceptance process before potentially ideal agents announce their priority. To prevent unnecessary shuffling as new agents bump out less ideal workers, the agents should take potential shuffling into account when bidding. Alternatively, if the agents can communicate with all other agents in the system, then the originating agent can delay publishing the permission list until all agents have announced their priorities.

To illustrate the algorithm, consider the following scenario. To simplify the illustration, the scenario will be shown from the perspective of the tasks.

[1]In the PSA application, "n" is three. I.e., three robots are required to triangulate source of the problem.

Algorithm 1 Bidding Scheme Pseudocode

```
while 1 do
  Sense surroundings
  Task List update
  Append new discovered tasks
  Append new heard tasks
  Update existing tasks
  for Each task t in Task List do
    Calculate and announce t_priority
    if t_originator == self then
      Calculate t_permission List
      Announce task t
    end if
  end for
  Calculate best non-locked task
  for Each task t in Task List do
    if t ≠ best then
      Announce rejection
    else
      if Self rank < n^th non-rejecting then
        Announce acceptance
      else
        Do nothing
      end if
    end if
  end for
end while
```

Agents A and B have discovered and announced Tasks 1 and 2, respectively. Agents A, B, C, and D are within responding distance to these tasks. Figure 2.3 shows the state of the tasks after the agents have begun to respond with their priority to the tasks and the originating agents have published permission lists. For simplicity, permission is granted based solely on announced priority.

Task 1	Task 2
# Agents Needed: 2	# Agents Needed: 2
Priority: {A,7},{B,3}, {C,8},{D,9}	Priority: {A,1},{B,8}, {C,2},{D,7}
Permission: D, C, A, B	Permission: B, D, C, A
Acceptance: ?, ?, ?, ?	Acceptance: ?, ?, ?, ?

FIG. 2.3. *Permission List Publication*

With the publishing of the permission lists, agents are now free to begin accepting or rejecting the tasks as shown in Figure 2.4. Agents C and D are the most ideal candidates for Task 1. C will accept this task as B accepts Task 2. They will quickly reject the alternate tasks.

However, D has been accepted for both tasks. Greedily, it could accept Task 1, but its rejection of Task 2 would force Task 2 to be performed by A, a very unsuitable agent. It must decide on a course of action—greedy or benevolent.

Agent A cannot announce its acceptance of Task 1 despite its likely preference toward it. Rather, it will wait to see what Agent D announces so that it will not have to begin its inept performance of Task 2 and then possibly switch mid-execution to Task 1.

Next, consider how the algorithm will react to a dynamic situation. For this we introduce another agent, agent E. This agent hears the updates given by the two task originators and determines its priority for the tasks. Additionally, agent C detects a new task, Task 3. This situation is shown in Figure 2.5.

Task 1	Task 2
# Agents Needed: 2	# Agents Needed: 2
Priority: {A,7},{B,3}, {C,8},{D,9}	Priority: {A,1},{B,8}, {C,2},{D,7}
Permission: D, C, A, B	Permission: B, D, C, A
Acceptance: ?, A, ?, R	Acceptance: A, ?, R, ?

FIG. 2.4. *Partially Accepted*

Task 1	Task 2	Task 3
# Agents Needed: 2	# Agents Needed: 2	# Agents Needed: 2
Priority: {A,7},{B,3},	Priority: {A,1},{B,8},	Priority: {A,7},{B,3},
{C,8},{D,9}, {E,0}	{C,2},{D,7}, {E,5}	{C,9},{D,8}, {E,5}
Permission: D, C, A, B, E	Permission: B, D, E, C, A	Permission: C, D, A, E, B
Acceptance: ?, R, ?, R, R	Acceptance: A, ?, ?, R, ?	Acceptance: C, ?, ?, ?, ?

FIG. 2.5. *New Task and Agent*

In this situation, C chooses its own task and rejects its previous acceptance of Task 1. Additionally, E immediately sends rejection to Task 1 due to its absolute inability to perform the task as demonstrated from its priority announcement of 0.

This leaves several issues to be resolved. First, it allows A to accept its ideal Task 1 as it is now in the first 2 non-rejecting agents and does not need to wait for D's rejection.

Agent D is now desired by all three tasks. It still has some determination to make before choosing. For instance, D's choice could depend upon whether agent A chooses Task 1 or Task 3. It also depends upon whether important tasks will be left without adequate workers.

The exact method utilized for determining its choice depends on how much complexity the system designer imbues in the agents' decision-making process. Ideal efficiency is a difficult problem that is most likely beyond the practical scope of real-world agents regardless of the algorithm. However, the agent could play the prisoner's dilemma game to second-guess what other agents may choose. Perhaps the simplest and most computationally efficient method when faced with such incomplete information would be for Agent D to take the greedy choice and let the other agents adjust to maximize the remaining system performance.

Additionally, this example illustrates a problem with all task allocation algorithms—maximizing utility when not enough workers are present. If such a scenario is likely in the system, the designer could include a task priority that would modify the agents' behavior such that they would be more likely to accept critical tasks and leave less vital tasks understaffed.

Despite the problems, this example demonstrates how the algorithm can adapt to changes made mid-calculation. Rather than toss out the bidding process and start over or exclude new agents and tasks from the proceedings, the agents make some quick adjustments and continue.

3. An application: The Personal Satellite Assistant (PSA). A PSA is a small (basketball-sized) flying robot that is under development at NASA Ames (at the Moffet field AFB[2]) for deployment on the international space station. These robots are an outgrowth of a need to free astronauts from routine tasks of inventory control, safety checks, and fault detection and isolation. PSAs are loaded with a variety of sensors including equipment for gas and pressure sensing. In the remainder of this section we describe an implementation of our algorithm that allows PSAs to perform several appropriate tasks such as fire and gas leak (i. e., on- and off-gassing) detection while reasoning about their autonomy and level of collaboration.

As per the algorithm, the PSA that detects the problem formulates a broadcast alert to send to the other PSAs. This is initiated when a PSA locates an abnormality in its environment. The abnormality could be a variation in the ambient temperature or an atmospheric imbalance such as high or low pressure, or excess oxygen, carbon dioxide, or nitrogen. The PSA sends the alert containing the type of problem and type of room in which the problem is located to persuade other agents to help it pinpoint the source of the problem more accurately. This process is similar to the method used in radio signal triangulation.

[2]We thank Yuri Gawdiak for a tour and discussions in 2002.

To determine its suitability for this task, the PSA must account for its energy resources. Each PSA has a limited battery power that will be consumed during transit as well as during the task execution. It is assumed that the PSA has a means of evaluating its resources R, which in this case is its battery charge. It will then calculate its cost C to perform the task.

C is initially computed by calculating the distance to travel to the task and the subsequent distance to a power recharge station. It does the system little good for a PSA to assist in locating a problem only to run out of energy and shut down. The total distance to be moved is multiplied by the energy consumption rate. An estimation of the amount of energy required to perform the task is added to get the total cost C.

$$\begin{aligned} \mathrm{C} =& (\text{Distance to target} + \text{Distance from target to recharge}) \times \text{Energy Consumption Rate} \\ &+ \text{Energy required for task} \end{aligned}$$

If $\mathrm{C} > \mathrm{R}$ then an unfavorable priority is return indicating unavailability. Otherwise, when $\mathrm{C} \leq \mathrm{R}$, the PSA can successfully help locate the problem and still recharge itself. In this case, priority P is calculated by first considering what type of room in which the problem is located. This is done since some locations are inherently more important than others. For instance, laboratories are relatively less important than the control center. Additionally, the particular anomaly detected can influence the priority for a particular room. For example, off-gassing of oxygen in a equipment storage module would be less disastrous than the same problem in a habitation module. Conversely, high levels of magnetic interference may be dangerous for the equipment but could be of little consequence to humans inhabiting their quarters. The determined value, which we denote as Q, is used for calculating the job weight and is used in the final priority calculation for P.

$$\mathrm{Q} = \ln(\text{Time} + \text{RoomProblemFactor})$$

The natural log is used for this equation because it causes Q to change along a predictable curve as either Time or RoomProblemFactor increases.

P is computed by using distance as a scalar and comparing the new job weight to the old job weight.

$$\mathrm{P} = \mathrm{Q}_{\mathrm{new}} \times \left(1 - \frac{\text{Distance to new target}}{\text{MAXDISTANCE}}\right) - \mathrm{Q}_{\mathrm{old}} \times \left(1 - \frac{\text{Distance remaining to old target}}{\text{MAXDISTANCE}}\right)$$

MAXDISTANCE is the maximum distance a PSA can move through the entire station. The distance plays an important role in the calculation of P. This is due to the observation that the PSA with the smallest distance to move will be the most likely to arrive quickest. Thus, the time to complete the task is lower with this PSA.

As the PSAs proceed through the bidding process—priority declaration, permission, and acceptance—and the chosen PSAs begin to arrive at the problem location, they will take a prism on the face of the search space and begin scanning. This will allow PSAs that arrive quicker to begin the search process, while PSAs that arrive later can help refine the results. Thus, a measure of completion can be taken at any point in time during the triangulation.

4. Experiments. Experiments were performed utilizing the PSA scenario. The locations of problems and PSAs were arranged such that the system was relatively balanced. The Q value of each problem was randomly generated. The number of PSAs in the system was sufficient in each test to meet the demands.

The exact method of acceptance was performed under two strategies. In strategy 1, agents chose to accept the task in which they were highest ranked for permission. Note that this does not necessarily mean that the PSA greedily chooses the task for which it attributed the highest priority. Rather, it will choose the task of the originator that most values the PSA's assistance. For instance, if a PSA is listed as first in the permission list, it will accept that over a task where it is listed second. In strategy 2, PSAs perform as described in the algorithm—they choose to accept a task such that the sum of all priorities chosen is maximized. This strategy should spread the quality of help across the problems.

The results of the experiments are shown in Figure 4.1 and shows that the two strategies produce very similar results. However, the first strategy gives slightly better performance in this particular simulation and is computationally less intensive in general.

The reason for this decrease in performance lies in the nature of the decision making in the system as a result of the additional process. By decentralizing the decision-making, choices are being made based upon less than the total amount of information in the system.

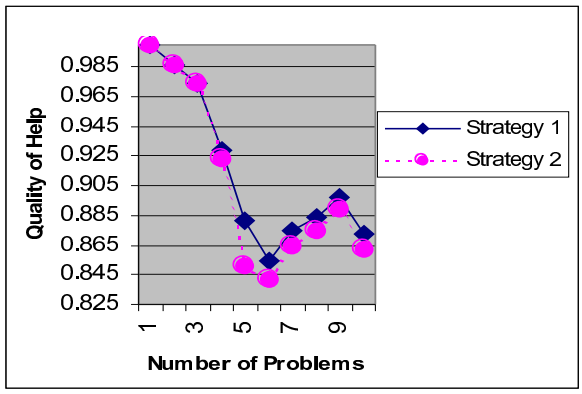


FIG. 4.1. *Quality of help for two strategies*

From the perspective of autonomy, the first strategy restricts the agents to a greater degree. The individual PSAs have less freedom in mobility and choice of tasks. Priority only plays a role in the very first stage of the process. After that, it is up to the originating PSA. The second method allows the bidding PSAs to undertake whichever task is both best fitting to them and compliant to the greater needs of the system.

5. Conclusion. As computer controlled systems increase in complexity, automated collaboration of subsystems becomes more relevant and critical to system efficiency. Utilizing the concept of adjustable autonomy—reasoning about commitments in particular—is a critical component to solving this problem. This work has shown how reasoning about autonomy can form the basis of moment-to-moment commitment making.

We have shown an algorithm that can be utilized for dynamic decision-making that is flexible enough to handle agents that join or leave before tasks are completed, as well as being able to handle tasks that appear during the execution of other tasks.

We have shown how this algorithm can be implemented in a relevant and current problem—that of task management of NASA's Personal Satellite Assistants on board the international space station. The domain of PSAs is a dynamic environment where multiple and possibly concurrent problems may develop, and is an area that will benefit from the teamwork made possible by this algorithm.

Future work in this area can take many directions. For instance, we could consider subjective attributes such as qualities of relationships and satisfaction of agents with the task assignment process. Additionally, we can look at the application of autonomy determination in reasoning about teams [3, 16] and its effect on this algorithm.

REFERENCES

[1] K. S. BARBER AND C. MARTIN, *Agent Autonomy: Specification, Measurement, and Dynamic Adjustment*, In Proceedings of the Autonomy Control Software Workshop, Agents '99, May 1–5, Seattle, WA., 1999, pp. 8–15.

[2] K. S. BARBER, A. GOEL, AND C. MARTIN, *Dynamic Adaptive Autonomy in Multi-Agent Systems*, In Journal of Experimental and Theoretical Artificial Intelligence, 12(2), Taylor and Francis, 2000, pp. 129–147.

[3] G. BEAVERS AND H. HEXMOOR, *Teams of Agents*, In Proceedings of the IEEE Systems, Man, and Cybernetics Conference, IEEE, 2001.

[4] S. BRAINOV, AND H. HEXMOOR, *Quantifying Relative Autonomy in Multiagent Interaction*, In IJCAI-01 Workshop, Autonomy, Delegation, and Control, ACM, 2001.

[5] K. S. BRAINOV AND T. SANDHOLM, *Power, Dependence and Stability in Multiagent Plans.* In Proceedings of AAAI/IAAI 1999, AAAI, 1999, pp. 11–16.

[6] C. CASTELFRANCHI, *Guaranties for Autonomy in Cognitive Agent Architecture*, In N. Jennings and M. Wooldridge, eds., Agent Theories, Architectures, and Languages, Spinger-Verlag, 1995, pp. 56–70.

[7] C. CASTELFRANCHI, *Founding Agent's Autonomy on Dependence Theory*, In proceedings of ECAI'01, Berlin, 2000, pp. 353–357.

[8] G. DORAIS, P. BONASSO, D. KORTENKAMP, B. PELL, AND D. SCHRECKENGHOST, *Adjustable Autonomy for Human-Centered Autonomous Systems on Mars*, Presented at Mars Society Conference, AIAA, 1998.

[9] G. DWORKIN, *The Theory and Practice of Autonomy*, Cambridge University Press, 1988.

[10] H. HEXMOOR, *A Cognitive Model of Situated Autonomy*, In Proceedings of PRICAI-2000 Workshop on Teams with Adjustable Autonomy, Australia, 2000a.

[11] H. HEXMOOR, *Case Studies of Autonomy*, In proceedings of FLAIRS 2000, J. Etherege and B. Manaris, eds., Orlando, FL., AAAI, 2000b, pp. 246–249.

[12] H. Hexmoor, D. Kortenkamp, *Autonomy Control Software*, An introductory article of the special issue of Journal of Experimental and Theoretical Artificial Intelligence, Kluwer, 2000.
[13] S. Irani, S. K. Shukla, R. K. Gupta, *Online strategies for dynamic power management in systems with multiple power-saving states*. ACM Trans. Embedded Comput. Syst. 2(3), ACM, 2003, pp. 325–346.
[14] Mele, *Autonomous Agents: From Self-Control to Autonomy*, Oxford University Press, 1995.
[15] J. Schneewind, *The Invention of Autonomy: A History of Modern Moral Philosophy*, Cambridge Univ. Press, 1997.
[16] M. Tambe, D. Pynadath, C. Chauvat, A. Das, and G. Kaminka, *Adaptive agent architectures for heterogeneous team members*, In Proceedings of the International Conference on Multi-agent Systems (ICMAS 2000), Boston, MA, 2000.

Edited by: Marcin Paprzycki, Niranjan Suri
Received: October 1, 2006
Accepted: December 10, 2006

Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 49–62. http://www.scpe.org
ISSN 1895-1767

A TOP DOWN APPROACH FOR DESCRIBING THE ACQUAINTACE ORGANISATION OF MULTIAGENT SYSTEMS*

JOAQUÍN PEÑA†, RAFAEL CORCHUELO† AND ANTONIO RUIZ-CORTÉS†

Abstract.
When the protocol of a complex Multi-Agent System (MAS) needs to be developed, the *top-down approach* emphasises to start with abstract descriptions that should be refined incrementally until we achieve the detail level necessary to implement it. Unfortunately, there exist a semantic gap in interaction protocol methodologies because most of them first, identify which tasks has to be performed, and then use low level description such as sequences of messages to detail them.

In this paper, we propose an approach to bridge this gap proposing a set of techniques that are integrated in a methodology called MaCMAS (Methodology for Analysing Complex Multiagent Systems). We model MAS protocols using several abstract views of the tasks to be performed, and provide a systematic method to reach message sequences descriptions from task descriptions. These tasks are represented by means of interactions that shall be refined systematically into lower-level interactions with the techniques proposed in this paper (simpler interactions are easier to describe and implement using message passing.) Unfortunately, deadlocks may appear due to protocol design mistakes or due to the refinement process that we present. Thus, we also propose an algorithm to ensure that protocols are deadlock free.

Key words. Top-down approach, agent protocol descriptions, interaction refinements, and deadlock detection.

1. Introduction.

Agent-Oriented Software Engineering (AOSE) is paving the way for a new paradigm in the Software Engineering field. This is the reason why a large amount of research papers on this topic are appearing in the literature. One of the main research lines in AOSE arena is devoted to developing methodologies for modelling interaction protocols (hereafter protocols) between agents.

1.1. Motivation.

When a large system is modeled, its complexity becomes a critical factor that has to be managed properly to achieve clear, readable, reusable, and correct specifications [8, 24, 30]. In the literature, there exist various techniques to palliate this problem. The most important are the *top down* and the *bottom up* approachs. The top down approach, which is the focus of this paper, first tries to describe software from a high level of abstraction, and then goes into further details until they are enough for implementing the system [32].

When the protocol of a large MAS has to be developed, it is desirable to start with an abstract description that can be refined incrementally according to the *top down approach*. In our opinion, there exist two drawbacks in most existing methodologies:

- On the one hand, most of them provide top-down approaches for modeling and developing these systems. These methodologies, general or protocol-centric, agree on using abstract messages and sequence diagrams to describe protocols [3, 19, 37, 15]. Although these messages represent a high level view of a protocol, which shall be refined later, the tasks that are performed are formulated as a set of messages. This representation implies that the abstraction level falls dramatically since a task that is done by more than two agents requires several messages to be represented. This occurs even if we consider a task between two agents. For instance, an information request between two agents must be represented with two messages at least (one to ask, and another to reply). This introduces a semantic gap between tasks to be performed identified at requirements and its internal design since it is difficult to identify the tasks represented in a sequence of messages. This representation becomes an important problem regarding readability and manageability of large MAS.
- On the other hand, abstractions of protocols (interactions) that allow designers to encapsulate pieces of a protocol that is executed by an arbitrary number of agents has been proved adequate in this context [3, 4, 19, 20, 38]. Unfortunately, interactions are generally used to hide unnecessary details about some views of the protocol. This improves readability and promotes reusability of protocol patterns, but they are not used for bridging the existing semantic gap between tasks and its representation.

1.2. Contributions.

In our proposal, we present a different approach to use interactions, which is based on the ideas presented in [4, 26, 38]. This approach is integrated on a methodology called MaCMAS that covers top-down and bottom-up. The top down software process is sketched in Figure 1.1. As shown, our goal is to

*This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project Web-Factories (TIN2006-00472) and TIC200302737C0201
†University of Seville, {joaquinp, corchu}@lsi.us.es, aruiz@us.es

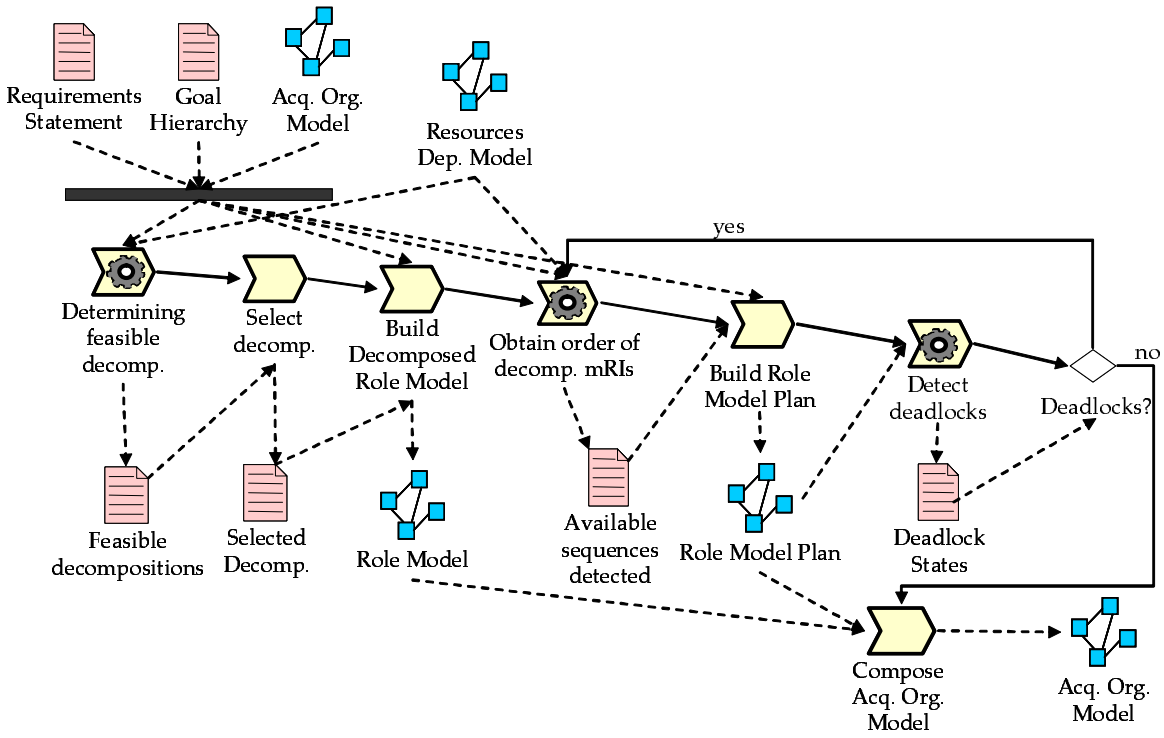


Fig. 1.1. *Software process of refinements.*

bridge this gap using interaction abstractions to model the tasks to be performed, and Finite State Automata (FSA), represented using UML 2.0, to model how to sequence them. Afterwards, we refine them systematically into simpler ones iteratively. This decreases the level of abstraction so that the interaction we obtain are simpler. Thus, they are described internally as message sequences easily, e.g. using AUML [3].

We have used a protocol abstraction called multi-role interaction (mRI), which was first proposed in [25]. An mRI is an abstraction that encapsulates a set of messages between an arbitrary number of agent roles. Furthermore, the refinement process we use is based on the ideas presented in [10] since the interaction we use is similar to such used in this work. The refinement process relies on analysing the knowledge used by each role in an mRI and using this information to transform an mRI into several simpler mRIs automatically. An mRI is simpler when both the number of participant roles and the computation made by it decreases. The main advantages of refining mRIs are the followings:

- First, its internal description is easier since the computation to perform in the obtained tasks are simpler.
- Second, it is easier to implement interactions with a low number of participant roles [12, page 206] [2, 33, 21, 35].
- Finally, mRIs are critical deadlock free regions and they are mutually exclusive. Thus, if the number of participant roles increases, the concurrency grain decreases, what is clearly not desirable [34].

The main drawback of such refinements is that they may lead to deadlocks. In this paper, we also propose a technique to detect if a refinement may introduce deadlocks (see Figure 1.1); it also characterises them by means of regular expressions that help finding the refinements that are not adequate in a given context. It is based on analysing the FSA that represents the protocol of a role model and some previous work on deadlock detection in the context of client/server interactions [5, 14, 36]. It improves on other results in that it can be automated because it does not require any knowledge about the implied, intuitive semantics of the interactions as other approaches.

This paper is organised as follows: in Section 2 we present the related work about protocol modeling in MAS and about interaction refinements; in Section 3, we summarise the methodology where this work is integrated; in Section 4 we present the example that we use to illustrate our approach; in Section 5 we present our ideas on

protocol modeling and we show the refinement techniques applicable; in Section 6 we present our approach to the automatic deadlock detection process; Section 7, we show our main conclusions. Finally, an appendix that shows an implementation of the case study using IP.

2. Related work. In this section we cover the related work on protocol modelling and on refinements.

2.1. Protocol Modeling. As we showed in the previous section, we think that most approaches model protocols at low level of abstraction since they require the designer to model complex cooperations as message-based protocols. This issue has been identified in the Gaia Methodology [38], and also in the work of Caire *et. al.* [4], where the protocol description process starts with a high level view based on describing tasks as complex communication primitives (hereafter interactions). We think that the ideas presented in both papers are adequate for this kind of systems where interactions are more important than in object-oriented programming.

On the one hand, in the Gaia methodology, protocols are modeled using abstract textual templates. Each template represents an interaction or task to be performed between an arbitrary number of participants. Furthermore, interactions are decorated with the knowledge they process and the permissions each role has, their purpose, their inputs and outputs, and so on.

On the other hand, in [4], the authors propose a methodology in which the first protocol view is a static view of the interactions in a system. Each interaction is used by a set of agent roles and they are decorated with the knowledge each role uses/supplies. Later, the internals of these interactions are described using AUML [3].

As the methodologies cited above, we also use interactions to deal with the first stage of protocol modeling. Furthermore, we also represent a static view of interactions and the knowledge that each role consumes and produces in each of them. Unfortunately, both methodologies do not provide an automatic method for refining complex interactions into smaller interactions that are closer to the implementation level. In this paper, we elaborate on such a method.

Furthermore, in methodologies that use sequence diagrams to model protocols, it has been also identified the need for advanced multi-role interactions that encapsulate a piece of protocol. Unfortunately, in most of them these interactions are used to define reusable patterns of interaction or for hiding details in some complex views. Several examples of such use of interactions can be found in the literature: For instance, AUML nested protocols [3] or micro-protocols [19]. These approaches provide the user with a set of tools to model complex co-operations; however, most designers use message–based descriptions.

2.2. Refinements. The need for such protocol primitives has also been identified in other areas such as distributed systems [11, 7, 23]. In this context such interactions have been studied for long, and there exist advanced techniques to refine them (synchrony loosening refinements [10]). Unfortunately, these refinements can lead to deadlock. Although the theory of refinements has reached a rather elaborate state in other contexts, cf. [1], there are not many results on interaction refinements or the characterisation of their anomalies. The main reason is that classical refinements are context-free, whereas interaction refinements are context–sensitive. Thus, the main problem is the establishment of their monotonicity properties [10], whereby their application to subparts of a protocol preserves the correctness of the whole protocol with respect the set of valid synchronisation patterns it describes.

The state–of–the–art technique that focus on design time properties was presented in [12]. It is based on designing a formal proof system (*cooperating proof*) that allows to prove a sufficient condition for monotonicity that ensures that a system composed of interactions is deadlock free. It is based on analysing linked interactions, i.e., interactions that need to be executed in sequence, to avoid deadlocks, which was previously suggested in [9, 18]. Unfortunately, this technique is quite difficult to apply in practice because it requires in-depth knowledge of the implied, intuitive meaning of the interactions, and no automatic proof rules were designed for showing the satisfaction of the sufficient condition.

Our proposal can detect if a refinement may lead to a deadlock situation automatically, and also characterises the set of traces that lead to it by means of regular expressions. It is based on FSA analysis used by many researchers in the context of client/server deadlock detection of interaction models [5, 14, 36].

3. Engineering MultiAgent Systems with MaCMAS. MaCMAs[1] is a methodology for engineering complex multiagent systems that is integrated with several research fields, i.e. autonomic computing [31], software product lines [27, 28] and evolving systems [29].

[1] see `james.eii.us.es/MaCMAS/` for further details on MaCMAS

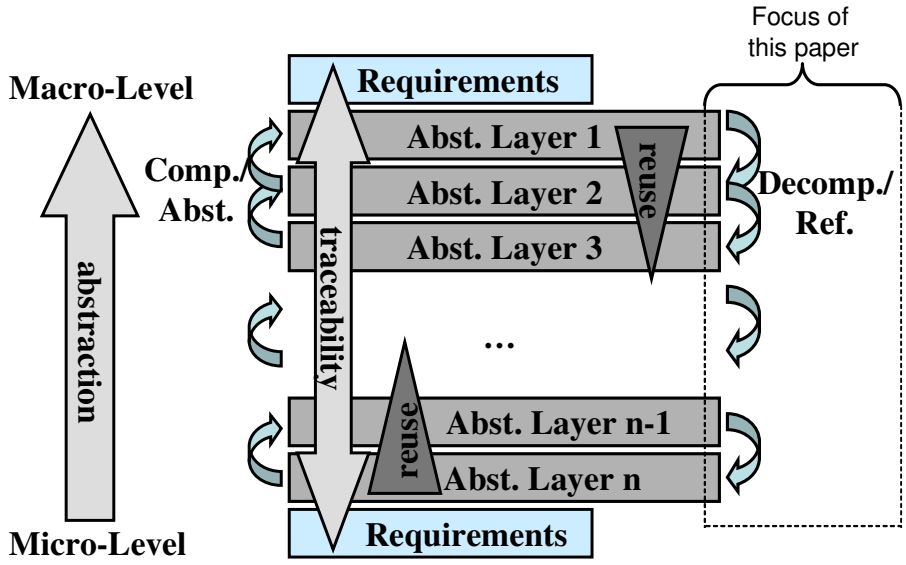


FIG. 3.1. *Process Overview*

MaCMAS covers carefully the five principles to deal with complexity in software engineering where top-down and bottom-up are of high importance [16, 17, 30]: abstraction, decomposition/refinements, composition/abstraction, automation and reuse.

In Figure 3.1, we show an overview of the main concepts applied in MaCMAS from the software process point of view. As shown, models of the system are structured into a set of abstraction layers. Top models are the most abstract while bottom models are the most refined models. MaCMAS provides also a set of vertical and horizontal transformations. Vertical transformations are applied to split models or to compose models, and horizontal transformations are used to refine and abstract models in order to cover bottom-up and top-down software processes.

As shown, for covering the rest of principles, traceability between models at different abstraction layers and reuse of models and their abstractions/refinements is also provided.

In MaCMAS, two kind of refinements are proposed. One that is base on analyzing information on requirement documents, concretely system goals hierarchies, to recommend the user of the CASE tool which models can be refined and which is the best decomposition recommended. The other refinement, which is the focus of this paper, is based on analyzing the dependencies between the elements in a model to recommend a refinement.

3.1. Models. In other to engineer MASs, MaCMAS provides a rich set of UML2.0-based models that can be summarized in:

a) Static Acquaintance Organization View: This shows the static interaction relationships between roles in the system and the knowledge processed by them. It comprises the following UML models:

Role Models: shows an acquaintance sub-organization as a set of roles collaborating by means of several mRIs. As mRIs allow abstract representation of interactions, we can use these models at whatever level of abstraction we desire. We use role models to represent autonomous and autonomic properties of the system at the level of abstraction we need.

Parameterized Role Models : A parameterised role model permits us to represent reusable collaboration patterns parameterising some of their elements.

Resources dependency model: A resources dependency model provides means for documenting the dependencies between knowledge entities and services provided by roles in the context of an mRI and for documenting the dependencies between the knowledge of mRIs.

Relating role models model: As a result of using decomposition and composition and of instantiating parameterised role models, we usually manage role models that are obtained from others. This model show the relationships between several role models.

Ontology: shows the ontology shared by roles in a role model. It is used to add semantics to the knowledge owned and exchanged by roles.

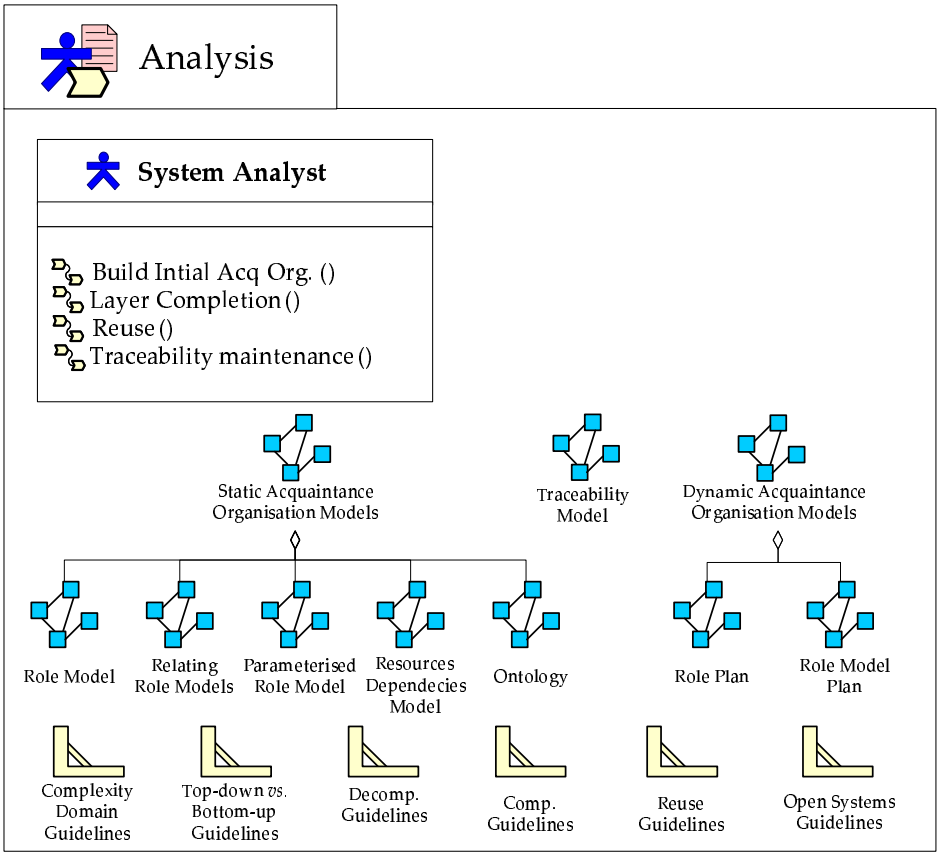


FIG. 3.2. *Acquaintance analysis discipline*

b) Behavior of Acquaintance Organization View: The behavioral aspect of an organization shows the sequencing of mRIs in a particular role model. It is represented by two equivalent models:

Plan of a role: separately represents the plan of each role in a role model showing how the mRIs of the role sequence. It is represented using UML 2.0 ProtocolStateMachines [22, p. 422]. It is used to focus on a certain role, while ignoring others.

Plan of a role model: represents the order of mRIs in a role model with a centralized description. It is represented using UML 2.0 StateMachines [22, p. 446]. It is used to facilitate easy understanding of the whole behavior of a sub-organization.

c) Traceability view: This model shows how models in different abstraction layers relate. It shows how mRIs are abstracted, composed or decomposed by means of *classification*, *aggregation*, *generalization* or *redefinition*. Notice that we usually show only the relations between interactions because they are the focus of modeling, but all the elements that compose an mRI can also be related. Finally, since an mRI presents a direct correlation with system goals, traceability models clearly show how a certain requirement system goal is refined and materialized. This is main what helps us to bridge the gap between requirements and design.

For the purpose of this paper, we only need to detail role models, role model plans, which are shown in the following sections.

4. The Example. The example we use hereafter is a debit–card system. This problem can be viewed as one of the basic coordination patterns in the agent e-commerce world, and it involves three different agent roles (hereafter roles): a point of sales role (PS) which interacts with the user, a customer account manager role(CA), and a merchant account manager role (MA). When a customer uses his or her debit card, the agent playing role PS agrees with a CA agent and merchant account agent on performing a sequence of tasks to transfer the money from the customer account to the merchant account, which shall also be charged the costs of the transaction. If

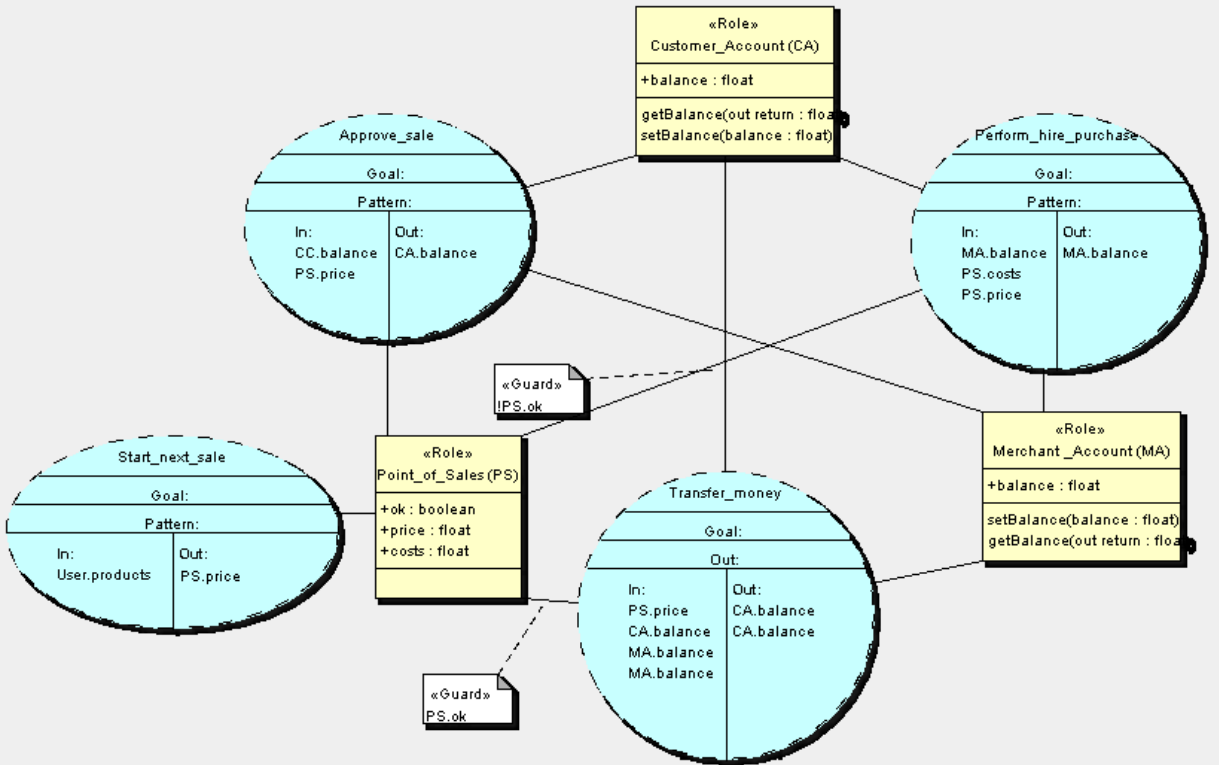


Fig. 5.1. *Static interaction view of the debit–card system.*

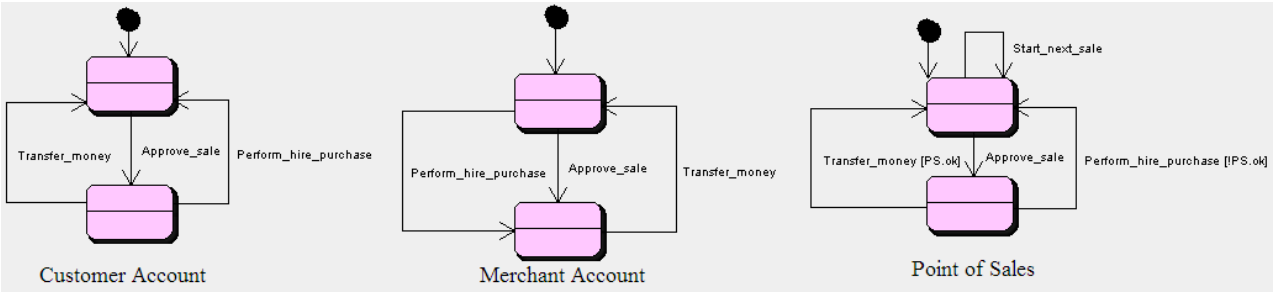


Fig. 5.2. *Plans of the roles in the debit–card system.*

the customer account cannot afford the purchase because it has not enough money, the customer account agent then pays on hire–purchase.

5. Modeling the Protocol with MaCMAS. As we showed above, our approach starts when the requirements system goals to be performed have been already obtained. Then, we model each task as an mRI as we show in the role model in Figure 5.1.

These system goals in our example are modeled as the following mRIs: *approv* is used by the CA role to inform the other parties if it can afford a purchase; *transfer* is used to transfer money from the CA to the MA by means of the PS; mRI *hire_p* is used to buy on hire-purchase; finally, there is a two-party mRI called *next_sale*, which is not further detailed, whose goal is to encapsulate the operations needed to read the sum to be transferred and the customer data from his or her debit card. For further details on the knowledge processed by each participants and in the mRI see the Appendix.

Once the mRIs are identified and linked with their participant roles, we represent their possible sequences by means of FSAs (see Figure 5.2). When an mRI is executed by more than one role it must appear a transition in all the roles that perform it. Each of these transitions represents the part of the mRIs that a role perform. Whereby,

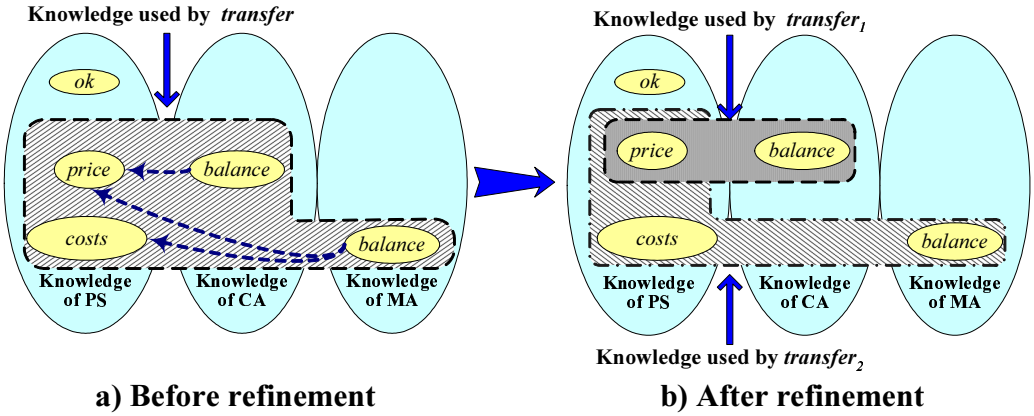


FIG. 5.3. *Decoupling mRI transfer.*

to execute an mRI we must transit from one state to another in all the roles that participate on it. Furthermore, with the algorithms presented in [25], which we outline in section 6, we can automatically infer a single FSA that represents the role model protocol as a whole. This alternative representation can be used for better readability.

Finally, each mRI have to be decorated with some additional information: such as the dependencies between they knowledge it process, a guard for each role, and so on. The knowledge dependency, as we show in the next section, can be analysed in order to refine mRIs. Furthermore, the guard of mRIs allows each role to decide if it want to execute the mRI or not, which has been proved adequate to deal with proactivity of agents [7, 19, 25].

5.1. Refinements. The model we presented in previous section takes advantage of complex three–party mRIs, which provides a high level design of the protocol. However, it should be refined in an attempt to transform its mRIs into a set of simpler ones that are closer to message sequences description. That is to say, describing them internally shall be easier. This is the next step in our approach.

The refinements are based on analysing the dependencies between the knowledge that roles use from others in a particular mRI. In order to automate the refinement process the designer has to build a dependency graph (see Figures 5.3, 5.4 and 5.5) which shall be analysed with the algorithms proposed in [18, 10]. To illustrate how our technique works we applied it to our example.

The first refinement we can apply is *decoupling* [12]. It can transform certain n–party mRIs into an m–party mRI ($m < n$) followed by an mRI with $n - m + 1$ participants. We can illustrate it by means of mRI $transfer$ in our example. Figure 5.3 shows a diagram in which we have depicted the knowledge of its roles and their dependencies. As shown, both the MA and CA need to update their balances according to some information in the knowledge of the PS. The idea is thus to decouple mRI $transfer$ into two binary mRIs so that the CA updates its balance before the MA. Thus, as we can see in Figure 5.3 mRI $transfer_1$ will executed by PS and CA, and $transfer_2$ by PS and MA (see Figure 5.7 for the new sequences of execution). We have applied this refinement to the mRI $hire\_p$, as well.

The second refinement we can apply is *participant elimination* [12]. It consists of eliminating those roles from the set of participant roles of an mRI whose knowledge is not referred to by other roles and do not refer to the knowledge of any other role. Figure 5.4 shows a diagram in which we have depicted the knowledge of the roles participating in mRI *approv* and their relationships. Obviously, role MA can be eliminated from this mRI.

Another refinement called *splitting*, which cannot be apply to our example, consist in breaking an mRI into two mRIs if the knowledge accessed by several groups of roles are disjoint as is depicted in Figure 5.5 with a fictitious mRI.

The resulting role plans after applying all refinements are presented in Figure 5.7. Apparently, they works well but we can discover that the refinements have introduced a deadlock situation if we take a closer look. Consider a trace in which the following mRIs are executed: $next\_sale, approv, transfer_1$, and $hire\_p_1$. This execution deadlocks because of an unfortunate interleaving in which, after approving a sale and charging the CA, this role is ready to interact with the PS by means of $transfer_2$; however, the MA is readied then to

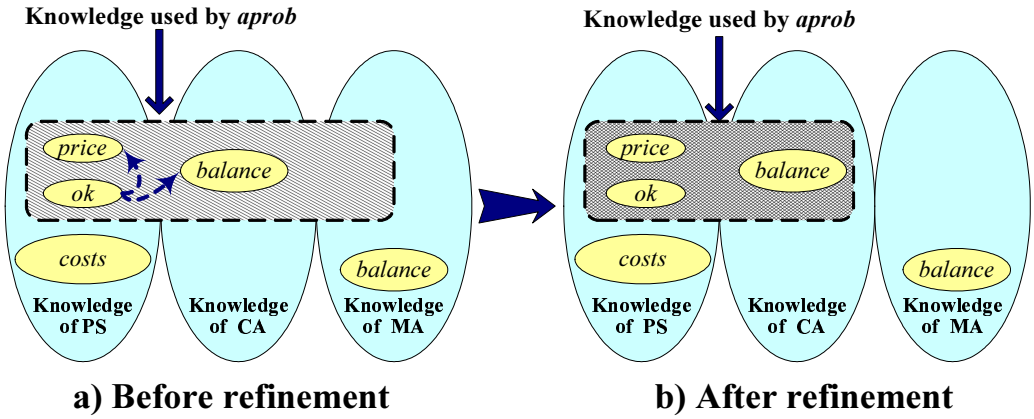


FIG. 5.4. *Eliminating role from approv.*

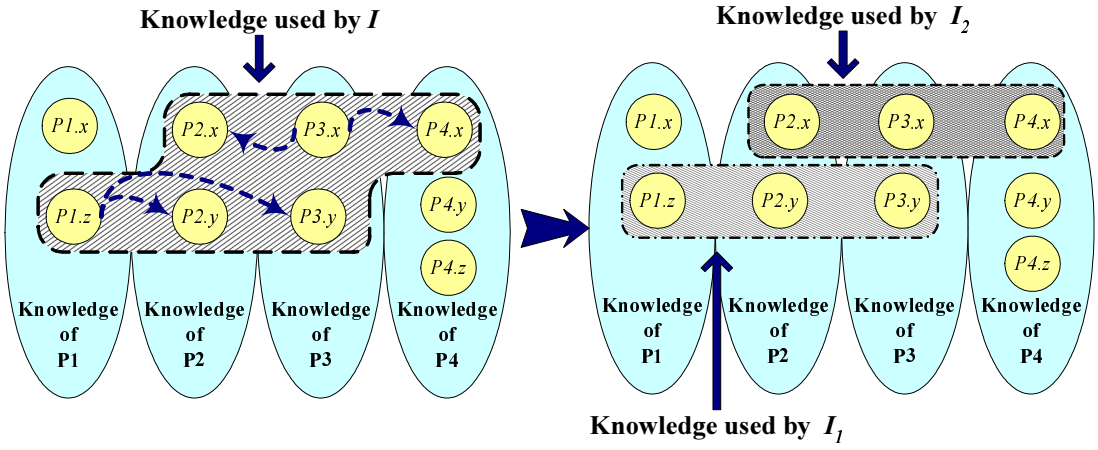


FIG. 5.5. *Splitting fictitious mRI I.*

execute both $transfer_1$ and $hire\_p_1$. If $hire\_p_1$ is executed now, it leads to a situation in which no role can continue because PS is readying $transfer_2$ and waits for the CA to ready it, the CA is readying *approv* and waits for the PS to ready it, and the MA is waiting for any of them to ready $transfer_1$ or $hire\_p_1$. This situation can be avoided if we use a guard for $transfer_i$ and $hire\_p_i$ that ensures that when one of these mRI is executed the guard of the others shall be evaluated as false, but unfortunately this is not possible in general.

These refinements allow us to execute several mRIs at the same time since the the knowledge they computed before refinements is now computed separately in different mRIs. In addition, they simplify the number of participant roles that each mRI uses, which lead us to easier implementations (the protocol to coordinate n parties is more difficult that such for two parties) [12, page. 206][2, 33, 21, 35]. Finally, another advantage is that the amount of knowledge to be processed in each mRI decreases thus easing their internal design.

For instance, the mRI *transfer* has been broken into two simpler mRIs: $transfer_1$ and $transfer_2$. $transfer_1$ computes the balance of the CA and $transfer_2$ computes the balance of the MA. Thus, simpler computations are performed. Furthermore, the original mRI had three participant roles, and the new mRIs have only two, whose coordination/negotiation protocol is simpler to implement. The refined role model is presented in Figure 5.6.

6. Ensuring Deadlock Free Refinements. Our approach to detect deadlocks is based on building an FSA and analysing its paths. Next, we present some results we need, and then we show how to construct the FSA and how to analyse it.

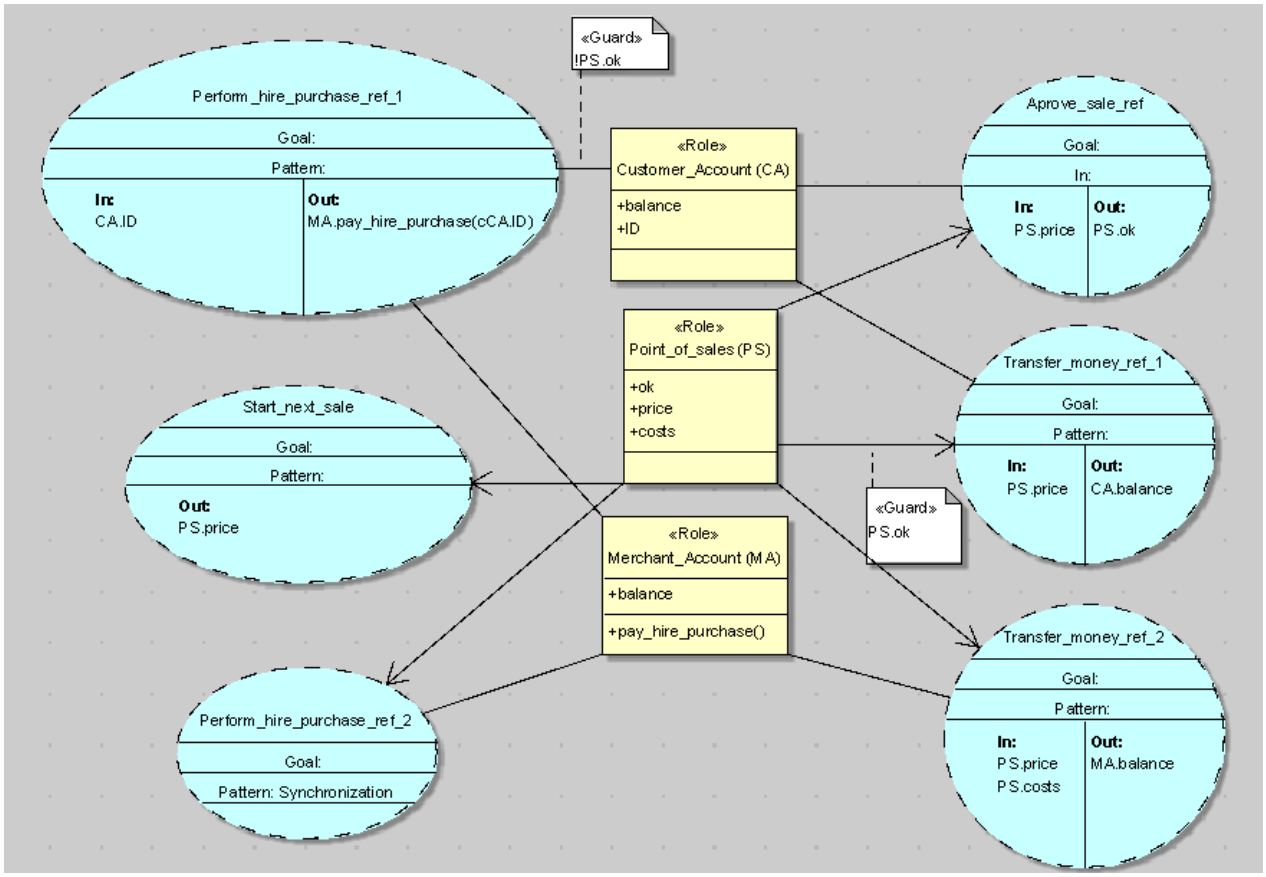


FIG. 5.6. *Role model of the debit–card system after refinements.*

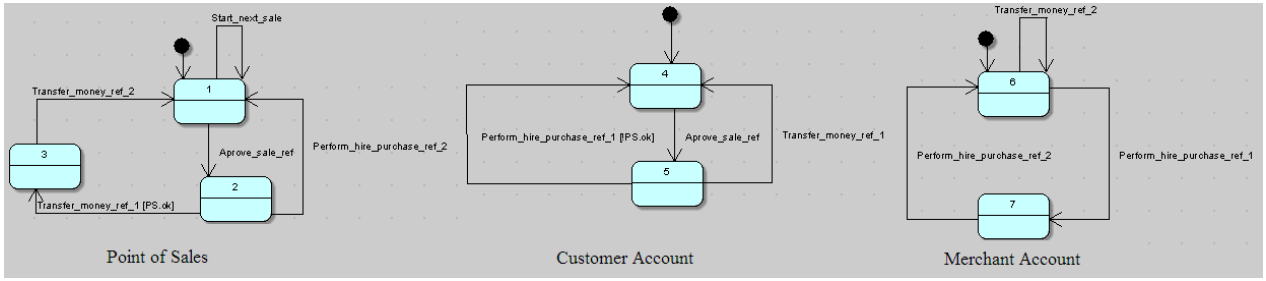


FIG. 5.7. *Role plans after refinement.*

As we can see in Figure 5.7, the definition of the protocol of each role is done by means of FSAs. They can be characterised as follows:

DEFINITION 6.1 (Finite State Automaton). *A finite state automaton (FSA) is a tuple of the form* $(S, \Sigma, \delta, s^0, F)$*, where* S *is a set of states,* Σ *is a set of mRIs (the vocabulary in FSA theory),* $\delta : S \times \Sigma \rightarrow S$ *is a transition function that represents an mRI execution,* $s^0 \in S$ *is an initial state, and* $F \subseteq S$ *is a set of final states.*

Thus, let $A_i = (S_i, \Sigma_i, \delta_i, s_i^0, F_i)$ $(i = 1, 2, \cdots, n)$ be the set of FSAs that represents each role in a role model. Starting from this information we can build a new FSA $C = (S, \Sigma, \delta, s_0, F)$ that represents the protocol as a whole, where

- $S = S_1 \times \cdots \times S_n$
- $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$
- $\delta(a, \{s_1, \ldots, s_n\}) = \{s'_1, \ldots, s'_n\}$ iff $\forall\ i \in [1..n] \cdot (a \notin \Sigma_i \wedge s_i = s'_i) \vee (a \in \Sigma_i \wedge \delta(a, s_i) = s'_i)$

- $e_0 = \{e_1^0, \ldots, e_n^0\}$
- $F = \{F_1, \ldots, F_n\}$

This algorithm has been presented in [25] and builds the new FSA exploring all the feasible executions of mRI. Their states are computed as the cartessian product of all state in FSA of roles. Then, for each new state (composed of one state of each role) we check if an mRI may be executed (all their roles can do it from that state), and if so, we add it to the result. The FSA we obtain in our example is shown in Figure 6.1.

6.1. Analysing the Resulting FSA. The final step consists in analysing the resulting FSA by searching for deadlock states, i.e., states from which a final state cannot be reached.

We use a transition relation called $\longrightarrow_B$ to calculate these states. It is applied on tuples of the form (C, N, X), where C denotes an FSA, N denotes the set of states to be analysed, and X denotes the set of deadlock states found so far. We formalise $\longrightarrow_B$ by means of the following inference rule:

$$\frac{s \in N \wedge s \notin X \wedge P = \mathsf{pred}(s, C)}{(C, N, X) \longrightarrow_B (C, N \setminus P, X \cup P)}$$

Where the predicate pred is defined as follows:

Definition 6.2 (Predecessors). *Let A be an FSA and $s \in S$ a state. We denote its set of predecessors by* $\mathsf{pred}(s, A)$ *and define it as follows:*

$$\mathsf{pred}(s, A) =$$

$$\{s' \in S \mid \exists \sigma \in \Sigma \cdot \delta(s', \sigma) = s\}$$

This transition relation allows us to explore the set of states of an FSA starting at its final states and going back to its predecessors until no new unexplored state is found. The set of unexplored states at that step is the set of deadlock states because there is no path in the FSA that links them to a final state. Therefore, we can define a function deadlock that maps an FSA into its set of deadlock states as follows:

$$\mathsf{deadlock}(C) = C_S \setminus N \text{ if } N \subseteq C_S \wedge$$

$$X \subseteq C_S \wedge (C, C_F, \emptyset) \longrightarrow_B^! (C, N, X)$$

Here, $\longrightarrow_B^!$ denotes the normalisation of $\longrightarrow_B$, i.e., its repeated application to a given tuple until it can not be further applied to the result. Formally,

$$T \rightarrow^! T' \Leftrightarrow T \longrightarrow_E^* T' \wedge \nexists T'' \cdot T' \longrightarrow_E T''$$

If deadlock returns an empty set, then the refinements we have applied do not introduce any deadlocks. Otherwise, we need to characterise the execution paths that may lead to them.

Consider that $\mathsf{deadlock}(C) = \{b_1, b_2, \ldots, b_k\}$, thus, we can build a new set of FSAs

$$B_i = (C_S, C_\Sigma, C_\delta, C_{s^0}, \{b_i\})(i = 1, 2, \ldots, k).$$

Notice that these FSAs have only a final state that is a deadlock state in the original FSA. Thus, if we use the algorithms presented in [14] for transforming an FSA into its corresponding regular expression, we can obtain the set of regular expressions that characterise the execution paths that lead to deadlocks.

If we analyse the FSA in Figure 6.1, we can easily check that its set of deadlock states is a singleton of the form $\{(3, 4, 7)\}$. Thus, if we make this the only final state, we can obtain the following regular expression that characterises the execution paths that lead to deadlocks:

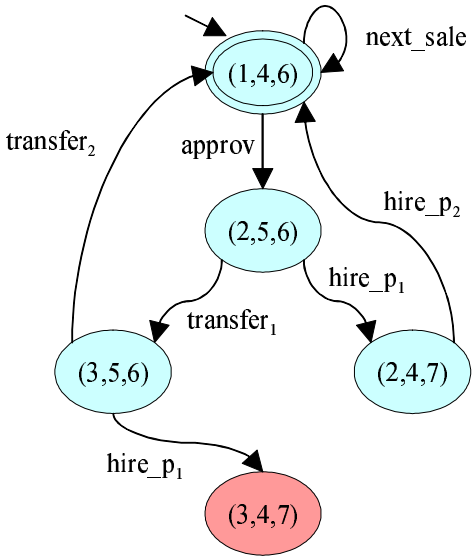


FIG. 6.1. *Resulting FSA.*

$$(next\_sale \mid approv \cdot transf_1 \cdot \\ \cdot transf_2 \mid approv \cdot hire\_p_1 \cdot hire\_p_2)^* \cdot \\ \cdot approv \cdot transf_1 \cdot hire\_p_1$$

Thus, when a set of refinements are applied we can use the technique presented above to search for deadlocks, and if they appear, we characterise it by the deadlock regular expression. Then, we can use this characterization to apply a different set of refinements and repeat this process until getting a deadlock free protocol. Finally, we obtain a set of new simpler mRIs that can be described internally and implemented easier. In our example the deadlock appears between mRI $transfer$ and $hire_p$ and the problem can be easily solved not refining one of them or applying another set of refinements.

7. Conclusions. The description of interaction protocols in complex MASs may be a difficult, tedious process due to the large number of complex tasks that agents must perform coordinately. Thus, in order to palliate this problem, we have proposed a refinement technique integrated in a methodology that is based on an interdisciplinary technique that builds on MAS and distributed systems research results.

Our technique improves previous research in that we add some protocol views between requirements analysis and the description of a protocol by means of message sequences; we use interactions as first class modeling elements. Furthermore, these descriptions are easily refined to reach the needed abstraction level to be described internally. Thus, we provide a progressive method to proceed from requirements analysis to message sequences descriptions. Furthermore, we have provided an automatic method to detect deadlocks.

REFERENCES

[1] R. BACK, A calculus of refinements for program derivations. *Acta Informatica*, 25(6):593–624, 1988.
[2] R. BAGRODIA, Synchronization of asynchronous processes in CSP. *Transactions on Programming Languages and Systems*, 11(4):585–597, Oct. 1989.
[3] B. BAUER, J. MULLER, AND J. ODELL, Agent uml: A formalism for specifying multiagent interaction. In M. Wooldridge and P. Ciancarini, editors, *Proceedings of 22nd International Conference on Software Engineering (ISCE)*, LNCS, pages 91–103, Berlin, 2001. Springer-Verlag.
[4] G. CAIRE, F. LEAL, P. CHAINHO, R. EVANS, F. GARIJO, J. GOMEZ, J. PAVON, P. KEARNEY, J. STARK, AND P. MASSONET, Agent oriented analysis using MESSAGE/UML. In *Proceedings of Agent-Oriented Software Engineering (AOSE'01)*, pages 101–108, Montreal, Canada, May 2001.
[5] J. C. CORBETT, Evaluating deadlock detection methods for concurrent software. *IEEE Transactions on Software Engineering*, 22(3):161–180, March 1996.

[6] R. Corchuelo, D. Ruiz, M. Toro, and A. Durán, Avances en la coordinación de objetos activos. *Novática*, 143:34–37, Jan.–Feb. 2000.
[7] J. C. Cruz, OpenCoLaS a coordination framework for CoLaS dialects. In *Proceedings of COORDINATION 2002*, York, United Kingdom, 2002.
[8] E. Dijkstra, *A Discipline of Programming*. Prentice-Hall, 1976.
[9] T. Elrad and N. Francez, Decomposition of distributed programs into communication-closed layers. *Science of Computer Programming*, 2:55–173, 1982.
[10] N. Francez and I. Forman, Synchrony loosening transformations for interacting processes. In J. Baeten and J. Klop, editors, *Proceedings of Concurr'91: Theories of concurrency—Unification and extension*, number 527 in LNCS, pages 27–30, Amsterdam, The Netherlands, Aug. 1991. Springer-Verlag.
[11] N. Francez and I. Forman, *Interacting processes: A multiparty approach to coordinated distributed programming*. Addison–Wesley, 1996.
[12] N. Francez and I. R. Forman, *Interacting Processes*. Addison–Wesley, 1996.
[13] C. A. R. Hoare, Communicating sequential processes. In R. M. McKeag and A. M. Macnaghten, editors, *On the construction of programs—an advanced course*, pages 229–254. Cambridge University Press, 1980.
[14] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.
[15] C. Iglesias, M. Garrijo, and J. Gonzalez, A survey of agent-oriented methodologies. In J. Müller, M. P. Singh, and A. S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555, pages 317–330. Springer-Verlag: Heidelberg, Germany, 1999.
[16] N. Jennings, An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.
[17] A. Karageorgos and N. Mehandjiev, A design complexity evaluation framework for agent-based system engineering methodologies. In A. Omicini, P. Petta, and J. Pitt, editors, *Fourth International Workshop Engineering Societies in the Agents World*, volume 3071 of *Lecture Notes in Computer Science*, pages 258–274. Springer, 2004.
[18] S. Katz, I. Forman, and W. Evangelist, Language constructs for distributed systems. In *IFIP TC2 Working Conference on Programming Concepts and Methods*, Galilea, Israel, Apr. 1990.
[19] J. Koning, M.Huget, J. Wei, and X. Wang, Extended modeling languages for interaction protocol design. In M. Wooldridge, P. Ciancarini, and G. Weiss, editors, *Proceedings of Second Internationa Workshop on Agent-Oriented Software Engineering (AOSE'02)*, LNCS, Montreal, Canada, May, 2001. Springer–Verlag.
[20] H. J. Levesque, P. R. Cohen, and J. H. T. Nunes, On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 94–99, Boston, MA, 1990.
[21] N. Natarajan, A distributed synchronisation scheme for communicating processes. *The Computer Journal*, 29(2):109–117, Apr. 1986.
[22] O. M. G. (OMG), Unified modeling language: Superstructure. version 2.0. Final adopted specification ptc/03–08–02, OMG, August 2003. `www.omg.org`
[23] G. Papadopoulos and F. Arbab, Coordination models and languages. In *Advances in Computers*, volume 46. Academic Press, 1998.
[24] D. L. Parnas, On the criteria to be used in decomposing system into modules. *Communications of the ACM*, 15(12):1053–1058, December 1972.
[25] J. Peña, R. Corchuelo, and J. L. Arjona, Towards Interaction Protocol Operations for Large Multi-agent Systems. In *Proceedings of the 2nd Int. Workshop on Formal Approaches to Agent-Based Systems (FAABS 2002)*, volume 2699 of *LNAI*, pages 79–91, NASA-GSFC, Greenbelt, MD, USA, 2002. Springer–Verlag.
[26] J. Peña, R. Corchuelo, and J. L. Arjona, A top down approach for mas protocol descriptions. In *ACM Symposium on Applied Computing SAC'03*, pages 45–49, Melbourne, Florida, USA, 2003. ACM Press.
[27] J. Peña, M. G. Hinchey and A. Ruiz-Cortés, Multiagent system product lines: Challenges and benefits. *Communications of the ACM*, 49(12), December 2006.
[28] J. Peña, M. G. Hinchey, A. Ruiz-Cortés and P. Trinidad, Building the core architecture of a nasa multiagent system product line. In *7th International Workshop on Agent Oriented Software Engineering 2006*, page to be published, Hakodate, Japan, May, 2006. LNCS.
[29] J. Peña, M. G. Hinchey, M. Resinas, R. Sterritt, and J. L. Rash, Designing and managing evolving systems using a mas-product-line approach. *Journal of Science of Computer Programming*, 2006.
[30] J. Peña, R. Levy, and R. Corchuelo, Towards clarifying the importance of interactions in agent-oriented software engineering. *International Iberoamerican Journal of AI*, 9(25):19–28, 2005.
[31] J. Peña, M. G. Hinchey, and R. Sterritt, Towards modeling, specifying and deploying policies in autonomous and autonomic systems using an aose methodology. In *EASE '06: Proceedings of the Third IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASE'06)*, pages 37–46, Washington, DC, USA, 2006. IEEE Computer Society.
[32] R. Pressman, *Software Engineering: a Practitioner's Approach*. MacGraw Hill, New York, N.Y., 2nd edition, 1986.
[33] F. Schneider, Synchronization in distributed programs. *ACM Transactions on Programming Languages and Systems*, 4(2):125–148, Apr. 1982.
[34] M. Singhal, Deadlock detection in distributed systems. *Computer Magazine of the Computer Group News of the IEEE*, 22(11):37–48, 1989.
[35] J. Van De Snepscheut, Synchronous communication between asynchronous components. *Information Processing Letters*, 13(3):127–130, Dec. 1981.
[36] M. Y. Vardi and P. Wolper, An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings 1st Annual IEEE Symp. on Logic in Computer Science, LICS'86, Cambridge, MA, USA, 16–18 June 1986*, pages 332–344. IEEE Computer Society Press, Washington, DC, 1986.

[37] M. Wood and S. A. DeLoach, An overview of the multiagent systems engineering methodology. In *Proceedings of the First International Workshop on Agent-Oriented Software Engineering*, number 1957 in LCNS, Limerick, Ireland, 2001. Springer-Verlag.
[38] M. Wooldridge, N. R. Jennings, and D. Kinny, The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

Appendix A. IP Code of the example. It exists several languages based on the Multi-party Interactions (MPI) to describe systems where several processes have to coordinate [6, 10, 13]. IP [12] is worthy of special attention since, although its implementation is relatively simple, moreover it allows to check properties thanks its formal character. Following we will do a brief review of its statements and its more relevant characteristics for our work, and finally we will write the source code of the debit–card system example.

An IP specification is built with a set of sequential processes that cooperates between them using multiparty interactions. Its abstract syntax is the following:

$$\begin{array}{rcl} S & ::= & I_1[\overline{x:=e}] \\ & | & [[]_{i=1}^{n} B_i \,\&\, I_i[\overline{x_i:=e_i}] \rightarrow S_i] \\ & | & \star[[]_{i=1}^{n} B_i \,\&\, I_i[\overline{x_i:=e_i}] \rightarrow S_i] \\ & | & S_1; S_2 \\ & | & skip \end{array}$$

Each processes will be able to participate in several interactions, but only one at the same time. The statement of interaction has the form $I[\overline{x:=e}]$ where I is the name of the interaction and $\overline{x:=e}$ is a sequence of parallel assignments in where we can consult the state of the rest of participants in the interaction, usually referred as communication code. Each Interaction has a set of fixed participants in the set of processes of the system, so that it can be executed only when not any is executing other interaction and all of them are in a point of the specification where the questioned interaction can be executed.

```
TRANSFERS :: [PST() || CustomerAccount() || MerchantAccount()],
where
 PST() :: s: sale := null, ok : boolean;
   *[ v ≠ null & approv[ ok := (cc.balance ≥ s.price)] →
      [ok & transfer[v := null] → skip
      []
      ¬ok & hire_p[] → skip]
    []
      v = null & next_sale[...] → skip],
 CustomerAccount() :: cc: account;
   *[ approv[] →
      [transfer[cc.balance := cc.balance - s.price] → skip
      []
      hire_p[cc.hire_purchase(ma.ID)] → skip] ],
 MerchantAccount() :: ma: account;
   *[ approv[] →
      [transfer[ ma.balance := ma.balance + s.price - v.m_costs ] → skip
      []
      hire_p[ma.balance := ma.balance - s.m_costs] → skip ]
    ].
```

Fig. 7.1. *IP specification of the debit–card system.*

For example, if we analyze the interaction *transfer* in the IP code of the example in the figure 7.1, we can notice it has in its participants[2] with the *PST*, with the *CustomerAccount* and with the *MerchantAccount*. This interaction will not be executed until all its participants will be in an adequate point of the specification and

[2]To determine the participants of an interaction we only have to see in which processes appears in the specification

```
TRANSFERS :: [PST() || CustomerAccount() || MerchantAccount()], where
 PST() :: v: sale := null; ok : boolean;
   *[ v ≠ null & approv[ ok := (cc.balance ≥ s.price)] →
      [ok & transfer_1[] → transfer_2[v := null]
      []
      ¬ok & hire_p_2[] → skip
      ]
    []
      v = null & next_sale[...] → skip ],
 CustomerAccount() :: cc: account;
   *[approv[] →
      [transfer_1[cc.balance := cc.balance - s.price] → skip
      []
      hire_p_1[cc.hire_purchase(ma.ID] → skip ] ],
 MerchantAccount() :: ma: account;
   *[ ι[] →
      [transfer_2[ ma.balance := ma.balance + s.price - s.m_costs] → skip
      []
      hire_p_1[ma.balance := ma.balance - s.m_costs] → hire_p_2[] ]
    ].
```

FIG. 7.2. *IP specification of the example after applying the refinements.*

when this will happen, its participant will execute its communication code. For example, the *PST* will calculate the value of variable *ok* using the balance of the *CustomerAccount* and the amount to transfer.

IP also has statements to write non-deterministic choice with guards $[[]_{i=1}^{n} G_i \rightarrow S_i]$ and loops with nondeterministic choice with guards $*[[]_{i=1}^{n} G_i \rightarrow S_i]$. The guards are of the form $B\&a[\overline{x:=e}]$, where B is a boolean condition involving the local state of a process, and the rest is an usual interaction statement. The behaviour of these statements is very simple: The non-deterministic choice checks all the boolean conditions and wait then for the interactions whose boolean condition is true to have all its participants; if no one could do so the statement will not have any effect. In loops the behaviour is similar, only that it will repeat the non-deterministic choice until all the boolean conditions are false.

Furthermore, in IP we can make the statements above to execute sequence $(S_1; S_2)$, and we can use the null statement that is represented as *skip*.

Finally, the code resultant after applying all the refinements described above is shown in Figure 7.2.

Edited by: Marcin Paprzycki, Niranjan Suri
Received: October 1, 2006
Accepted: December 10, 2006

Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 63–77. http://www.scpe.org
ISSN 1895-1767

OBSERVATION-BASED PROACTIVE COMMUNICATION IN MULTI-AGENT TEAMWORK

YU ZHANG*

Abstract. Multi-agent teamwork is governed by the same principles that underlie human cooperation. This paper describes how to give agents the same cooperative capabilities, observability and proactivity, that humans use. We show how agents can use observation of the environment and of teammates' actions to estimate the teammates' beliefs without generating unnecessary messages; we also show how agents can anticipate information needs among the team members and proactively communicate the information, reducing the total volume of communication. Finally, we present several experiments that validate the system developed, explore the effectiveness of different aspects of observability and introduce the scalability of the use of observability with respect to the number of agents in a system.

Key words. Multi-agent systems, teamwork, agent communication, observability

1. Introduction. Recently, the focus of much research on multi-agent systems (MAS) has shifted from strong agency [26] to teamwork, which is a cooperative effort by a team of agents to achieve a common or shared goal [23]. Research on multi-agent teamwork builds on findings about effective human team behaviors and incorporates them into intelligent agent technologies. For example, the shared mental model, one of the major aspects of the psychological underpinnings of teamwork, has been adopted widely as a conceptual basis of multi-agent teamwork. Based on the shared mental model, an effective team often can anticipate the information needs of teammates and offer pertinent information proactively [18, 22]. Consequently, supporting proactive information exchange among agents in a multi-agent teamwork setting is crucial [29]. Substantial challenges arise in a dynamic environment because agents need to deal with changes. Although partial observability of dynamic, multi-agent environments has gained much attention [17, 11], little work has been done to address how to process what is observable and under which conditions; how an agent's observability affects the individual's mental state and whole team performance; and how agents can communicate proactively with each other in a partially observable environment.

In this paper, we focus on how to include represent observability in the description of a plan, and how to include it into the basic reasoning for proactive communication. We define several different aspects of observability (e.g., seeing a property, seeing another agent perform an action, and believing another can see a property or action are all different), and propose an approach to the explicit treatment of an agent's observability that aims to achieve more effective information exchange among agents. We employ the agent's observability as the major means for individual agents to reason about the environment and other team members. We deal with communication with the 'right' agent about the 'right' thing at the 'proper' time in the following ways:

- Reasoning about what information each agent on a team will produce, and thus, what information each agent can offer others. This is achieved through: 1) analysis of the effects of individual actions in the specified team plans; 2) analysis of observability specification, indicating what and under which conditions each agent can perceive about the environment as well as the other agents.
- Reasoning about what information each agent will need in the process of plan execution. This is done through the analysis of the preconditions of the individual actions involved in the team plans.
- Reasoning about whether an agent needs to act proactively when producing some information. The decision is made in terms of: 1) whether or not the information is mutable according to information classification; 2) which agent(s) needs this information; and 3) whether or not an agent who needs this information is able to obtain the information independently according to the observation of environment and other agents' behaviors.

We also present several experiments that validate the system developed, explore the effectiveness of different aspects of observability and introduce the scalability of the use of observability with respect to the number of agents in a system.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 is an overview of the system architecture, which is called CAST-O. Section 4 discusses how an agent's observability is represented, and how an agent's beliefs are maintained in the course of observations. Section 5 describes observation-based

*Department of Computer Science, Trinity University, San Antonio, TX, 78216, USA.

proactive communication among agents. Section 6 is an empirical study based on a multi-agent Wumpus World. Section 7 summarizes our work and discusses issues for further research.

2. Related Work. A single agent's observability and reasoning have received researchers' attentions for some time. Perception reasoning is one of these research directions [16, 24]. For example, "seeing is believing" has been adopted for perception-based belief reason[2, 13]. In recent years, observability has been used widely to understand behaviors of multi-agent systems. One study of particular interest is a logic for visibility, seeing and knowledge (VSK), which explores relationships between what is true, visible, perceived, and known; it also investigates a number of interaction axioms among agents, such as under which condition agent a sees everything agent b sees or agent b knows everything agent a sees [27]. However, VSK logic does not address two major issues regarding agent cooperation: 1) an agent uses the effects of actions in reasoning what others are likely to know, but VSK does not provide a way to treat actions through observation; 2) VSK does not provide agents with an effective way to utilize their observation to manage communication. Isozaki and Katsuno propose an algorithm to reason about agents' nested beliefs (which are one's belief about the belief of another), based on observatio[10]. However, they do not represent the process of observation, such as what can be seen and under which conditions. Tambe and Kaminka use observation to monitor failed social relationships between agents [12], but they do not give details about how agents' belief about their teammates' mental states are updated. Viroli and Omicini devise a formal framework for observation that abstracts conditions that cause agents' interactive behavi [25]. But, they don't say much about how the observation to environment is processed. All of above fall into the category "passive observation", in the sense that each agent evaluates observability conditions at the appropriate times. Our work also belongs to passive observation. However, we aim to reduce the amount of communication by reasoning about agent observability, the capability to observe environment and actions. We relate an agent's observability to its mental state, and then use observation and belief about others' observabilities to estimate its teammates' mental states. That is, an agent can exploit knowledge about what it and its teammates can see to help decide when others might or might not know some information. Ioerger has considered "active observation", in which he invokes additional 'find-out' plans to seek values for unknown conditions knowledge of whose values would enable situation assessment [9].

To date, control paradigms for cooperative teamwork have allowed agents to communicate about their intentions, plans, and the relationships between them [23, 21]. However, this complex team cooperation behavior requires high-frequency communication and computation time, which weakens teamwork efficiency. Moreover, some researchers have found that communication, while a useful paradigm, is expensive relative to local computation [1]; therefore techniques that reduce extraneous communication during teamwork processes are of particular importance. On the other hand, there exist several communication-less agent cooperation techniques such as social conventions [20], focal points [14], plan recognition [8], decision-theoretic modeling [15, 28], and game-theoretic recursive modeling [5]. In general, these techniques emphasize inferring others' actions implicitly or explicitly, based on established norms for behavior or on knowledge about the preferences or interests of others. However, strategies such as social conventions or focal points totally eliminate communication and use convention rules to guide agents' actions, strategies such as plan recognition or decision-theoretic normally have high computational complexity in dealing with uncertainty which weakens teamwork efficiency, and game-theoretic recursive modeling is primarily suitable for two-member teams. Our approach to proactive communication is different in that agents are capable of predicting team-related information (by analyzing team plans) and distributing such information only when it is necessary. The communication need is reduced, by using belief of what agents can observe, and hence don't have to be told.

3. The CAST-O Architecture. The CAST-O architecture is an extension of CAST (Collaborative Agents for Simulating Teamwork) [29]. There are three aspects to the extension: 1) representation of agent observability about the environments and other agents' actions; 2) belief-maintenance in terms of observation; 3) observation-based proactive communication among agents.

An agent team is composed of a set of agents. The team members share the team knowledge that is represented in MALLET (Multi-Agent Logic Language for Encoding Teamwork), which provides descriptors for encoding knowledge about teamwork processes (i. e. individual/team plans and operations), as well as specifications of team structures (e.g., team members and roles) [30]. Each agent has an individual knowledge base (KB) to specify its beliefs about the environment and beliefs about teammates' mental states. The environment simulation provides an interface through which the agents can interact with the environment. In

the process of plan execution, individual agents can observe the environment and their teammates' behaviors, infer the teammates' mental states, communicate with each other, and perform actions.

Plans are at the center of activity. They describe how individuals or teams can go about achieving various goals. Plans are classified into individual plans and team plans. Each individual plan has a process consisting of a set of operations, each of which is either a primitive operator, or a composite operation (e.g., a sub-plan). Team plans are similar to individual plans, but they allow multiple agents or agent variables to be assigned to carry out operations or plans (some of the requiring a team). A DO statement is used to assign one or several agents to carry out specific operators or sub-plans. The following is an example team plan for the multi-agent version of Wumpus World (refer to section 6 for more details):

```
(tplan killwumpus()
  (process
    (par
      (seq
        (agent-bind ?ca (constraint (play-role ?ca carrier)))
        (DO ?ca (findwumpus ?w))) // carrier is assigned
      (seq
        (agent-bind ?fi (constraint ((play-role ?fi fighter)
                        (closest-to-wumpus ?fi ?w))))
        (DO ?fi (movetowumpus ?w)) // fighter who is closest to
                                   // wumpus is assigned
        (DO ?fi (shootwumpus ?w)))))) // shootwumpus is an operator
```

where `findwumpus` and `movewumpus` are individual plans, and `shootwumpus` is an individual operator specified as follows:

Generally, operators are defined by their preconditions and effects, which are logical conjunctions. An individual action is the execution of an instantiated operator in a DO statement. It is represented as:

```
<action> ::= (DO <doer> (<operator-name> <args>))
```

where `<doer>` is the agent assigned to the action and `<operator-name>` and `<args>` are correspondent to the name and arguments of the operator. Sample individual actions in the extended Wumpus World are as follows:

```
(DO ?fi (shootwumpus ?w))
(DO ?ca (pickupgold ?g))
```

We assume that the precondition of the action must be believed by <doer> before the action can be performed and the effect must be believed after the action is performed. Since actions are domain-dependent, when agents perform the actions, they send a signal to the environment simulation. Then the actions are visible to any team member whose observability (see section 4) permits it at the time the actions are performed.

An essential feature that differentiates an agent team from a set of individual agents is that a team of agents may perform a joint action, which is the union of simultaneous individual actions performed by individuals sharing certain specific mental properties [4]. MALLET provides a descriptor joint-do for agents performing the joint action, and specifies three different joint types: AND, OR or XOR [29]. For example, we may define following joint action in the extended Wumpus World:

```
(joint-do AND
   (DO ?ca (move ?x ?y))
   (DO ?fi (move ?x ?y)))
```

which means agents `?ca` and `?fi` move simultaneously.

Given a team plan expressed in MALLET, we can explicitly deduce information needs and production from the pre-conds and effects of operators and implicitly deduce others from the plan structure, e.g., joint-do requires coordination regarding starting time, or operations in parallel need coordination in terms of the starting and ending of the par set of branches. The latter, for example, might be determinable from observations, avoiding the need for explicit communication. In addition, if multiple agents are capable of performing the same tasks, the MALLET team plan is likely to contain agent selection criteria (e.g., the closest agent to a wumpus should kill it). Again, this falls in the realm of implicitly determinable coordination communication. While this paper has focused on the only the explicitly determinable part of this (i. e., things derived from pre-conds and effects conditions), the basic structure of the use of observation can be applied to more general situations.

Another important setting for agents' teamwork is environment. The environment is composed of objects. Each object has some properties. A property is represented as follows:

```
<property> ::= (<property-name> <object> <args>)
<object>     ::= <agent>|<non-agent>
```

where <object> could be either agent or non-agent, and <args> is a list of arguments describing the property. Sample properties in the extended Wumpus World are as follows:

```
(location fi ?x ?y),
(dead w1 ?state).
```

The usefulness of properties derives from treating them as queries to the environment, using variables for any or all of the arguments. Unification will provide values, if any, for the free variables that make the query true; if there are no such values, then the value for the query will be false.

During a teamwork process, the environment simulation provides an interface through which the agents can observe the environment and their teammates' actions. The environment evolves from the state at one time to the state at the next time with an action possibly being taken during the time interval, saving only the current environment states. Each agent maintains knowledge of the environment in its KB, updating this knowledge as needed to carry out its plan or provide information to team members.

4. Agent Observability. To express agent observability, we define a query function CanSee(<observer> <observable> <cond>), where <observer> specifies the agent doing the observing, <observable> identifies what is to be observed, and <cond> specifies the conditions under which the <observer> can see the <observable>. When needed, the query is submitted to the knowledge base for evaluation after first forming the conjunction of the arguments. As <observablea> and <cond> may be predicates, missing values for variables will be supplied via unification if there are any such values that allow the <cond> to be satisfied, or else return FALSE. This allows an agent, for example, to determine the location (through variables) of a target if the conditions are satisfied (e.g., the target is within range). Time is implicit in this query and is taken to be the time of the current step. Note that strong constraints weaken agents' observability; weak constraints strengthen agents observability. The strongest constraint is FALSE, which means that the agent can see nothing. The weakest constraint is TRUE, which means that the agent can see everything.

Successful teamwork requires interdependency among the agents [6]. This suggests that an agent should know at least some things about what other team members can see. However, an agent may not know for sure that another agent can see something. Rather, an agent may only believe, based on its current beliefs, that another agent can see something. We then use BelieveCanSee(<believer> <observer> <observable> <cond>) to mean that one agent believes another agent can see something under certain condition.

We also make the assumption of "seeing is believing". While philosophers may entertain doubts because of the possibility of illusion, common sense indicates that, other things being equal, one should believe what one sees [13, 2]. Thus, we assume that an agent believes an observed property persists until it believes the property has been negated later.

In the following subsections, we describe the syntax and semantics of observability in more detail.

4.1. The Syntax of Observability. The syntax we use for observability is given in Table 4.1. For example, the observability specification for a carrier in the extended Wumpus World is shown below, where ca, rca, fi, rfi represent the carrier, carrier's detection radius, fighter and fighter's detection radius, respectively.

```
(CanSee ca (location ?o ?x ?y)
  (location ca ?xc ?yc) (location ?o ?x ?y)
  (inradius ?x ?y ?xc ?yc rca)
) // The carrier can see the location property of an object.

(CanSee ca (DO ?fi (shootwumpus ?w))
  (play-role fighter ?fi) (location ca ?xc ?yc) (location ?fi ?x ?y)
  (adjacent ?xc ?yc ?x ?y)
) // The carrier can see the shootwumpus action of a fighter.

(BelieveCanSee ca fi (location ?o ?x ?y)
  (location fi ?xi ?yi) (location ?o ?x ?y)
  (inradius ?x ?y ?xi ?yi rfi)
) // The carrier believes the fighter is able to see the
  // location property of an object.
```

TABLE 4.1
The Syntax of Observability

1:	$< observability >$	$:= (CanSee < viewing >)*$
2:		$(BelieveCanSee < believer >< viewing >)*$
3:	$< viewing >$	$:= < observer >< observable >< cond >$
4:	$< believer >$	$:= < agent >$
5:	$< observer >$	$:= < agent >$
6:	$< observable >$	$:= < property > \mid < action >$
7:	$< property >$	$:= (< property - name >< object >< args >)$
8:	$< action >$	$:= (DO < doer > (< operator - name >< args >))$
9:	$< object >$	$:= < agent > \mid < non - agent >$
10:	$< doer >$	$:= < agent >$

```
(BelieveCanSee ca fi (DO ?f (shootwumpus ?w))
  (play-role fighter ?f) (  ?f fi) (location ca ?xc ?yc)
  (location fi ?xi ?yi) (location ?f ?x ?y)
  (inradius ?xi ?yi ?xc ?yc rca) (inradius ?x ?y ?xc ?yc rca)
  (adjacent ?x ?y ?xi ?yi)
) // The carrier believes the fighter is able to see the
  // shootwumpus action of another fighter.
```

An agent has two kinds of knowledge, shared team knowledge, encoded in MALLET, and individual knowledge, contained in its knowledge base. The syntax of observability can be used either, as rules in an agent's knowledge base [31], or as capability incorporated into MALLET. In this paper, we encode observability as rules in agents' knowledge bases.

4.2. The Semantics of Observability. To give operational semantics to observability, we need to clarify the relationships of: 1) what an agent can see, what it actually sees, and what it believes from its seeing; 2) what an agent believes another agent can see, what it believes another agent actually sees, and what it believes another agent believes from its seeing.

In order to properly discuss the semantics, we need to introduce a notion of time, as preconditions and effects refer to different points in time. For purposes of exposition, we will simply assume that time is a discrete and indexed in order by the natural numbers, and use the indices to reference points in time. Since we are dealing with multiple agents, multiple actions may occur at the same time instant. We do not try to elaborate further on time in this paper, as there are a number of useful different ways of dealing with issues such as the synchronization among team members performing actions, and they are not central to the point of the paper.

Let $See_t(a, \psi)$ express that agent a observes ψ at time t. There are two cases to consider, first where ψ is a property, and secondly, where ψ is an action. When ψ is a property, seeing ψ means determining the truth value of ψ, with unification of any free variables in ψ. If ψ is an action, seeing ψ means that the agent believes the doer believed the precondition of ψ immediately before the action occurred and the doer believes the effect of ψ immediately after performing the action. We use the meta-predicate $Hold_t$(c) to mean c holds in the world (environment simulation) at time t. We make the assumption below:

$$\forall a, \psi, c, t, CanSee(a, \psi, c) \wedge Hold_t(c) \rightarrow See_t(a, \psi) \tag{4.1}$$

which means that if the condition c holds at time t and agent a has the capability to observe ψ under condition c, then agent a actually does determine the truth-value of ψ at time t.

Next, we consider the relation between seeing something and believing it. Belief is denoted by the modal operator BEL and for its semantics we adopt the axioms K, D, 4, 5 in modal logic. The assumption of "seeing is believing" is again stated separately for properties and actions. In the case of properties, it is formalized in the axiom below:

$$\forall a, \varphi, t, See_t(a, \varphi) \rightarrow [Hold_t(\varphi) \rightarrow BEL_t(a, \varphi)] \wedge [\neg Hold_t(\varphi) \rightarrow BEL_t(a, \neg\varphi)] \tag{4.2}$$

which says that for any property φ seen by agent a, if φ holds, agent a believes φ; if φ does not hold, agent a believes not φ ($\neg\varphi$).

Agent a's belief is more complex when an action, ϕ, is observed. Let $Doer(\phi)$, $Prec(\phi)$, $Efft(\phi)$ denote the doer, the precondition, and the effect of action ϕ. When agent a sees action ϕ performed by some agent, agent a believes that the agent believed the precondition and believes the effect. This process is expressed by the following axiom:

$$\forall a, \phi, t, See_t(a, \phi) \rightarrow BEL_t(a, BEL_{t-1}(Doer(\phi), Prec(\phi))) \wedge BEL_t(a, BEL_t(Doer(\phi), Eff_t(\phi))) \tag{4.3}$$

From the belief update perspective in our current implementation where beliefs are assumed persistent, for any $p \in Prec(\phi)$, agent a believes that $Doer(\phi)$ still believes p at time t (i. e. $BEL_t(a, BEL_t(Doer(\phi), p))))$ unless $\neg p$ is contained in Efft(ϕ). This is similar for *BelieveCanSee*.

An agent's belief about what another agent sees is based on the following axiom:

$$\forall a, b, \psi, c, t, t', BelieveCanSee(a, b, \psi, c) \wedge BEL_t(a, BEL_{t'}(b, c)) \rightarrow BEL_t(a, See_{t'}(b, \psi)) \tag{4.4}$$

which means that if agent a believes that agent b is able to observe ψ under condition c, and agent a believes c at time *t'*, then agent a believes at time t that agent b saw (*t'*<t), sees (*t'*=t), or will see (*t'*>t, which requires some prediction capability for agent a) ψ at time *t'*. In our approach, each agent focuses on the reasoning about current observability, not in the past or in the future. Therefore, the axiom above can be simplified as follows:

$$\forall a, b, \psi, c, t, BelieveCanSee(a, b, \psi, c) \wedge BEL_t(a, c) \rightarrow BEL_t(a, See_t(b, \psi)) \tag{4.5}$$

Note that agent a evaluates condition c according to its own beliefs.

Combining this with the previous assumption that "seeing is believing". we extend this to belief. We have two separate cases for properties and actions. When agent a believes agent b sees a property φ, a believes that b believes φ:

$$\forall a, b, \varphi, t, BEL_t(a, See_t(b, \varphi)) \rightarrow BEL_t(a, BEL_t(b, \varphi)) \tag{4.6}$$

When agent a believes agent b sees an action ϕ, a believes that b believes the doer believed the precondition at the previous time step and believes the effect at the current time step. This consequence is expressed by the following:

$$\forall a, b, \phi, t, BEL_t(a, See_t(b, \phi)) \rightarrow BEL_t(a, BEL_t(b, BEL_{t-1}(Doer(\phi), Prec(\phi)))) \wedge BEL_t(a, BEL_t(b, BEL_t(Doer(\phi), Efft(\phi)))) \tag{4.7}$$

4.3. Belief Maintenance. From the semantics, agents' observability is closely tied to their beliefs about the environment and other agents. Agents must update these beliefs when they perform, or reason about others', observation.

4.3.1. Maintaining Belief About Self's Observability. The axiom of "seeing is believing" bridges the gap between what an agent sees and what it believes. An agent maintains its beliefs in two aspects: 1) for an observed property, the agent believes the property; 2) for an observed action, the agent believes that the doer believed the precondition before the action and the doer believes the effect after the action. The algorithm for updating what an agent has observed, according to the observability rules, is given in Figure 4.1.

This algorithm builds beliefs in the believer's (i. e., agent self's), knowledge base by checking the following: Observing a property

- When evaluating observability `(CanSee self (<prop-name> <object> <args>) <cond>)`, self queries `<cond>` to environment KB. The query returns a list of substitutions of variables, or null if `<cond>` are not satisfied. When the returned tuple is not null, if the property holds in the environment, self updates its knowledge base with belief `(<prop-name> <object> <args>)` for each variable bindings, otherwise, self updates its knowledge base with belief `(not (<prop-name> <object> <args>))` for each variable bindings.

- Observing an action
  In the case of `(CanSee self (<action-name> Agd(≠ self) <args>) <cond>)`, the query `<cond>` is made with respect to environment KB as well. If the result of query is not null, self updates its beliefs by that self believes that agent Agd knew the precondition, and that Agd infers the effect. To handle the temporal issue correctly, self updates Agd's belief about the precondition first and then Agd's belief about the effect. These beliefs are useful in communication. For example, if agent a needs information I and believes agent b believes I, a may ask b for I.

updateSelfObs(self, KBself)
/* Let self be the agent invoking the algorithms. We denote the knowledge base for agent a by KB_a, for the environment by KB_{env}.*/
1: for each rule in KB_{self} of the form (CanSee self (prop object args) cond)
2: if cond is true in KB_{env} for some bindings of variables
3: if (prop object args) is true in KB_{env} for some bindings of variables
4: update(KB_{self}, (prop object args))
5: for each such binding of values to the variables;
6: else
7: update(KB_{self}, (not (prop object args)))
for each such binding of values to the variables;
8: for each rule in KB_{self} of the form (CanSee self (action doer args) cond),
if cond is true in KB_{env} for some binding of variables,
9: for each conjunct of precondition of action
10: update(KB_{self}, (BEL doer conjunct));
11: for each conjunct of effect of action
12: update(KB_{self}, (BEL doer conjunct));

FIG. 4.1. *An Algorithm of Maintaining Self s Belief by Direct Observation*

4.4. Maintaining Belief About Others' Observabilities. Figure 4.2 shows an algorithm for updating what an agent can determine about what other agents can see.

updateSelfBel(self, KB_{self})
1: for each rule of the form (BelieveCanSee self Ag (prop object args) cond) that
2: cond is true in KB_{self} for some binding of arguments to agents $Ag \neq self$
3: for each such binding of arguments to the variables
4: update(KB_{self}, (BEL Ag (prop object args)));
5: for each rule of the form (BelieveCanSee self Ag (action doer args) cond) that
6: cond is true in KB_{self} for some binding of arguments to agents $Ag \neq self$
7: for each conjunct of the precondition of action
8: update(KB_{self}, (BEL Ag (BEL doer conjunct)));
9: for each conjunct of the effect of action
10: update(KB_{self}, (BEL Ag (BEL doer conjunct)));

FIG. 4.2. *An algorithm of maintaining belief about others observabilities*

The algorithm records which agents are known to be able to see what, and updates what an agent believes, according to the precondition and effect of the actions it observes other agents performing. For the agent to determine whether a piece of information is needed by others, it simulates the inference process of others' observability to determine which is known by others.

- Observing a property
  In the case of `(BelieveCanSee self Ag(≠ self) <property> <cond>)`, a query `<cond>` is made with respect to KB_{self}. If the condition is satisfied, self believes Ag can see the property. However, self may or may not have knowledge of `<property>`. For example, a carrier may believe a fighter can smell a wumpus if the fighter is adjacent to the wumpus, but the carrier does not itself smell the wumpus.

- Observing an action In the case of (BelieveCanSee self Ag(≠self) (<action-name> doer(≠self) <args>) <cond>), <cond> is evaluated with respect to KB_{self}. Self adds tuples to KB_{self}, indicating that Ag believes that doer believed the preconditions of the action, and believes the effects of the action[1].

4.5. Execution Model. At each time step, every agent, denoted by self, has a function cycle: (possibly) observe, receive information from others, belief coherence, (possibly) send information to others, and act. If self needs an information item or produces an item needed by others, it will observe the world and other agents. It then checks messages and adjusts its beliefs for what it sees and what it is told. Self keeps track of the other agents' mental states by reasoning about what they see from observation, in order to decide when to assist the others with the needed information proactively. Finally, self acts cooperatively with teammates and enters the next time step.

An algorithm for overall belief maintenance along with the function cycle is shown in Figure 4.3. The algorithm begins with updateWorld by self's last action. We will not elaborate on how updateWorld works which is beyond the focus of this paper. Basically, the environment simulation updates the environment KB after receiving any action from the agent. Because the agent can infer the effect of its own action, the algorithm saves the effect as a new belief. UpdateSelfObs evaluates observability rules with information obtained from KB_{env} and updates KB_{self} with the results of the observation. UpdateSelfBel updates self's beliefs about what others' beliefs by observing environment and actions.

updateKB(self, action, KB_{self})
/* The algorithm is executed independently by each agent, denoted self below,
after the completion of each step in the plan in which the agent is involved.*/
1: updateWorld(action, self); //notify the environment to update KB_{env}
2: for each conjunct in the effect of action
3: update(KB_{self}, conjunct);
4: if self produces/needs information I
5: updateSelfObs(self, KB_{self}); //update KB_{self} by observability
6: updateSeflBel(self, KB_{self}); //update KB_{self} by beliefs about
//others observabilities
7: for each coming information I
8: update(KBself, I); //update KB_{self} by communication

FIG. 4.3. *An overall belief-maintenance algorithm*

The function update manages history and is responsible for coherence and persistence of belief in an agent's KB. The agent's beliefs about the world are saved as primitive predicates as they were expressed originally in the world. Such beliefs are generated from three sources: (1) belief from observation, i. e., a property self observes; (2) belief from inference, i. e., conjuncts inferred from the effect of the action self performs; (3) belief from communication, i. e., messages other agents send to self by communication. How does communication affect the agent's mental state? Van Linder et al. propose that the communication can also be translated to a belief saved in the mental state in the same way as observation is [13]. In any situation in which belief is required from multiple sources, conflicts may arise, such as self simultaneously sees $\neg\psi$ and hears ψ. A strategy is needed that prescribes how to maintain the coherence of the knowledge base of an agent in the case of conflicts among incoming information from different sources. Castelfranchi proposes that such a strategy should prescribe that more credible information should always be favored over less credible information [3]. To define a strategy complying with this idea, we propose that each source is associated with a credit and the credit decreases in this order: source from observation, source from inference, and source from communication. At certain time point, when an agent gets conflict information from different sources, it always believes what it sees.

Since the number of time steps could be infinite, an agent keeps only current beliefs in its mental state, except that the most recent one is kept, even if it is not generated currently. That an agent does not directly observe or infer some predicates from current observation does not mean it does not believe them. The agent has memory of them from before. Memory is useful in proactive communication; thus, if a piece of information is infrequently changed, at the time when agent a realizes that agent b needs the information, even if agent a does not have the information, agent a can tell agent b the information in its memory.

[1]Note, however, that self does not necessarily know what there values are. This is useful, however, in case self needs to make an activeAsk.

5. Proactive Communication. The purpose of proactive communication is to reduce communication overhead and to improve the efficiency or performance of a team. In our approach, proactive communication is based on two protocols named proactiveTell and activeAsk. These protocols are used by each agent to generate inter-agent communications when information exchange is desirable. Proactive communication answers the following questions pertinent to agent proactivity during teamwork. First, when does an agent send the information to its teammates if it has a new piece of information (either from performing an action or observing)? A simple solution could be sending the information when requested. That is, the agent would only send the information after it has received a request from another agent. Our approach is that the agent observes its teammates, and commits to proactive tell once it realizes that one of the teammates needs the information to fulfill its role and does not have it now. Meanwhile, if the agent needs some information, it does not passively wait for someone else to tell it; it should ask for this information actively. Second, what information is sent in a session of information exchange? There are two kinds of information that can be communicated. One is the information explicitly needed by an agent to complete a given plan, i. e., conjuncts in a precondition of plans or operators that the agent is going to perform. The other is the information implicitly needed by the agent. For example, if agent a needs predicate p and knows p can be deduced from predicate q, even if the providing agent does not know p, it still can tell agent a about q once it has q, because it knows that agent a can deduce p from q. This paper, however, deals only with agents communicating information that is explicitly needed.

The proactiveTell and activeAsk protocols are designed based on following three types of knowledge:

- Information needers and providers. In order to find a list of agents who might know or need some information, we analyze the preconditions and effects of operators and plans and generate a list of needers and a list of providers for every piece of information. The providers are agents who might know such information, and the needers are agents who might need to know the information.
- Relative frequency of information need vs. production. For any piece of information I, we define two functions, f_C and f_N. $f_C(I)$ returns the frequency with which I changes. $f_N(I)$ returns the frequency with which I is used by agents. We classify information into two types: static[2] and dynamic. If $f_C(I) \leq f_N(I)$, I is considered static information; if $f_C(I) > f_N(I)$, I is considered dynamic information. For static information we use proactiveTell by providers, and for dynamic information we use activeAsked by needers[3].
- Beliefs generated after observation. Agents take advantage of these beliefs to track other team members' mental states and use beliefs of what can be observed and inferred to reduce the volume of communication. For example, if a provider believes that a needer sees or infers information I, the provider will not tell the needer.

An algorithm for deciding when and to whom to communicate for activeAsk and proactiveTell[4] is shown in Figure 5.1.

Considering the intractability of general belief reasoning [7], our algorithm deals with beliefs nested no more than one-layer. This is sufficient for our current study on proactive behaviors of agents, which focuses on peer-to-peer proactive communication among agents. For activeAsk, an agent requests the information from other agents who may know it, having determined it from the information flow. The agent selects a provider among agents who know I and ask for I. For proactiveTell, the agent tells other agents who need I. An agent always assumes others know nothing until it can observe or reason that they do know a relevant item. Information sensed and beliefs about others' sensing capabilities become the basis for this reasoning. First, the agent determines what another agent needs from the information flows. Second, the observation rules are used to determine whether or not one agent knows that another agent can sense the needed information.

6. Empirical Study. While one would think that if one gives an agent additional capabilities, its performance would improve, and indeed this turns out to be correct, there are several other interesting aspects of our scheme to evaluate. For example, when there are several different capabilities, the interesting question arises of how much improvement each capability gives and which capabilities are the most important to add in different situations. Moreover, while it is obvious that one should not see decreasing performance from increasing

[2]Here, static information includes not only the information never changed, but also the information infrequently changed but frequently needed.

[3]In future work, we will address some statistical methods to calculate frequencies and hence will be able to provide more comprehensive proactive communication protocols.

[4]Note that there is no need to say anything anout previous time points, as those would have been handled when they were first entered. Furthermore, these is no need to consider $\neg I$ explicitly; if ture, it will be entered as a fact on its own.

activeAsk(self, I, KB_{self}, T)
/* Let T be the time step when the algorithm is executed.
Independently executed by each agent (self) when it
needs the value of information I.*/
1: candidateList=null;
2: if (I is dynamic and (I t) $\vee$ ($\neg I$ t) is not true in KB_{self} for any t$\leq$T)
3: if there exists a x$\geq$0 such that
4: ((BEL Ag I T-x) $\vee$ (BEL Ag $\neg I$ T-x)) is true in KB_{self}
5: let xs be the smallest such value of x;
6: for each agent Ag$\neq$self
7: if ((BEL Ag I T-xs) $\vee$ (BEL Ag $\neg I$ T-xs)) is true in KB_{self}
8: add Ag to candidateList;
9: randomly select Ag from candidateList;
10: ask Ag for I;
11: else
12: randomly select a provider
13: ask the provider for I;

proactiveTell(KB_{self}, T)
/* Independently executed by each agent (self), after it executes updateKB.*/
14: for each conjunct I for which (I, T) is true in KB_{self} and I is static
15: for each Agn needers
16: if (BEL Agn I T) is not true in KB_{self}
17: tell Agn I;

FIG. 5.1. *Proactive Communication Protocols*

capabilities, there are still interesting questions of how much performance increase can be obtained and how one can incorporate the capabilities into the system in a computationally tractable manner. And, one there is an interest in how the scheme scales with the number of agents involved. Our empirical study is intended to address these questions.

To test our approach, we have extended the Wumpus World problem [19] into a multi-agent version. The world is 20 by 20 cells and has 20 wumpuses, 8 pits, and 20 piles of gold. The goals of the team, four agents, one carrier and three fighters, are to kill wumpuses and get the gold. The carrier is capable of finding wumpuses and picking up gold. The fighters are capable of shooting wumpuses. Every agent can sense a stench (from adjacent wumpuses), a breeze (from adjacent pits), and glitter (from the same position) of gold. When a piece of gold is picked up, both the glitter and the gold disappear from its location. When a wumpus is killed, agents can determine whether the wumpus is dead only by getting the message from others, who kill wumpus or see shooting wumpus action. The environment simulation maintains object properties and actions. Agents may also have additional sensing capabilities, defined by observability rules in their KBs.

There are two categories of information needed by the team: 1) an unknown conjunct that is part of the precondition of a plan or an operator (e.g., "wumpus location" and "wumpus is dead"); 2) an unknown conjunct that is part of a constraint (e.g., "fighter location", for selecting a fighter closest to wumpus). The "wumpus location" and "wumpus is dead" are static information and the "fighter location" is dynamic information. Agents use proactiveTell to impart static information they just learned if they believe other agents will need it. For example, the carrier proactiveTells the fighters the wumpus' location. Agents use activeAsk to request dynamic information if they need it and believe other agents have it. For example, fighters activeAsk each other about their locations and whether a wumpus is dead.

We used two teams, Team A and Team B. Each team was allowed to operate a fixed number of 150 steps. Except for the observability rules, conditions of both teams were exactly the same. In the absence of any target information (wumpus or gold), all agents reasoned about the environment to determine their priority of potential movements. If they were aware of a target location requiring action on their part (shoot wumpus or pick up gold), they moved toward the target. In all cases, they avoided unsafe locations.

We report three experiments. The first explores how observability reduces communication load and improve team performance in multi-agent teamwork. The second focuses on the relative contribution of each type of

TABLE 6.1
Team Performance and Communication Frequency in Sample Run. T1: number of wumpuses left alive, T2: amount of gold left unfound, T3: total number of avtiveAsks used, T4: total number of proactiveTells used, T5: average number of activeAsks per wumpus killed, T6: average number of proactiveTells per wumpus killed

	$T1$	$T2$	$T3$	$T4$	$T5$	$T6$
$TeamA$	4.8	7.2	77.4	33.8	5.09	2.23
$TeamB$	15	14.6	67.6	28.8	13.6	5.9

belief generated from observability to the successes of CAST-O as a whole. Finally, the third evaluates the impact of observability on changing communication load with increase of team size.

Two teams are defined as follows:

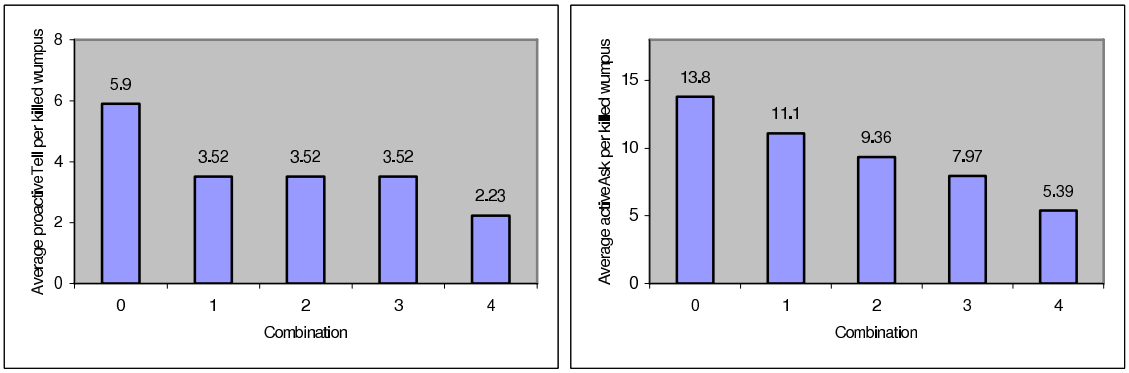
- Team A: The carrier can observe objects within a radius of 5 grid cells, and each fighter can see objects within a radius of 3 grid cells.
- Team B: None of the agents have any seeing capabilities beyond the basic capabilities described at the beginning of the section.

We use measures of performance, which reflect the number of wumpuses killed, the amount of communication used and the gold picked up. In order to make comparisons easier, we have chosen to have decreasing values indicate improving performance, e.g., smaller numbers of communication messages are better. To maintain this uniformity with some parameters of interest, we use the quantity not achieved by the team rather than the number achieved, e.g., the number of wumpuses left alive rather than the number killed. The experiments were performed on 5 randomly generated worlds. The results are shown in Table 1.

Table 1 shows that, as expected, Team A killed more wumpuses and found more gold than Team B. From other experiments we have learned that the further the agents can see, the more wumpuses they kill. It is interesting that the absolute number of communications is higher for Team A with observabilities than that of Team B, thus 33.8 vs. 28.8 for proactiveTell and 77.4 vs. 67.6 for activeAsk. The reason for the increased number of proactiveTells is that in Team A, the carrier, who is responsible for finding wumpuses and proactiveTelling wumpuses' locations to fighters, has further vision than that of the carrier in Team B. Hence the carrier in Team A can see more wumpuses. This feature leads to more proactiveTells from the carrier to the fighters in Team A. The number of proactiveTells can be reduced by the carrier's beliefs about the fighters' observability, i. e., if the carrier believes the fighters can see the wumpus' location, it will not proactiveTell the fighters. However, since the fighters' detect range is smaller than that of the carrier, the reduction cannot offset the number of extra proactiveTells. The reason for the increased number of activeAsks in Team A is that the more wumpuses they find, the more likely it becomes that messages are sent among fighters to decide who is closest to the wumpuses. Since fighters in Team A may find wumpuses by themselves, they need to ask other teammates if the wumpus is dead, to decide whether to kill it or not. Although the number of the messages could be reduced by factors such as allowing the fighter to see other fighters' locations and to see other fighters killing a wumpus, the increase cannot be totally offset because of the fighters' short vision. Hence, it makes more sense to compare the average number of messages per wumpus killed. In these terms, the performance of Team A, is much better than that of Team B, thus 2.23 vs. 5.9 for proactiveTell and 5.09 vs. 13.6 for activeAsk. Hence, our algorithms for managing the observability of agents have been effective.

The results of this experiment produced a bit of a surprise. By introducing observabilities to agents, the amount of communication actually increased slightly. This can be explained by the fact that because observability is a major means for an individual agent to obtain information about environment and team members; the more information obtained by the agent, the more messages were conveyed to help others. The proper way to interpret the results, then, is to normalize them by the performance of the team, which in this case is the average number of communications per wumpus killed, denoted by ACPWK, in this example. From this perspective, the amount of communication was reduced, as expected, also validating our approach.

6.1. Evaluating Different Beliefs Generated from Observability. The second experiment tested the contribution of different aspects of observability to the successful reduction of the communication. These aspects are belief about observed property, belief about the doer's belief about preconditions of observed action, belief about the doer's belief about effects of observed action and belief about another's belief about observed property. For simplify, we call them belief1, belief2, belief3 and belief4 correspondently. We test their contributions by



(a) proactiveTell protocol (b) activeAsk protocol

FIG. 6.1. *Average Communication Per Killed Wumpus in Different Combinations*

combining them. We used Team A and Team B in this experiment and kept all conditions the same as those of the first experiment. We used Team B, as reference to evaluate the effectiveness of different combinations of observability with Team A. We named this test combination 0, since there is none of such four beliefs involved in. For Team A, we tested another 4 combinations of these beliefs to show the effectiveness of each, in terms of ACPWK. These combinations are:

- Combination 0: Team B, which involves none of beliefs.
- Combination 1: In Team A, for each agent, leave off BelieveCanSee rules and do not process belief2 and belief3 when maintaining beliefs after observation. Therefore every agent only has belief1 about the world.
- Combination 2: Keep every condition in combination 1, except for enabling the belief2 process. This combination tests how belief2 improves the situation.
- Combination 3: Enabling the belief3 process in combination 2. This combination tests the effect of belief3.
- Combination 4: Add BelieveCanSee rules into combination 3. This combination tests the effect of belief4 as well as show effectiveness of the beliefs as a whole.

Each combination is run in the five randomly generated worlds. The average results of these runs are presented in Figure 6.1, in which one bar shows ACPWK for one combination.

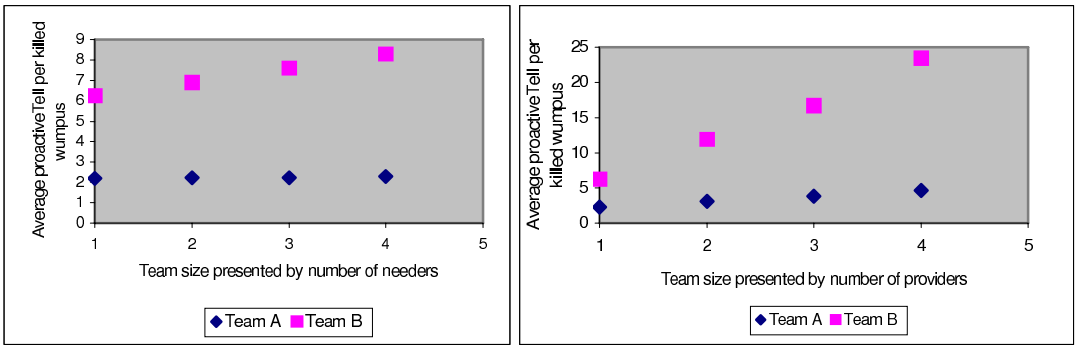
First of all that, agents' belief1 (combination 1) is a major contributor to effective communication, for both proactiveTell and activeAsk. For proactiveTell, in (a), compared to combination 0, ACPWK significantly drops from 5.9 to 3.52. For activeAsk, in (b), ACPWK drops from 13.8 to 11.1.

The second case, belief2 (combination 2) does not produce any further reduction and hence is not effective for proactiveTell, but produces improvement for activeAsk. For proactiveTell, when a provider sees an action, though it believes the doer knows the precondition and effect of the action, it does not know the precondition and effect by itself. So for this example belief2 can be of little help in proactiveTell. While for activeAsk, belief2 reduces ACPWK from 11.1 to 9.36, because with belief2, a needer will know who has a piece of information explicitly. Then it can activeAsk without ambiguity.

Third, for the same reason that belief2 only works for activeAsk, belief3 (combination 3) contributes little to proactiveTell but further decreases ACPWK to 7.97 for activeAsk.

Fourth, belief4 (combination 4) has a major effect on communications that applies to both protocols. It further drops ACPWK to 2.23 for proactiveTell and to 5.39 for activeAsk. Belief4 is particularly important for proactiveTell. For example, if the carrier believes that the fighters see a wumpus' location, it will not tell them.

This experiment examined the contribution of each belief deduced from observability to the overall effectiveness of communication. The result indicates three things. First, belief1 and belief4 have a strong effect on the efficiency of both proactiveTell and activeAsk. Therefore, CanSee/BelieveCanSee a property, the observability from which these two beliefs generated, can be generally applied to dual parts communication involving both Tell and Ask. Second, belief2 and belief3 have weak influence on the efficiency of proactiveTell, this suggests



(a) needers increment (b) providers increment

FIG. 6.2. *The Comparison of ProactiveTell with Different Team Size*

that CanSee an action may be applied to communication which incurs more Ask than Tell, such as goal-directed communication. Third, these beliefs work best together, because each of them provides a distinct way for agents to get information from the environment and other team members. Furthermore, they complement each other's relative weaknesses, so using them together better serves the effectiveness of the communication as a whole.

6.2. Evaluating the Effect of Observability Communication Load with Increased Team size. We designed the third experiment to show how communication load scales with increased team size. Based on the assumption that proactiveTell brings more communication into play than activeAsk, we choose to test the proactiveTell protocol. ActiveAsk is directed to only one provider at certain time, while the proactiveTell goes to all needers who do not have the information. If the test results are good for proactiveTell, we can expect that they are valid for activeAsk as well.

We used the same sensing capabilities for Teams A and Team B as in the first experiment. However, we increased the number of team members by 1, 2 and 3, in two tests that we ran. In the first test, we increased the number of needers, (i. e. fighters) and kept the same number of providers, (i. e. carriers). In the second test, we did it the other way around. In each test, for each increment and each team, we ran the five randomly generated worlds and used the average value of ACPKW produced in each world.

Figure 6.2 shows the trend of ACPKW as a function of increasing team size. In (a), Team B has an obvious increase in ACPKW with increasing the team size. However, Team A keeps the same ACPKW. The cause can be attributed to two factors: first, the amount of the increasing proactiveTells is held down because if the carrier believes the fighters can see wumpus, the carrier does not perform proactiveTell; second, the more fighters there are, the more wumpuses will be killed, which enlarges the numerator of ACPKW.

In (b), increasing the number of providers breaks the constant trend in Team A and shows an increased ACPWK. However, comparing this increase to that of Team B, it is a moderate number. In Team B, every provider increment means almost double the number of proactiveTells. The communication load increases because of duplicate proactiveTells of the same information by different carriers. For example, each carrier always provides the wumpus' location to fighters when observing a wumpus. The carriers lack an effective way to predict when a piece of information is produced and by whom, which is one of our main concerns of future work. This experiment shows that the team empowered with observability has a slower growth of ACPWK with increase of team size, which may indicate that observability will improve team scalability in some sense.

7. Conclusion. In this paper, we have presented an approach to dealing with agent observability for improving performance and reducing inter-agent communication. Each CAST-O agent is allowed to have some observability to see the environment, and to watch what others are doing inside its detection range. Based on the observation, the agent updates its knowledge base and infers what others may know at the current time. Reasoning about what others can see allows agents to decide whether to distribute information and to whom. We have proposed a proactive communication mechanism to confer some advantage to related team members for realizing team interaction and cooperation proactively also. We have conducted an in-depth empirical

evaluation in an extended Wumpus World, comparing the relative numbers of proactiveTell, activeAsk, and wumpuses killed for agent teams with and without observability.

A major point to the proactive communication approach with observabilities is that the underlying system that interprets the team plans of the agents does most of the work for handling the observation, inference and communication. This need only be designed once. It is re-used as one moves from one domain to another. It is only the explication of the observability conditions that changes from one domain to another, and this is essentially linearly proportional to the number of agents and "size" of the domain properties that are to be observed.

Though currently we are considering just the times of information production or need, the same approach can be extended to uncertainty in observability as well. Additionally, our present proactive information algorithm analyzes the pre-conditions and effects of operators for which each agent is responsible in the team plan. The purpose of doing so is to determine potentially useful information flow among agents. However, this approach is restrictive. We would like to make the recognition of needed information more dynamic. One way to solve this problem is to recognize the plans of other agents by observing actions of the other agents, and tracking the sequence of sub-goals on which they are working dynamically. Using this information together with the action an agent has most recently performed, the most likely information needs of other agents can be dynamically estimated over a finite time horizon. Then we can send other agents only unknown information that will be needed in the near future.

Acknowledgement. This work was supported in part by DoD MURI grant F49620-00-I-326 administered through AFOSR.

REFERENCES

[1] T. Balch and R. C. Arkin, *Communication in reactive multi-agent robotic systems,*, Autonomous Robots, 1 (1994), pp. 27–53.
[2] J. Bell and Z. Huang, *Seeing is believing*, in Proceedings of Common Sense 98, 1998, pp. 391–327.
[3] C. Castelfranchi, *Guarantees for autonomy in cognitive agent architecture*, Intelligent Agents, (1996), pp. 56–70.
[4] P. R. Cohen and H. J. Levesque, *Teamwork*, Nous, Special Issue on Cognitive Science and Artificial Intelligence, 25 (1991), pp. 487–512.
[5] P. J. Gmytrasiewicz, E. H. Durfee, and D. K. Wehe, *A decision-theoretic approach to coordinating multi-agent interactions.*, in Proceedings of 12th International Joint Conference on Artificial Intelligence, 1991.
[6] B. J. Grosz, *Collaborating systems*, AI Magazine, 17 (1996).
[7] J. Y. Halpern and Y. A. Moses, *A guide to completeness and complexity for modal logics of knowledge and belief*, Artificial Intelligence, (1992), pp. 319–379.
[8] M. J. Huber and E. H. Durfee, *Deciding when to commit to action during observation-based coordination*, in Proceedings of the 1st International Conference on Multi-agent Systems, 1995, pp. 163–170.
[9] T. R. Ioerger and L. He, *Modeling command and control in multi-agent systems*, in 8th International Command and Control Research and Technology Symposium (ICCRTS), June 17-19 2003.
[10] H. Isozaki and H. Katsuno, *Observability-based nested belief computation for multi-agent systems and its normalization*, Intelligent Agent IV, (2000), p. LNAI 1757.
[11] G. A. Kaminka, D. V. Pynadath, and M. Tambe, *Monitoring deployed agent teams*, in Proceedings of International Conference on Autonomous Agents, 2001.
[12] G. A. Kaminka and M. Tambe, *Robust agent teams via socially-attentive monitoring*, Journal of Artificial Intelligence Research, 12 (2000), pp. 105–147.
[13] B. V. Linder, W. V. D. Hoek, and J. Meyer, *Seeing is believingâĂŤand so hearing and jumping*, Topics in Artificial Intelligence, LNAI 992 (1995), pp. 402–413.
[14] S. K. M. Fenster and J. S. Rosenschein, *Coordination without communication: Experimental validation of focal point techniques,*, in Proceedings of the 1st International Conference on Multi-agent Systems,, 1995, pp. 102–108.
[15] M. G. M. Genesereth and J. Rosenchein, *Cooperation without communication,*, tech. rep., Stanford Heuristic Programming project, Computer Science Department, Stanford University,, 1984.
[16] D. Musto and K. Konolige, *Reasoning about perception*, in Proceedings of the AAAI Spring Symposium on Reasoning About Mental States, 1993, pp. 90–95.
[17] D. Pynadath and M. Tambe, *Multiagent teamwork: Analyzing the optimality and complexity of key theories and models*, in Proceedings of the 1st Autonomous Agents and Multiagent System Conference, 2002.
[18] W. B. Rouse, J. A. Cannon-Bowers, and E. Salas, *The role of mental models in team performance in complex systems*, IEEE Transactions on Systems, Man, Cybernetics, 22 (1992), pp. 1296–1308.
[19] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, NJ: Prentice Hall, 2002.
[20] Y. Shoham and M. Tennenholtz, *On the synthesis of useful social laws for artificial agents societies (preliminary report)*, in Proceedings of the 9th National Conference on Artificial Intelligence, 1992.
[21] F. Stone and M. Veloso, *Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork*, Artificial Intelligence, 110 (1999), pp. 241–273.
[22] K. P. Sycara and M. C. Lewis, *Forming shared mental models*, in Proceedings of 13th Annual Meeting of the Cognitive Science Society, 1991, pp. 400–405.

[23] M. Tambe, *Towards flexible teamwork*, Journal of Artificial Intelligence Research, 7 (1997), pp. 83–124.
[24] A. D. Val, P. M.-R. II, and Y. Shoham, *Qualitative reasoning about perception and belief*, in Proceedings of 15th International Joint Conference on Artificial Intelligence, 1997, pp. 508–513.
[25] M. Viroli and A. Omicini, *An observation approach to the semantics of agent communication languages*, Applied Artificial Intelligence, 16 (2002), pp. 775–793.
[26] M. Wooldridge and N. R. Jennings, *Intelligent agents: Theory and practice*, The Knowledge Engneering Review, 10 (1995), pp. 115–152.
[27] M. Wooldridge and A. Lomuscio, *Multi-agent vsk logic*, Proceedings of the 17th Eurapean Workshop on Logics in AI, (2000).
[28] P. Xuan, V. Lesser, and S. Zilberstein, *Communication decisions in multi-agent cooperation: Model and experiments*, in Proceedings of the 5th international conference on autonomous agents, 2001, pp. 616–623.
[29] J. Yen, X. Fan, and R. A. Volz, *A theoretical framework on proactive information exchange in agent teamwork*, Artificial Intelligence Journal, 169 (2005), pp. 23–97.
[30] J. Yen, X. Fan, R. Wang, S. Sun, and R. A. Volz, *Context-centric needs anticipation using information needs graphs*, Journal of Applied Intelligence, 24 (2006), pp. 75–89.
[31] Y. Zhang and R. A. Volz, *Modeling cooperation by observation in agent team*, in Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC'05), 2005, pp. 536–541.

Edited by: Marcin Paprzycki, Niranjan Suri
Received: October 1, 2006
Accepted: December 10, 2006

Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 79–86. http://www.scpe.org
ISSN 1895-1767

STABILITY, OPTIMALITY AND COMPLEXITY OF NETWORK GAMES WITH PRICING AND PLAYER DROPOUTS

ANDREW LOMONOSOV* AND MEERA SITHARAM†

Abstract. We study basic properties of a class of noncooperative games whose players are selfish, distributed users of a network and the game's broad objective is to optimize Quality of Service (QoS) provision. This class of games was previously introduced by the authors and is a generalization of well-studied network congestion games.

The overall goal is to determine a minimal set of static game rules based on pricing that result in stable and near optimal QoS provision.

We show the following. (i) Standard techniques for exhibiting stability or existence of Nash equilibria fail for these games—specifically, neither are the utility functions convex, nor does a generalized potential function exist. (ii) The problem of finding whether a specific game instance in this class has a Nash equilibrium is NP-complete.

To offset the apparent instability of these games, we show positive results. (iii) For natural subclasses of these games, although generalized potential functions do not exist, *approximate* Nash equilibria do exist and are easy to compute. (iv) These games perform well in terms of "price of stability" and "price of anarchy." I.e., all of these approximate Nash equilibria nearly optimize a communal (or social) welfare function, and there is atleast one Nash equilibrium that is optimal.

Finally, we give computer experiments illustrating the basic dynamics of these games which indicate that price thresholds could speed up convergence to Nash equilibria.

Key words. Congestion games, Selfish routing, Atomic unsplittable model, Nash Equilbria, Network pricing

1. Introduction. Recently much research has been done in applying game-theoretic concepts and general economics techniques to analysis of computer network traffic [2, 3, 5, 10, 11, 12, 16, 14, 20, 21, 24]. For a general survey see [1]. Stability in games refers to whether the game reaches a *Nash equilibrium,* a state where no player has incentive to move. Optimality is a measure of how close a Nash equilibrium is to optimizing a *social or communal welfare function*, usually the sum of the individual players' utility functions.

We consider primarily *atomic* games, where the number of players (network users) is finite. The case of *non-atomic games* where there is an infinite number of infinitesimally small players is easier to analyze. For similar reasons, *spittable* games, where network users can split their volume onto many service classes are easier to analyze and have more orderly behavior than *unsplittable games*, where each user is forced to place all their volume onto the same class.

The atomic splittable network game model has been studied [20, 12], with early results in the transportation literature. Efficiency (or optimality) of Nash equilibria in atomic splittable network games was studied in [24] and [28].

Here we consider primarily the unsplittable case that has also been studied for some time, for example [26].

Most of the research deals with *congestion games* where payoff to a player depends only on the player's strategy and on the number of players choosing the same strategy. Thanks to [26] it is known that such games always have Nash equilibrium. Two common techniques that are used to demonstrate existence of Nash Equilbria are the following. When the player utility functions are convex, Kakutani's fixed point theorem [25] directly shows existence. Also when such convexity properties are not present, *potential functions*, [18], certain functions that increase after every move, are used to show existence. These have a long history, for example, as Lyapunov stability functions classically used to describe equilibria in dynamical systems.

The [23] network games have realistic features that make them somewhat different from congestion games: in particular, players have non-convex utility functions caused by a threshold of total traffic volume in service classes that they are willing to tolerate. In addition in the [15] games, the players are allowed to refrain from participation, or to *dropout*, if their traffic quality demands are not satisfied. Hence existence of Nash equilibria or potential functions is not guaranteed for these classes of games. However, we were able to show existence of Nash equilibria for some of these classes of games by constructing *generalized potential functions.* (Generalized) potential functions have also been used by others to study versions of congestion and other games e.g., [7, 21, 22].

For the classes of games in [15, 16] we additionally showed that the Nash equilibria established via generalized potential functions are easy to compute. In general, however, while potential functions guarantee existence of Nash equilibrium, the problem of actually finding such an equilibrium remains computationally challenging.

*UGS Inc., 10824 Hope Street, Cypress, CA 90630 USA(lomonoso@ugs.com).
†CISE, University of Florida, Gainesville, FL 32611, USA(sitharam@cise.ufl.edu).

It has been shown [7] that the problem of finding Nash Equilbrium in congestion games is PLS-Complete, which intuitively means "as hard to compute as any object whose existence is guaranteed by a potential function".

Considerable research has gone into the *price of anarchy* and *price of stability* of Nash equilibria [27]. These notions describe how far or how close Nash equilbria can be to the *System Optimum* of a game, where system optimum is a configuration (not necessarily a Nash equilibrium) that has greatest communal welfare.

We showed that for the classes of games with Nash equilibria in [15, 16], the communal welfare at these equilibria was poor, i. e., they are far from the system optimum. To rectify this, we further generalized our classes of games by introducing *pricing* incentives (not to be confused with the word "price" in the previous paragraph). The effect of pricing on congestion games has also been studied in [9, 6, 8]. Our original goal was to modify our original class of games so that the Nash equilibria would be close to system optima. However, the priced games were shown to not have Nash equilibria, in general. We instead showed that there is trade-off between game stability (existence of Nash Equilbria) and communal welfare achieved by such games. I.e., while the priced games did not always have Nash equilibria, the Nash equilibria, when they existed, were close to the system optima.

This trade-off has since been formalized by examining *approximate Nash equilibria* i. e. states where no player can improve their individual welfare by more than a certain factor, and the value of communal welfare at such approximate equilibria [4]. For example, [2] demonstrated a tradeoff between welfare and stability when costs functions are semiconvex.

In this paper, our overall goal is to analyze our classes of realistic network congestion games with respect to these stability and communal welfare measures; investigate *mechanisms* for games to optimize these measures; and to pose formal questions about the structure of game classes imposed by such measures.

More specifically, the original classes of games introduced in [15] were: the class $\mathcal{Q}$ where players were solely motivated by their traffic quality demands and classes $\mathcal{PQ}$ where players were also influnced by prices imposed on traffic. Stability of games in $\mathcal{Q}$ was demonstrated by means of general potential functions, and concrete examples of instability of $\mathcal{PQ}$ were then given.

In this paper, we establish the NP-completeness of determining existence of Nash equilibria and for computing Nash Equilbria in $\mathcal{PQ}$. We further study stability and communal welfare of (a modified version of) approximate Nash equilibria in $\mathcal{PQ}$, as compared to class $\mathcal{Q}$ (i. e. effect of pricing on stability and social welfare in our games).

We also briefly look at game *dynamics*, i. e. number of steps that it actually takes to converge to Nash Equilbria for some of our games and conduct computer experiments to study trade-off between willingness to pay and speed of convergence.

Section 2 presents preliminary definitions, Section 3 presents previous results on the class $\mathcal{Q}$ of games, Section 4 presents the main results of this paper concerning the class $\mathcal{PQ}$, and Section 5 concludes by tabulating and comparing the results of Sections 3 and 4, followed by open problems.

2. Definitions. A *game (instance)* G in the base class of QoS provision network games is specified by the *game parameters* $G = \langle n, m \in N, \{\lambda_i \in R^+ : 1 \leq i \leq n\}, \{b_{i,j} \in R^+ : 1 \leq i \leq n; 1 \leq j \leq m\}, \{p_j : R^+ \rightarrow R, 1 \leq j \leq m\}\rangle$. The best way to define G is by identifying it with its finite game configuration graph (formally defined below) which consists of a set of feasible game configurations (vertices) and the valid or selfish game moves (oriented edges). The game G is played by n *users or players* each wanting to send a traffic of λ_i units through one of m network service classes and (for convenience of analysis) an overflow or Dummy Class with index 0, referred to as *DC*. Each player i additionally has a *volume threshold* $b_{i,j}$ (to be described below) for each class j. A *price* function $p_j()$ for each service class is a nonincreasing function that maps the total (traffic) volume in the class to a unit price. (Unit price typically decreases with increasing congestion or total volume in any service class). The price for using DC is 0. A *feasible configuration* Λ of G is fully specified by an allocation $J_\Lambda : \{1, \ldots, n\} \rightarrow \{1, \ldots, m\}$ which describes which service class $J_\Lambda(i)$ that the user or *player* i has decided to place their chunk λ_i of traffic. This allocation J_Λ results in a *total traffic volume* $q_{\Lambda,j} = \sum_{i:1\leq i\leq n \wedge J_\Lambda(i)=j} \lambda_i$ in each class $1 \leq j \leq m$ at the game configuration Λ. The set of feasible game configurations F form the *vertex* set of the *game configuration graph* Ω. *Individual utility function* $U_i(\Lambda)$ is a type of step function based on i's volume threshold being met at the configuration Λ, and on the unit price incurred by the player i in its class $j = J_\Lambda(i)$. $U_i(\Lambda)$ is:

- 0 if $j = 0$ (user i is in DC)

- $-\epsilon$, for small $\epsilon > 0$ if $b_{i,j} < q_{\Lambda,j}$ (volume threshold exceeded)
- equal to $\lambda_i(1 - p_j q_{\Lambda,j})$ otherwise.

It is assumed that the price functions are always appropriately normalized so that this quantity is always *strictly positive* for all players i and their classes $J_\Lambda(i)$ at any configuration Λ. A typical utility function is shown on Figure 2.1. We say that user i is *satisfied* at configuration Λ if $U_i(\Lambda) \neq 0$, and not satisfied otherwise. We define

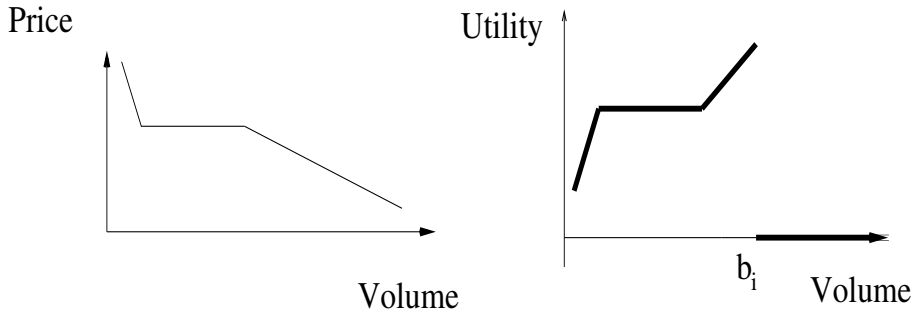


FIG. 2.1. *Utility as a function of volume, volume threshold and price*

a function $Sat_\Lambda(i) = 1$ if $U_\Lambda(i) \neq 0$, otherwise $Sat_\Lambda(i) = 0$. A *selfish move* by user i at a configuration Λ_1 is a reallocation of i's volume λ_i from a departure class j_1 (i.e $J_{\Lambda_1}(i) = j_1$), to a a destination class j_2 resulting in a configuration Λ_2 (i.e, $J_{\Lambda_2}(i) = j_2$) that increases utility of this user, i.e, $U_i(\Lambda_1) < U_i(\Lambda_2)$. Moves to DC by a user whose volume threshold is exceeded are called *user dropouts*. Note that user dropouts qualify as selfish moves according to our definition.

Each selfish move is an ordered pair of feasible game configurations (for example $(\Lambda_1, \Lambda_2) \in F \times F$), and represents an *oriented edge* of the game configuration graph Ω. A *generalized potential function* is a function defined on configurations that increases after every player move. A *game play* for G is a sequence of valid selfish moves in G, i.e $(\Lambda_1, \Lambda_2), (\Lambda_2, \Lambda_3), \ldots, (\Lambda_{k-1}, \Lambda_k)$, or a *path* in the game configuration graph Ω. A *Nash Equilibrium* or *NE* of a game G is a configuration Λ such that there is no selfish move possible for any user i. Nash equilibria are exactly sink vertices of a game configuration graph Ω that have no outgoing edges toward other vertices. For our classes of games, the *communal welfare function* for configuration Λ is defined as $C(\Lambda) = \Sigma_i Sat_\Lambda(i)\lambda_i$. The feasible game configuration that has highest value of communal welfare function is called the *System Optimum* or SO. Let Λ_N be a Nash Equilibrium that has the smallest value of communal welfare function taken over all Nash Equilibriums, while Λ_M be a Nash Equilibrium that has the largest value. As defined in say [27] a *price of anarchy* of a game is equal to $C(\Lambda_N)/C(\Lambda_*)$, where Λ_* is SO. A *price of stability* is equal to $C(\Lambda_M)/C(\Lambda_*)$.

Class of games that do not have pricing, i. e. $p_j(x) = 0$ for all classes j and their volumes x is denoted by $\mathcal{Q}$. In such games players are motivated only by their desire to satisfy their volume thresholds. Subclass $\mathcal{Q}_\mathcal{E} \subset \mathcal{Q}$ is a class of games with no pricing where all players have equal volume. Class of games that have only one pricing function $p(x)$ for all classes j and this function is strictly decreasing ($p(x) < p(y) \leftrightarrow x > y$) is denoted by $\mathcal{PQ}$. Subclass $\mathcal{PQ}_\mathcal{E} \subset \mathcal{PQ}$ is a class of games with single strictly decreasing price function where all players have equal volume. Here we will give a pictorial example, Figure 2.2, of some notions introduced in this section. A game configuration graph Ω and configurations Λ of a particular game G are shown. Columns represent classes, rectangles represent users, the size of a rectangle corresponds to volume of a user, volume thresholds of users are indicated on the right. In this example the game G in class $\mathcal{PQ}$ has 2 classes, 2 users A and B that have equal volumes and the volume threshold of A is greater than that of B. Game configuration graph Ω has 4 vertices. This game G has no Nash equilibrium.

Throughout this paper we assume wlog that every player i has the same volume threshold $b_i = b_{i,1} = b_{i,2} = \ldots b_{i,m}$ in every class $j = 1 \ldots m$. We also assume that players are sorted in the increasing order of their thresholds, i.e $b_1 \leq b_2 \leq \ldots \leq b_n$. (The former assumption could be easily generalized for all results in this paper, the latter assumption is realistic and commonly made [23]).

In proofs when describing a game configuration Λ, we will specify values of game parameters n and m, provide a list of users in the form User(Volume, Volume Threshold) (for example A(5,12) means that User A has volume 5 and volume threshold 12), as well as specify where these users are, i.e $\{J_\Lambda(i)\}$.

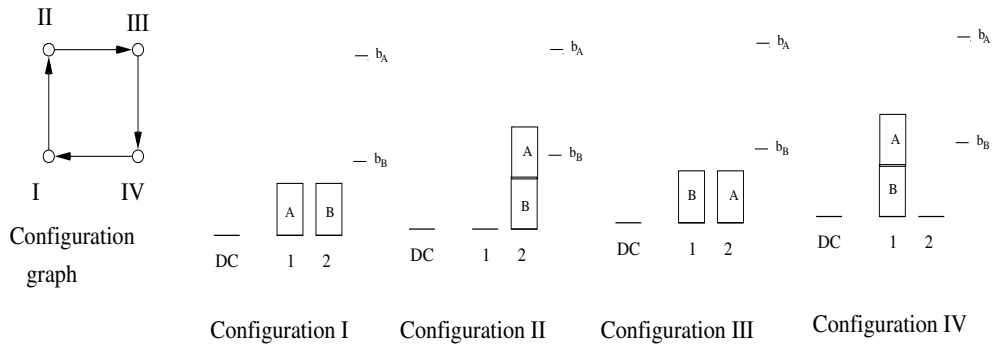


FIG. 2.2. *Game configuration graph and individual configurations*

3. Previously known properties of $\mathcal{Q}$. We list relevant properties of the class $\mathcal{Q}$ of games established in [15] concerning existence, optimality and complexity of computing Nash equilibria.

THEOREM 3.1. *Every game in $\mathcal{Q}$ has a generalized potential function and therefore every such game has a Nash Equilbrium.*

THEOREM 3.2. *For any $\epsilon > 0$ there is a game in $\mathcal{Q}$ that has price of anarchy and price of stability equal to ϵ.*

THEOREM 3.3. *A Nash Equilibrium that is also a System Optimum of a game in $\mathcal{Q}_{\mathcal{E}}$ can be found in time linear in the game parameters.*

THEOREM 3.4. *Any Nash Equilibrium of any game $G \in \mathcal{Q}_{\mathcal{E}}$ has communal welfare of at least a half of that of G's System Optimum.*

THEOREM 3.5. *For any initial configuration of every game in $\mathcal{Q}_{\mathcal{E}}$ there is a sequence of selfish moves by players that will terminate at Nash Equilibrium after $O(n^2)$ steps. This sequence can be determined by considering players in decreasing order of their volume thresholds and letting them make their selfish choices.*

4. New results. In this section we consider stability of games in class $\mathcal{PQ}$ and various properties of their Nash equilibria. Results will be compared to those of $\mathcal{Q}$ in Table 5.

We begin by establishing the following simple result about the prices of anarchy and stability of general games in the class $\mathcal{PQ}$, showing that they are not particularly well behaved.

THEOREM 4.1. *For any $\epsilon > 0$ there is a game in $\mathcal{PQ}$ that has a unique Nash equilibrium, whose communal welfare is ϵ, while the system optimum of this game has communal welfare equal to 1. This implies that prices of anarchy and stability of such a game are equal to ϵ.*

Proof. Consider a game with one non-DC class, and two players, $A(\epsilon, 1+\epsilon)$ and $B(1,1)$. The only equilibrium this game has is when player A is in class 1 and player B is in DC, as opposed to the system optimum when their positions are reversed. □

4.1. Approximate Nash equilibria. As we have noted in the Introduction and Figure 2.2, Nash equilibria do not necessarily exist in games $\mathcal{PQ}$ that involve pricing. One approach to examining such games involves *$\alpha-$approximate Nash equilibria*, defined in for example [4]. A configuration is said to be *$\alpha-$approximate Nash equilibrium* if no player can move and decrease her cost by more than an α multiplicative factor.

Note that since pricing functions of $\mathcal{PQ}$ are arbitrary decreasing linear functions, we will instead use a more appropriate notion of *$\delta-$approximate Nash equilibrium* instead, where δ is an *additive* factor.

Let $\mathcal{PQ}_{\mathcal{E}}$ be the subset of $\mathcal{PQ}$ where all players have volume $\epsilon = \delta$. In such a game a configuration where all players are satisfied and all classes have equal total volume would be a *$\epsilon-$approximate Nash equilibrium*, since no player would have an incentive to move.

When ϵ goes to zero and number of players goes to infinity, the class $\mathcal{PQ}_{\mathcal{E}}$ will be denoted as $\mathcal{PQ}_{\infty}$. This class of games has similar behavior to the class of games where players are allowed to split their volume between several classes.

THEOREM 4.2. *A Nash equilibrium ($\delta-$approximate Nash equilibrium) that is also system optimum can be constructed for any game in $\mathcal{PQ}_{\infty}$ ($\mathcal{PQ}_{\mathcal{E}}$) in time of $O(n)$.*

Proof. A greedy algorithm solves this problem. Here is the algorithm for $\mathcal{PQ}_{\mathcal{E}}$. Let $b_1 \leq \ldots \leq b_n$; place player n in class 1, place player $n-1$ in class 1 if $b_{n-1} \geq 2\epsilon$, otherwise place player $n-1$ in class 2; place player

$n-2$ in class 1 if $b_{n-2} \geq 3\epsilon$ etc. The resulting configuration is a system optimum and a $\delta-$approximate Nash equilibrium. □

Note that while the preceeding theorem guarantees existence of an approximate Nash equilibrium for games $\mathcal{PQ}_{\mathcal{E}}$, it does not promise that *every* sequence of selfish moves will arrive at an approximate Nash equilibrium. Consider the following observation, which also disproves existence of general potential functions for all games in $\mathcal{PQ}_{\mathcal{E}}$. This is also true for games in $\mathcal{PQ}_{\infty}$.

THEOREM 4.3. *There is a game in $\mathcal{PQ}_{\mathcal{E}}$ where there is a cycle of selfish moves.*

Proof. Let $\delta = 1$. Consider a game with 2 non-DC classes and 12 players:

$$A_1(1,9), A_2(1,9), A_3(1,9), B_1(1,6), B_2(1,6), B_3(1,6), C_1(1,3), \ldots, C_6(1,3).$$

Initial configuration Λ: players C_4, C_5 and C_6 are in class 2, all other players are in class 1. First players B_1, B_2 and B_3 move to class 2, after that players C_1, C_2, C_3 move to DC, then players A_1, A_2 and A_3 move to class 2 and finally players C_1, C_2, C_3 move from DC to class 1. The resulting configuration is essentially isomorphic to Λ, hence a cycle has occurred. □

Now we will examine properties of corresponding Nash equilibria.

THEOREM 4.4. *Price of anarchy of games in $\mathcal{PQ}_{\infty}$ is equal to 1/2. Price of stability of such games is equal 1.*

If price of anarchy and price of stability were redefined over ϵ-approximate Nash equilibria instead of regular Nash equilibria, then it would hold that price of anarchy of games in $\mathcal{PQ}_{\mathcal{E}}$ is equal to 1/2 and price of stability of such games is equal 1.

Proof. Price of stability follows from the fact that Nash equilibria constructed in Theorem 4.2 are system optima.

Price of anarchy can be demonstrated by following argument for games in $\mathcal{PQ}_{\mathcal{E}}$, and the proof for $\mathcal{PQ}_{\infty}$ is similar. Let Λ be a Nash equilibrium when all players have the same volume ϵ. Consider the unsatisfied player i that has the largest volume threshold b_i. (If there are no unsatisfied players then such a Nash equilibrium is a system optimum). Total traffic volume q_j in every class j is strictly greater than $b_i - \epsilon$, hence communal welfare of Λ is greater than or equal to $m(b_i - \epsilon)$ but communal welfare of system optimum cannot be more than $2(m(b_i - \epsilon))$. □

4.2. Finding a Nash equilibrium. It was shown in [16] that the problem of finding system optimum of a game in class $\mathcal{Q}$ is NP-Complete. It was also shown that the problem of finding a Nash equilibrium in $\mathcal{Q}$ can be solved in $O(n^2)$ time. Similarly the problem of finding a system optimum of a game in class $\mathcal{PQ}$ is NP-Complete. Now we will examine the problem of finding a Nash equilibrium (or determining that it does not exists) for games in $\mathcal{PQ}$.

THEOREM 4.5. *Problem of finding Nash equilibrium for games in $\mathcal{PQ}$ is NP-Complete.*

Proof. Consider the following version of MAXIMUM SUBSET SUM problem—given set $S = \{s_1, \ldots, s_n\}$ and targets t_1, t_2, find $A \subseteq S$ such that $t_1 \leq \sum_{i \in A} s_i \leq t_2$. This problem can be reduced to problem of finding a Nash equilibrium as follows. There are $n+1$ players and two non-DC classes. Players $1, \ldots, n$ all have same threshold $b_1 = b_2 = \ldots b_n = t_2$, individual volumes $\lambda_i = s_i$. Player $n+1$ has volume $\lambda_{n+1} = t_2$ and threshold $b_{n+1} = t_1 + t_2$. Then this game will have a Nash equilibrium if and only if the original MAXIMUM SUBSET SUM problem had a feasible solution. □

4.3. Price thresholds. In [16] it was shown that games in class $\mathcal{Q}$ will terminate in $O(n^2)$ steps, given certain assumptions on order of player moves. Here we will describe a computer experiment that examined speed of convergence of games where there was no such ordering of player moves.

This experiment involved a following natural assumption about players behavior. In practice, there could be a limit on how much a user is willing to pay, and this concept can be easily added to our games, resulting in the new classes of games. This concept has a desirable effect on the dynamics of the game, as explained below. Formally, for players i we define *price thresholds* (in addition to the old volume thresholds) t_i that have the following property. If the price in a class exceeds player i's price threshold, then player i is not satisfied. We assume that $b_i \leq b_j$ if and only if $t_i \geq t_j$, i.e users who demand better quality of service (smaller traffic volume in their class) are willing to pay more.

We conjecture that in addition to being realistic, such price thresholds also tend to improve the speed of convergence to Nash equilibria. This is because of players spending less time looping in non-terminal cycles.

To test this conjecture we ran a computer program simulating a game in class $\mathcal{PQ}$. Later we added pricing thresholds to the game which has considerably improved time lapsed before convergence to Nash equilibria. Game parameters were chosen such that Nash equilibrium would always exist. Parameters of the game were $M =$ number of classes, $M/T =$ number of *types* of users that have the same volume and volume threshold, $K =$ number of users of the same type that can fit in one class without exceeding their volume threshold. Volumes were in increments of one, i.e there are $T * K$ users that have volume 1 and volume threshold K, $T * K$ users that have volume 2 and threshold $2K$, $\ldots, T * K$ users that have volume M/T and threshold $M * K/T$. Thus there are a total of $M * K$ users. For example let $K = 10, M = 20, T = 5$. This means that there are 20 classes, 4 types of users and at most 10 users of any one type can fit into one class. Users are

$$A_1(1,10), \ldots, A_{50}(1,10), B_1(2,20), \ldots, B_{50}(2,20), C_1(3,30), \ldots, C_{50}(3,30),$$

$$D_1(4,40), \ldots, D_{50}(4,40).$$

Initially all users are in the dummy class (DC). A game proceeds by picking one of the $M * K$ users at random and this user moves either to the largest class where his threshold would not be exceeded or to the DC. Even if this move exceeds the volume threshold of some other users in the destination class of the moving user, these unsatisfied users cannot move until it is their turn to move and turns are determined at random. Eventually a Nash equilibrium was always reached, where all users of the first type were in T classes, all users of the second type were in the second set of T classes etc. Results are shown in table 4.1. "Moves1" denotes the total number of user moves until Nash equilibrium was reached.

Later a simulation of pricing thresholds was added to the experiment. Effectively it would prohibit a user i that has volume threshold b_i to move into any class j such that $q_j + \lambda_i < b_i - \Delta$ where Δ is some constant. The reason for this is that class j is too expensive for the i^{th} user.

Table 4.1

K	M	T	Moves1	Δ	Moves2
5	20	1	161,000	5	7,000
10	20	1	17,077,000	10	9,000
20	20	2	1,354,000	20	25,000
50	20	1	56,000	50	35,000
100	20	1	49,000	100	46,000
100	20	10	3,000	100	5,000
1000	20	10	35,000	1000	49000
5	40	1	2,360,000	5	190,000
5	50	1	8,391,000	5	940000

When $\Delta = \infty$ this is equivalent to the old experiment without pricing thresholds. In general introduction of small Δ significantly improved number of moves that was needed to reach the Nash equilibrium. See "Moves2" in the table 4.1.

5. Conclusions, Directions. Here we summarize known results about Nash Equilibria for various subclasses of $\mathcal{Q}$ and $\mathcal{PQ}$.

	NE/GenPotential always exists	Price of anarchy	Price of stability	Complexity of finding NE
$\mathcal{Q}$	Yes/Yes	ϵ	ϵ	$O(n^2)$
$\mathcal{Q}_\mathcal{E}$	Yes/Yes	1/2	1	$O(n)$
$\mathcal{PQ}$	No/No	ϵ	ϵ	NP-Complete
$\mathcal{PQ}_\mathcal{E}$	Yes/No	1/2	1	$O(n)$
$\mathcal{PQ}_\infty$	Yes/No	1/2	1	$O(n)$

Existence of Nash Equilbria for $\mathcal{Q}$ (and $\mathcal{Q}_\mathcal{E}$, since $\mathcal{Q}_\mathcal{E} \subset \mathcal{Q}$) is shown in Theorem 3.1. Example of nonexistence of Nash Equilbria in $\mathcal{PQ}$ is demonstrated in Figure 2.2. For $\mathcal{PQ}_\mathcal{E}$ entry "Yes" refers to $\delta-$approximate Nash Equilibria, not regular Nash Equilibria. This (and $\mathcal{PQ}_\infty$ case) is shown in Theorem 4.2. The nonexistence

of generalized potential functions for these classes is shown in Theorem 4.3. Prices of anarchy and stability of $\mathcal{Q}$ are shown in Theorem 3.2, of $\mathcal{Q}_{\mathcal{E}}$ in Theorem 3.4, of $\mathcal{PQ}$ in Theorem 4.1 (assuming that Nash Equilibrium exists), of $\mathcal{PQ}_{\mathcal{E}}$ and $\mathcal{PQ}_{\infty}$ in Theorem 4.4. Complexity of finding a Nash Equilibrium in games of class $\mathcal{Q}$ is shown in Theorem 3.5, case of $\mathcal{Q}_{\mathcal{E}}$ is Theorem 3.3, for games in $\mathcal{PQ}$ this problem is NP-Complete (Theorem 4.5), for games in $\mathcal{PQ}_{\mathcal{E}}$ and $\mathcal{PQ}_{\infty}$ result follows from Theorem 4.2.

5.1. Open questions. The class $\mathcal{PQ}$ contains both games that have Nash equibria and those who do not.

What is the structure of games in class $\mathcal{PQ}$ where Nash equilibria or approximate Nash equilibria (additive or multiplicative) are guaranteed to exist but they are hard to compute? For example, are there PLS-complete games in the class $\mathcal{PQ}$? For the subclasses such as $\mathcal{PQ}_{\mathcal{E}}$ Nash equilibria existence is easy to determine, and (approximate) Nash equilibria are easy to compute. Formally state and prove the conjecture of Section 4.3 concerning the usage of price thresholds and speed of convergence to Nash equilibria.

REFERENCES

[1] E. Altman, T. Boulogne, R. El-Azouzi, T. Jimenez and L. Wynter, *A survey on networking games in telecommunications*, Comput. Oper. Res., pages 286–311, 2006.

[2] E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler and T. Roughgarden, The Price of Stability for Network Design with Fair Cost Allocation, FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), pages 295–304, 2004.

[3] B. Awerbuch, Y. Azar and A. Epstein, *Large The price of routing unsplittable flow*, STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, pages 57–66, 2005.

[4] H. Chen and T. Roughgarden, *Network design with weighted players*, ACM Symposium on parallel algorithms and architecture, pages 29-38, 2006.

[5] R. Cocchi and D. Estrin and S. Shenker, *Pricing in Computer Networks: Motivation*, IEEE/ACM Transactions on Networking, Vol. 1, No. 6, December 1993, pages 614-627.

[6] R. Cole, Y. Dodis and T. Roughgarden, *Pricing network edges for heterogeneous selfish users*, STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, pages 521–530, 2003.

[7] A. Fabrikant, C. Papadimitriou and K. Talwar, *The complexity of pure Nash equilibria*, STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, pages 604–612, 2004.

[8] L. Fleischer, K. Jain and M. Mahdian, *Tolls for Heterogeneous Selfish Users in Multicommodity Networks and Generalized Congestion Games*, FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), pages 277–285, 2004.

[9] L. Fleischer, *Linear tolls suffice: new bounds and algorithms for tolls in single source networks*, Theor. Comput. Sci., (348), pages 217–225, 2005.

[10] D. Fotakis, S. C. Kontogiannis, E. Koutsoupias, M. Mavronicolas and P. G. Spirakis, *The Structure and Complexity of Nash Equilibria for a Selfish Routing Game*, ICALP '02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming, pages 123–134, 2002.

[11] D. Fotakis, S. Kontogiannis, P. Spirakis, *Selfish unsplittable flows*, Theor. Comput. Sci., (348), pages 226–239, 2005.

[12] Y. Korilis, A. Lazar, and A. Orda, *Architecting noncooperative networks*, IEEE Journal of Selected Areas in Communications, (13) pages 1241–1251, 1995.

[13] E. Koutsoupias and C. H. Papadimitriou, *Worst-case equilibria*, Symposium on Theoretical Aspects of Computer Science, pages 404–413, 1999.

[14] ———, *Optimal transport strategies for best-effort traffic over priced connections*, Technical report, Technion, 2000.

[15] A. Lomonosov, M. Sitharam and K. Park, *Stability vs optimality tradeoff in game theoretic mechanisms for QoS provision*, SAC '03: Proceedings of the 2003 ACM symposium on Applied computing, pages 28–32, 2003.

[16] ———, *Network QoS games: stability vs optimality tradeoff*, J. Comput. Syst. Sci., (69), pages 281–302, 2004.

[17] I. Milchtaich, *Congestion games with player-specific payoff functions*, Games and Economic Behavior (13), pages 111–124, 1996.

[18] D. Monderer and L. Shapley, *Potential Games*, Games and Economic Behavior (14), pages 124–143, 1996.

[19] N. Nisan and A. Ronen, *Algorithmic mechanism design*, Proc. 31st ACM Symp. on Theory of Computing, pages 129–140, 1999.

[20] A. Orda, R. Rom, and N. Shimkin, *Competitive routing in multiuser communication networks*, IEEE/ACM Transactions on Networking, (1), pages 510–521, 1993.

[21] ———, *Computing correlated equilibria in multi-player games*, STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, pages 49–56, 2005.

[22] C. Papadimitriou and T. Roughgarden, *Computing equilibria in multi-player games*, SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, pages 82–91, 2005.

[23] K. Park, M. Sitharam, and S. Chen, *Quality of service provision in noncooperative networks: Heterogeneous preferences*, Proceedings of the First Int. Conf. on Information and Computation Economics ICE'98, 1998.

[24] K. Park, M. Sitharam, and S. Chen, *Quality of service provision in noncooperative networks with diverse user requirements*, Decision Support Systems, Special Issue on Information and Computation Economies, vol. 28, pages 101–122, 2000.

[25] J. B. Rosen, *Existence and uniqueness of equilibrium points for concave n-person games*, Econometrica, pages 520-534, 1965.

[26] R. W. Rosenthal, *A Class of games possessing pure strategy nash equilibria*, International Journal of Game Theory, pages 65–67, 1973.
[27] T. Roughgarden, *Selfish Routing and the Price of Anarchy*, The MIT Press, 2005.
[28] T. Roughgarden and E. Tardos, *How bad is selfish routing?* IEEE Symposium on Foundations of Computer Science, pages 93–102, 2000.
[29] ———, *Mechanism design and the Internet*, Presentation in DIMACS Workshop on Computational Issues in Game Theory and Mechanism Design, 2001.

Edited by: Marcin Paprzycki, Niranjan Suri
Received: October 1, 2006
Accepted: December 10, 2006

Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 87–100. http://www.scpe.org
ISSN 1895-1767

THE SUCCESS OF COOPERATIVE STRATEGIES IN THE ITERATED PRISONER'S DILEMMA AND THE CHICKEN GAME

BENGT CARLSSON* AND K. INGEMAR JÖNSSON†

Abstract. The prisoner's dilemma has evolved into a standard game for analyzing the success of cooperative strategies in repeated games. With the aim of investigating the behavior of strategies in some alternative games we analyzed the outcome of iterated games for both the prisoner's dilemma and the chicken game. In the chicken game, mutual defection is punished more strongly than in the prisoner's dilemma, and yields the lowest fitness. We also ran our analyses under different levels of noise. The results reveal a striking difference in the outcome between the games. Iterated chicken game needed more generations to find a winning strategy. It also favored nice, forgiving strategies able to forgive a defection from an opponent. In particular the well-known strategy tit-for-tat has a poor successrate under noisy conditions. The chicken game conditions may be relatively common in other sciences, and therefore we suggest that this game should receive more interest as a cooperative game from researchers within computer science.

Key words. Game theory, prisoner's dilemma, chicken game, noise, tit-for-tat

1. Introduction. Within computer science, biology, social and economic sciences the issue of cooperation between individuals in an evolutionary context is widely discussed. An evolutionary context means some conflict of interest between the participants preferrably modeled in a game theoretical context using conflicting games. A simple, but frequently used, game model is between two participants each with two choices, either to cooperate or to defect (a $2*2$ matrix game) played once or repeated. In multi agent systems iterated games have become a popular tool for analyzing social behavior and cooperation based on reciprocity ([3, 5, 4, 9]). By allowing games to be played several times and against several other strategies a "shadow of the future", i. e. a non-zero probability for the agents to meet again in the future, is created for the current game. This increases the opportunity for cooperative behavior to evolve (e.g., [4]). A collection of different models of cooperation and altruism was discussed in Lehmann and Keller [14].

Most iterative analyses on cooperation have focused on the payoff environment defined as the prisoner's dilemma (PD) ([5, 9, 13, 20]). In terms of payoffs, a PD is defined when $T > R > P > S$, where R = reward, S = sucker, T = temptation and P = punishment. It should also hold that $2R > T + S$ according to table 1.1a. The second condition means that the value of the payoff, when shared in cooperation, must be greater than it is when shared by a cooperator and a defector. Because it pays more to defect, no matter how the opponent chooses to act, an agent is bound to defect, if the agents are not deriving advantage from repeating the game. If $2R < T + S$ is allowed there will be no upper limit for the value of the temptation. However, there is no definite reason for excluding this possibility. Carlsson and Johansson [11] argued that Rapoport and Chammah [23] introduced this constraint for practical more than theoretical reasons. PD belongs to a class of games where each player has a dominating strategy of playing defect in the single play PD.

Chicken game (CG) is a similar but much less studied game than PD, but see Tutzauer et al. [26] for a recent study. CG is defined when $T > R > S > P$, i. e. mutual defection is punished more in the CG than in the PD. In the single-play form, the CG has no dominant strategy (although it has two Nash equilibria in pure strategies, and one mixed equilibrium), and thus no expected outcome as in the PD [16]. Together with the generous chicken game (GCG), also called the battle of sexes [17] or coordination game, CG belongs to a class of games where neither player has a dominating strategy. For a GCG, playing defect increases the payoff for both of them, unless the other agent also plays defect ($T > S > R > P$).

In table 1.1b, R and P are assumed to be fixed to 1 and 0 respectively. This can be obtained through a two steps reduction where all variables are first subtracted by P and then divided by $R - P$. This makes it possible to describe the games with only two parameters $S' = (S - P)/(R - P)$ and $T' = (T - P)/(R - P)$. In fact we can capture all possible $2x2$ games in a two-dimensional plane.

In figure 1.1 the parameter space for PD, CG and GCG defined by S' and T', is shown. $T' = 1$ marks a dividing line between conflict and cooperation. $S' = 0$ marks the line between CG and PD. $T' < 1$ means that playing cooperate (R) is favored over playing defect (T) when the other agent cooperates. This prevents an

*School of Engineering, Blekinge Institute of Technology, S-372 25 Ronneby, Sweden, +46 457 385813, bengt.carlsson@bth.se
†Department of Mathematics and Sciences, Kristianstad University, S-291 88 Kristianstad, Sweden. +46 44 203429, ingemar.jonsson@mna.hkr.se

TABLE 1.1
Pay-off matrices for 2∗2 games where R = reward, S = sucker, T = temptation and P = punishment. In b the four variables R, S, T and P are reduced to two variables $S' = (S-P)/(R-P)$ and $T' = (T-P)/(R-P)$

a	Cooperate	Defect	b	Cooperate	Defect
Cooperate	R	S	Cooperate	1	$(S-P)/(R-P)$
Defect	T	P	Defect	$(T-P)/(R-P)$	0

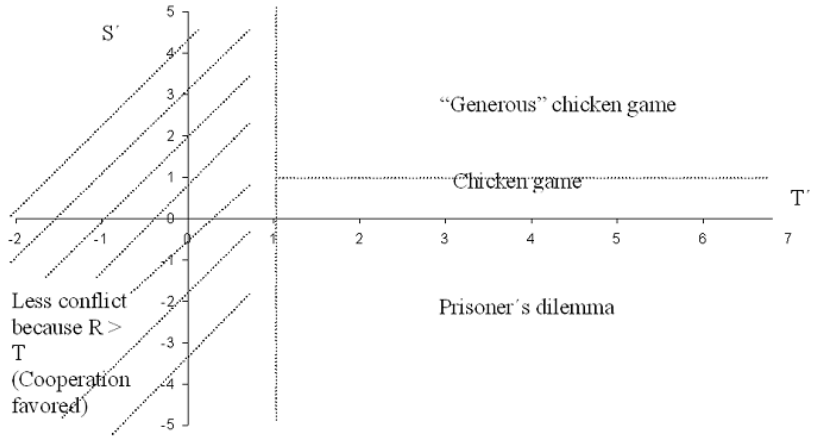


FIG. 1.1. *The areas covered by three kinds of conflicting games in a two-dimensional plane: prisoner's dilemma, chicken game and generous chicken game*

agent from being "selfish" in a surrounding of cooperation. Conflicting games are expected when $T' > 1$ because of better outcome playing temptation (T).

In an evolutionary context, the payoff obtained from a particular game represents the change in fitness (reproductive success) of a player. Maynard Smith [18] describes an evolutionary resource allocation within a $2x2$ game as a hawk and dove game. In the matrices of table 1.1 a hawk constitutes playing D, and a dove constitutes playing C. A hawk gets all the resources playing against a dove. Two doves share the resource whereas two hawks escalate a fight about the resource. If the cost of obtaining the resource for the hawks is greater than the resource there is a CG, otherwise there is a PD. In a generous CG (not a hawk and dove game) more resources are obtained for both agents when one agent defects compared to both playing cooperate or defect.

Recent analyses have focused on the effects of mistakes in the implementation of strategies. In particular, such mistakes, usually called noise, may allow evolutionary stability of pure strategies in iterated games [9]. Two separate cases are generally considered: the trembling hand noise and misinterpretations. Within the trembling hand noise ([24, 4]) a perfect strategy would take into account that agents occasionally do not perform the intended action[1]. In the misinterpretations case an agent may not have chosen the "wrong" action. Instead it is interpreted as such by at least one of its opponents, resulting in agents keeping different opinions about what happened in the game. This introduction of mistakes represents an important step, as real biological systems as well as computer systems will usually involve uncertainty at some level.

Here, we study the behavior of strategies in iterated games within the prisoner's dilemma and chicken game payoff structures, under different levels of noise. We first give a background to our simulations, including a round robin tournament and a characterization of the strategies that we use. We then present the outcome of iterated population tournaments, and discuss the implications of our results for game theoretical studies on the evolution of cooperation.

[1] In this metaphor an agent chooses between two buttons. The trembling hand may, by mistake, cause the agent to press the wrong button

2. Games, Strategies, and Simulation Procedures.

2.1. Games. A game can be modeled as a strategic or an extensive game. A strategic game is a model of a situation in which each agent chooses his plan of action once and for all, and all agents' decisions are made simultaneously while an extensive game specifies the possible orders of events. The strategic agent is not informed of the plan of action chosen by any other agent while an extensive agent can consider its plan of action whenever a decision has to be made. All the agents in our analyses are strategic. All strategies may affect the moves of the other agent, i. e. to play C or D, but not the payoff value, so the latter does not influence the strategy. The kind of games that we simulate here have been called ecological simulations, as distinguished from evolutionary simulations in which new strategies may arise in the course of the game by mutation ([3]). However, ecological simulations include all components necessary for the mimicking of an evolutionary process: variation in types (strategies), selection of these types resulting from the differential payoffs obtained in the contests, and differential propagation of strategies over generations. Consequently, we find the distinction between ecological and evolutionary simulations based on the criteria of mutation rather misleading.

The PDs and CGs that we analyze are repeated games with memory, usually called iterated games. In iterated games some background information is known about what happened in the game up to now. In our simulation the strategies know the previous moves of their antagonist[2]. In all our simulations, interactions among players are pair-wise, i. e. a player interacts with only one player at a time

2.2. Nice and Mean Strategies. Axelrod ([1, 5, 2, 3]) categorized strategies as nice or mean. A nice strategy never plays defection before the other player defects, whereas a mean strategy never plays cooperation before the opponent cooperates. Thus the nice and mean terminology describes an agent's next move.

According to the categorization of Axelrod Tit-for-tat, TfT, is a nice strategy, but it could as well be regarded as a repeating strategy. Another category of strategies is a group of forgiving strategies consisting of Simpleton, Grofman, and Fair. They can, unlike TfT, avoid getting into mutual defection by playing cooperate. If the opponent does not respond to this forgiving behavior they start to play defect again. Finally we separate a group of revenging strategies, which retaliate a defection at some point of the game with defection for the rest of the game. Friedman and Davis belong to this group of strategies.

The principle for the categorization of strategies into nice and forgiving against defecting strategies, which use threats and punishments, is unclear. For instance, why is TfT not just treated as a strategy repeating the action of the other strategy instead?

2.3. Generous and Greedy Strategies. One alternative way of categorizing strategies is to group them together as being generous, even-matched, or greedy ([11, 10]). If a strategy more often plays as a sucker, n_S, than playing temptation, n_T, then it is a generous strategy $n_S > n_T$. An even-matched strategy has $n_S \approx n_T$ and a greedy strategy has $n_S < n_T$ where n_S and n_T are the proportion an agent plays sucker and temptation, respectively.

Boerlijst, et al [8] uses a similar categorization into good or bad standings. An agent is in good standing if it has cooperated in the previous round or if it has defected while provoked, i. e., if the agent is in good standing it should not be greedy unless the other agent was greedy the round before. In every other case of defection the agent is in bad standing, i. e. it tries to be greedy. The generous and greedy categorization uses a stable approach, a once and for all categorization[3], contrary to the more dynamic good and bad standing dealing with what happened in the previous move.

The stable approach of the generous and greedy categorization makes it easier to analyze this model. The basis of the partition is that it is a zero-sum game at the meta-level in that the sum of proportions of the strategies n_S must equal the sum of the strategies n_T. In other words, if there is a generous strategy, then there must also be a greedy strategy.

The classification of a strategy can change depending on the surrounding strategies. Let us assume we have the following four strategies:

- Always Cooperate (AllC) has 100 per cent co-operate $n_R + n_S$ when meeting another strategy. AllC will never act as a greedy strategy.
- Always Defect (AllD) has 100 percent defect $n_T + n_P$ when meeting another strategy. AllD will never act as a generous strategy.

[2]One of the strategies, Fair, also remembers its own previous moves
[3]For a certain set of strategies

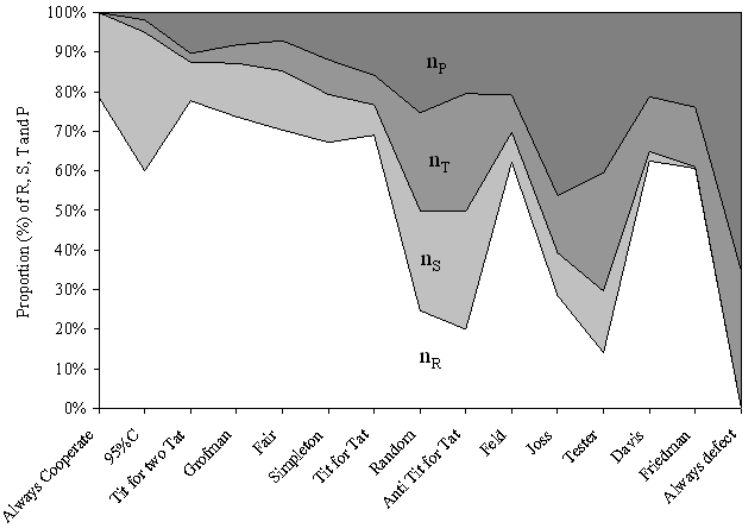


FIG. 2.1. *Proportions of R, S, T and P for different strategies. There is a generous strategy if $n_S > n_T$ and a greedy strategy if $n_S < n_T$*

- Tit-for-tat (TfT) always repeats the move of the other contestant, making it a repeating strategy. TfT naturally entails that $n_S \approx n_T$.
- Random plays cooperate and defect approximately half of the time each. The proportions of n_S and n_T will be determined by the surrounding strategies.

Random will be a greedy strategy in a surrounding of AllC and Random, and a generous strategy in a surrounding of AllD and Random. Both TfT and Random will behave as an even-matched strategy in the presence of only these two strategies as well as in a surrounding of all four strategies, with AllC and AllD participating in the same proportions. All strategies are even-matched when there is only a single strategy left.

The strategies used in our iterated prisoner's dilemma (IPD) and iterated chicken game (ICG), in all 14 different strategies plus playing Random, are presented in table 2.1. AllC, AllD and Random do not need any memory function at all because they always do the same thing (which for Random means always randomize). TfT and ATfT need to look back one move because they repeat or reverse the move of its opponent. Most of the other strategies also need to look back one move but may respond to defection or show forgiveness.

AllC definitely belongs to a group of generous strategies and so do 95% Cooperate (95%C), tit-for-two-tats (Tf2T), Grofman, Fair, and Simpleton, in this specific environment.

The even-matched group of strategies includes TfT, Random, and Anti-tit-for-tat (ATfT).

Within the group of greedy strategies, Feld, Davis, and Friedman belong to a smaller family of strategies doing more co-operation moves than Random, i. e. having significantly more than 50 % R or S. An analogous family consists of Joss, Tester, and AllD. These strategies co-operate less frequently than does Random.

What will happen to a particular strategy depends both on the surrounding strategies and on the characteristics of the strategy. For example, AllC will always be generous while 95%C will change to a greedy strategy when these two are the only strategies left. The described relation between strategies is independent of what kind of game is played, but the actual outcome of the game is related to the payoff matrix.

2.4. Simulation Procedures. The set of strategies used in our first simulation includes some of Axelrod's original strategies and a few, later reported, successful strategies. Of course, these strategies represent only a very limited number of all possible strategies. However, the emphasis in our work is on differences between IPD and ICG. Whether there exists a single "'best of the game"' strategy is outside the scope of our analyses.

Mistakes in the implementation of strategies (noise) were incorporated by attaching a certain probability p between 0.02 and 20% to play the alternative action (C or D), and a corresponding probability $(1-p)$ to play the original action.

TABLE 2.1
Description of the different strategies used in the first simulation (see section 3.1)

Strategy	*First move*	*Description*
AllC	C	Cooperates all the time
95%C	C	Cooperates 95% of the time
Tf2T	C	tit-for-two-tats, Cooperates until its opponent defects twice, and then defects until its opponent starts to cooperate again
Grofman	C	Cooperates if R or P was played, otherwise it cooperates with a probability of 2/7
Fair	C	A strategy with three possible states, - 'satisfied' (C), 'apologizing' (C) and 'angry' (D). It starts in the satisfied state and cooperates until its opponent defects; then it switches to its angry state, and defects until its opponent cooperates, before returning to the satisfied state. If Fair accidentally defects, the apologizing state is entered and it stays cooperating until its opponent forgives the mistake and starts to cooperate again
Simpleton	C	Like Grofman, it cooperates whenever the previous moves were the same, but it always defects when the moves differed (e.g.S)
TfT	C	Tit-for-tat. Repeats the moves of the opponent
Feld	C	Basically a tit-for-tat, but with a linearly increasing (from 0 with 0.25% per iteration up to iteration 200) probability of playing D instead of C
Davis	C	Cooperates on the first 10 moves, and then, if there is a defection, it defects until the end of the game
Friedman	C	Cooperates as long as its opponent does so. Once the opponent defects, Friedman defects for the rest of the game
ATfT	D	Anti-tit-for-tat. Plays the complementary move of the opponent
Joss	C	A TfT-variant that cooperates with a probability of 90%, when opponent cooperated and defects when opponent defected
Tester	D	Alters D and C until its opponent defects, then it plays a C and TfT
All D	D	Defects all the time

Our population tournament involves two sets of analyses. In the first set, the strategies are allowed to compete within a round robin tournament with the aim of obtaining a general evaluation of the tendency of different strategies to play cooperate and defect. In a round robin tournament, each strategy is paired once with all other strategies plus its twin. The results from the round robin tournament are used within the population tournament but will not be presented here (for the results see [10]). In the second set, the competitive abilities of strategies in iterated population tournaments were studies within the IPD and the ICG. We also conducted a second simulation of the IPD and the ICG where two sets of strategies were used. We used the strategies in figure 2.2 represented by finite automata [15]. The play between two automata is a stochastic process where all finite memory strategies can be represented by increasingly complicated finite automata. Memory-0 strategies, like AllC and AllD, do not involve any memory capacity at all. If the strategy in use only has to look back at one draw, there is a memory-1 strategy (a choice between two circles dependent of the other agent's move). All the strategies in figure 2.2 belong to memory-0 or memory-1 strategies.

Both sets of strategies include AllD, AllC, TfT, ATfT and Random. In the first set of strategies, the cooperative-set five AllC variants (100, 99.99, 99.9, 99 and 90% probability of playing C) are added. In the second set of strategies, the defective-set the corresponding five AllD variants (100, 99.99, 99.9, 99 and 90%

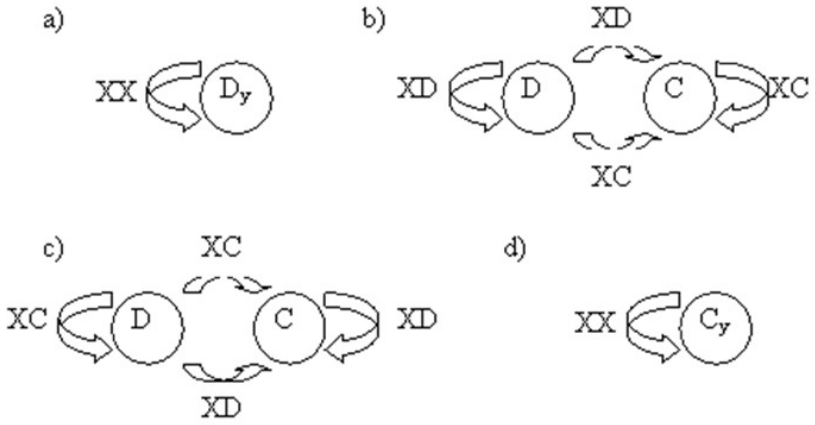


FIG. 2.2. *a) AllD (and variants) b) TfT c) ATfT d) AllC (and variants). On the transition edges, the left symbol correspond to an action done by a strategy against an opponent performing the right symbol, where an X denotes an arbitrary action. Y in Cy and Dy denotes a probability factor for playing C and D respectively*

probability of playing D) are added. C_y and D_y in figure 2.2 show a probability factor y 100, 99.99, 99.9, 99, 90% or for the Random strategy 50% for playing C and D respectively.

3. Population Tournament With Noise.

3.1. First Simulation. We evaluated the strategies in table 2.1 by allowing them to compete within a round robin tournament.

To obtain a more general treatment of IPD and ICG, we used several variants of payoff matrices within these games, based on the general matrix of table 3.1. In this matrix, C stands for cooperate; D for defect and q is a cost variable.

TABLE 3.1

Payoff values used in our simulation. q is a cost parameter. $0 < q < 0.5$ defines a prisoner's dilemma game, while $q > 0.5$ defines a chicken game

	Player 2	
Player 1	C	D
C	1.5	1
D	2	1.5 - q

The payoff for a D agent playing against a C agent is 2, while the corresponding payoff for a C agent playing against a D agent is 1, etc. Two C agents share the resource and get 1.5 each.

The outcome of a contest with two D agents depends on q. For $0 < q < 0.5$, a PD game is defined, and for $q > 0.5$ we have a CG. Simulations were run with the values for $(1.5 - q)$ set to 1.4 and 1.1 for PD, and to 0.9, 0.6, and 0.0 for the CG (these values are chosen with the purpose to span a wide range of the games but are otherwise arbitrarily chosen). We also included Axelrod's original matrix Ax ($R = 3, S = 0, T = 5$ and $P = 1$) and a compromise dilemma game CD ($R = 2, S = 2, T = 3$ and $P = 1$). A CD is located on the borderline between the CG area and the generous CG area. In the discussion part we also compare the mentioned strategies with a coordination game CoG ($R = 2, S = 0, T = 0$ and $P = 1$), the only game with $T' < 1$. CoG is included as a reference game and does not belong to the conflicting games. In figure 3.1 all these games are shown within the two-dimensional plane. The CD is closely related to the chicken game and CoG is a game with two Nash equilibria, playing (C,C) or playing (D,D) (see also Johansson et

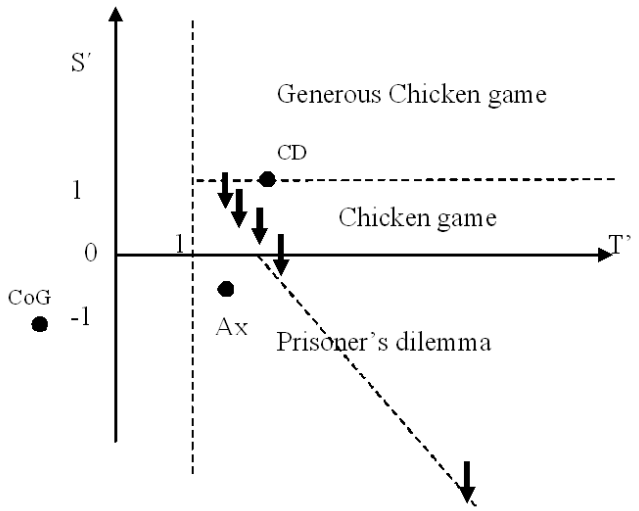


FIG. 3.1. *The different game matrices represented as dots in a 2-dimensional diagram. CoG is the coordination game, CD the compromise dilemma and Ax is the original Axelrod game. The unmarked dots represent 0.0, 0.6, 0.9, 1.1 and 1.4 from upper left to lower right*

al. [12]). Each game in the tournament was played on average 100 times (randomly stopped)[4] and repeated 5000 times.

In the second part of the simulation, strategies were allowed to compete within a population tournament for the iterated games. These simulations were based on the same payoff matrices for IPD and ICG as in the initial round robin tournament. Based on the success in the single round-robin tournaments, strategies were allowed to reproduce copies into the next round robin tournament, creating a population tournament, i. e. a quality competition in the round-robin tournament (make a good score) is transformed to an increased number of copies in the population tournament. Each of the fifteen strategies starts with 100 copies resulting in a total population of 1500. The number of copies for each strategy changes, but the total of 1500 copies remains constant. The proportions of the different strategies propagated into a new generation were based on the payoff scores obtained in the preceding round-robin tournament. A given strategy interacts with the other strategies in the proportions that they occur in their global population. The games were allowed to continue until a single winning strategy was identified, i. e. the whole population consists of the same strategy, or until the number of generations reached 10,000. In most of the simulations, a winning strategy was found before reaching this limit.

Also, if a pure population of agents with the random strategy are allowed to compete with each other in a population game, a single winning strategy will be found after a number of generations, i. e. there are small simulation variations between different agents in their actual play of C and D moves. As seen in figure 3.2, with increased total population size of agents the number of generations for finding a winning strategy increases. This almost linear increase ($r = 0.99$) is only marginally dependent of what game is played.

Randomized strategies with 100 individuals are according to figure 3.2 supposed to halt, i. e. all 1500 individuals belong to the same initial strategy, after approximately 2800 generations in a population game. Which strategy that wins will vary between the games. There are two possible kinds of winning strategies: pure strategies that halt, and mixed strategies (two or more pure strategies) that do not halt. If there is an active choice of a pure strategy it should halt before 2800 generations, because otherwise playing random could be treated as a winning pure strategy. There is no reason to believe that a single strategy winner should be found by extending the simulation beyond 10000 generations. If there exists a pure solution, this solution should turn up much earlier.

The effect of uncertainty (noise) in the choice of actions (C or D) by the agents within the tournaments was analyzed by repeating the tournaments in environments of varying levels of noise. Tournaments were run

[4]If an agent knows exactly or with a certain probability when a game will end, it may use such information to improve its behavior. Because of this, the length of the games was determined probabilistic, with an equal chance of ending the game with each given move (see also [1])

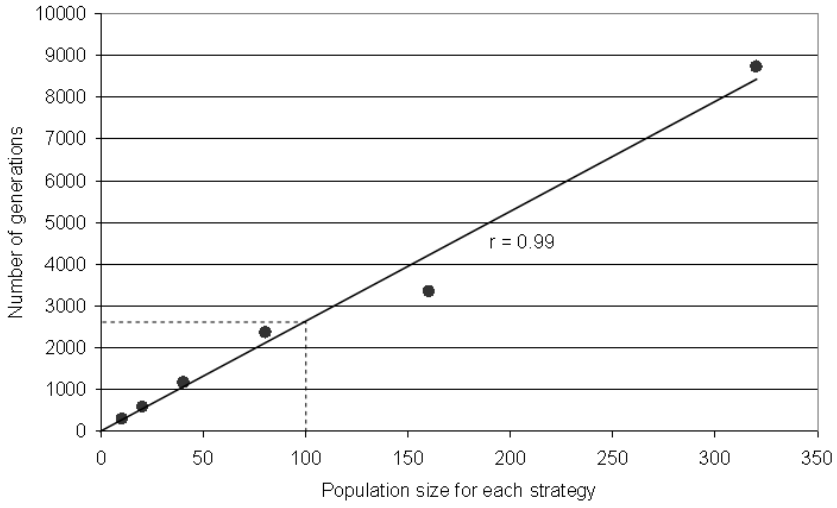


FIG. 3.2. *Number of generations for finding a winning strategy among 15 random strategies with a varying population size*

at 0, 0.02, 0.2, 2, and 20% noise. The probability of making a mistake was neither dependent on the sequence of behaviors up to a certain generation, nor on the identity of the player. Noise will affect the implementation of all strategies except for the strategy Random. We focused on three different aspects when comparing the IPDs and ICGs, which will be further analyzed in the discussion part:

1. The number of generations for finding a winning strategy.
2. Differences in robustness for the investigated strategies.
3. The behavior of the, generally regarded, cooperative strategy TfT in IPD and ICG.

3.2. Second Simulation. To obtain a more general treatment of IPD and ICG, we used several variants of payoff matrices within these games, based on the general matrix of table 3.2.

TABLE 3.2
A payoff matrix for PD and CG. C stands for cooperate, D for defect, and s_1 and s_2 are cost variables. If $s_1 > 1$ it is a PD. If $s_1 < 1$ it is a CG

	Cooperate (C)	Defect (D)
Cooperate (C)	1	1-s_1
Defect (D)	1+s_2	0

In the first set of simulations we investigated the successfulness of the agents using different strategies (one strategy per agent) in a round-robin tournament. Since this is independent of the actual payoff value, the same round-robin tournament can be used for both IPD and ICG. Every agent was paired with all the other agents plus a copy of itself. Every meeting between agents in the tournament was repeated on average 100 times (randomly stopped) and played for 5000 times.

The result from the two-by-two meetings between agents using different strategies in the round robin tournament was used in a population tournament. The tournament starts with a population of 100 agents for each strategy, making a total population of 900. The simulation halts when there is a winning strategy (all 900 agents use the same strategy) or when the number of generations exceeds 10.000. Agents are allowed to change strategy and the population size remains the same during the whole contest. For the IPD the following parameters were used: $s_1 \in \{1.1, 1.2 \ldots 2.0\}$ and $s_2 \in \{0.1, 0.2 \ldots 1.0, 2.0\}$, making a total of 110 different games. For the ICG games with parameter settings $s_1 \in \{0.1, 0.2 \ldots 0.9\}$ and $s_2 \in \{0.1, 0.2 \ldots 1.0, 2.0\}$ a total of 99 different games were run. Each game is repeated during 100 plays and the average success is calculated for each strategy. For each kind of game there is both the cooperative set and the defective set explained in section 2.4.

4. Results.

4.1. First Simulation. In figure 4.1 and figure 4.2 the success of individual strategies in IPD, ICG and CD population games at no noise and 0.2% of noise are shown. The repeating strategy TfT is represented by a solid line, the generous strategies Simpleton, Grofman, and Fair by dashed lines, and the greedy strategies Friedman and Davis by dotted lines.

In the IPD games TfT, Friedman and Davis are the most successful with no noise (figure 4.1), while TfT, Grofman, Fair and Friedman are the most successful with 0.2% noise (figure 4.2). For the other levels of noise (not shown in figures) TfT, and for Axelrod's matrix also Tf2T, is dominating with 0.02%. With 2% noise Davis and TfT dominates, and finally AllD and Friedman are the dominating strategies with 20% noise.

At no noise all three groups of strategies are approximately equally successful in ICG (figure 4.1), with a minor advantage for the generous strategies Simpleton, Grofman, and Fair. This advantage increases with increasing noise. The greedy strategies Friedman and Davis disappear at 0.02% noise and TfT at 0.2% noise (figure 4.2) leaving the generous strategies alone at 0.2% and 2% noise. At 20% noise AllD supplements the set of successful strategies.

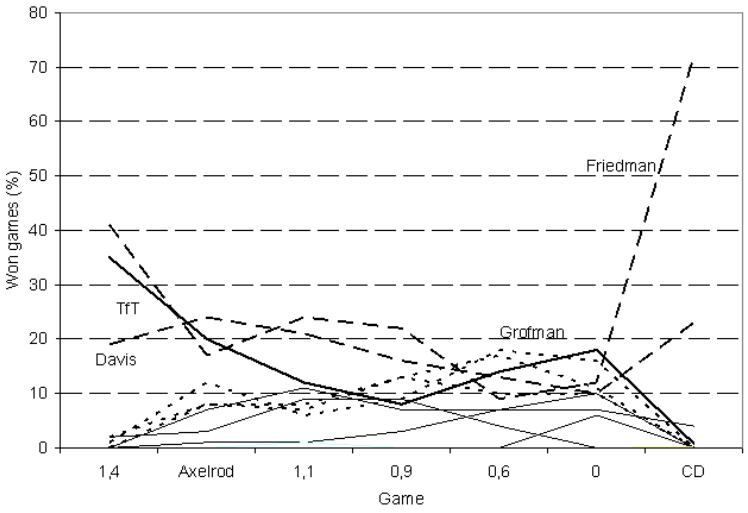


Fig. 4.1. *Percentage of runs won by strategies in the population games for different chicken games (0.9, 0.6, 0), prisoners dilemmas (1.4, Ax, 1.1) and the compromise dilemma with 0% noise*

The greedy strategies Friedman and Davis completely outperform Simpleton, Grofman, Fair and TfT strategies in CD. With increasing noise ATfT (0.2-20% noise) and AllD (20% noise) become more successful as part of a mixed set of strategies, because CD does not find a single winner (Figure 10).

Finally, in CoG Tf2T and TfT are dominating with 0% noise. Tf2T together with AllC and Grofman constitute all the winning strategies with 0.02%, 0.2% and 2% noise. 95%C is the only winner with 20% noise.

With increased noise the group of Simpleton, Grofman, and Fair become more and more successful in ICG up to and including 2% noise. When noise is introduced, IPDs favor the repeated TfT. With increased noise the greedy Friedman and Davis disappears for both ICG and IPD. Finally, with 20% noise AllD is the dominating strategy. More and more defecting strategies will dominate with increasing noise in IPD. Finally in CD the greedy strategies Friedman and Davis dominates. In contrast to IPD and CD cooperating and generous strategies dominate in ICG which makes the ICG the best candidate for finding robust strategies.

On average there was 80% accordance (for all levels of noise) between winning strategies in different ICG, i. e. four out of five strategies being the same. In the IPD there was a discrepancy with only on average 35% of the winning strategies being the same. The performance of the 0.4 and Ax matrices are similar within the ICG. This was especially notable for both matrices without noise (on average 75%) and for the 0.4 matrices with 2 and 20% noise (on average 55%).

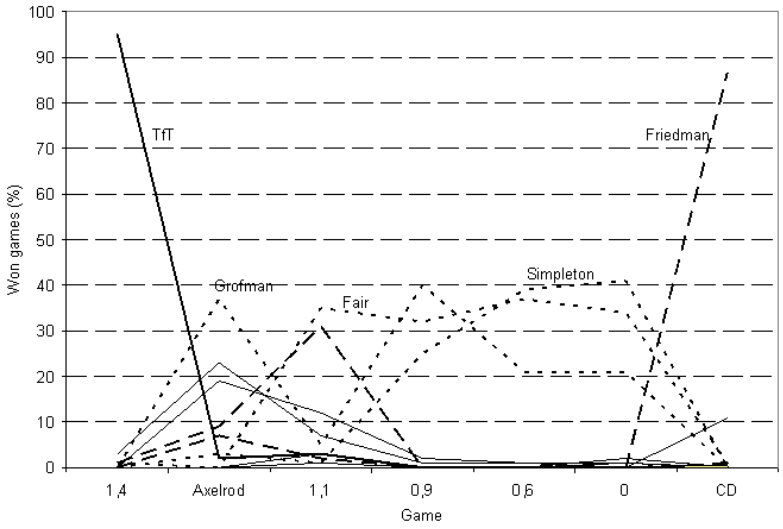


FIG. 4.2. *Percentage of runs won by strategies in the population games for different chicken games (0.9, 0.6, 0), prisoners dilemmas (1.4, Ax, 1.1) and the compromise dilemma with 0.2% noise*

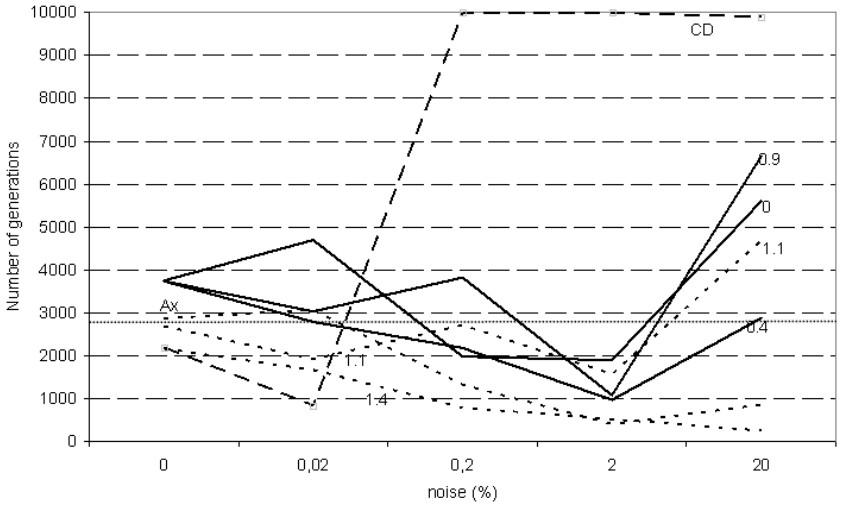


FIG. 4.3. *Number of generations for finding a winning strategy in chicken games, prisoners dilemmas and compromise dilemma at different levels of noise*

In figure 4.3, the number of generations needed to find a winning strategy is plotted for different level of noise. The dotted line shows the expected generations (2800) for competing Random strategies mentioned earlier. At 0 or low levels of noise more generations are needed in the ICG for finding a winner than in IPD. The lowest numbers of generations are needed with 2% of noise and the highest with 0% and 20% noise. There is no single strategy winner for the CD game with 0.2% noise and above

In summary; coordination games give mutual cooperation the highest results, which favors nice, but to a less extent too forgiving, strategies. Compared to the ICG, IPD is less punishing towards mutual defection,

TABLE 4.1
The difference between pure and mixed-strategies in IPD and ICG. For details see text

	IPD		ICG	
	Cooperative set	Defective set	Cooperative set	Defective set
Pure strategies	TfT 78% AllD 20%	TfT 75% AllD 20%	TfT 3%	TfT 2%
Mixed strategies	none	none	2-strat 61% 3-strat 33%	2-strat 69% 3-strat 24%

which allows repeating and greedy strategies to become more successful. Finally in the compromise dilemma, where playing the opposite to the opponent is favored, greedy and/or a mixture of different strategies are favored. With increased noise (2% or below), generous strategies become more and more successful in ICG while repeating and greedy strategies are more successful in IPD.

4.2. Second Simulation. In a surrounding of a cooperative or a defective set of strategies a major difference between pure and mixed strategies for IPD and ICG are shown in table 4.1. IPD has no successful mixed strategies at all, while ICG favors mixed-strategies for an overwhelming majority of the games. Some details not shown in table 4.1 are discussed below.

For the cooperative set there is a single strategy winner after on average 167 generations. TfT wins 78% of the plays and is dominating in 91 out of 110 games[5]. AllD is dominating in the rest of the games and wins 20% of the plays.

For the defective-set there is a single strategy winning in 47 generations on average. TfT is dominating 84 games, AllD 21 games and 99.99D, playing D 99.99% of the time, 5 games out of 110 games in all. TfT wins 75% of the plays, AllD 20% and 99.99D 4%.

In the cooperative-set there are two formations of mixed strategies winning most of the games; one with two strategies and the other with three strategies involved. This means that when the play was finished after 10000 generations not a single play could separate these strategies finding a single winner. The two-strategy set ATfT and AllD wins 61% of the plays and the three-strategy set ATfT, AllD and AllCtot wins 33% of the plays. AllCtot means that one and just one of the strategies AllC, 99.99C, 99.9C, 99C or 90C is the winning strategy. For 3% of the games there was a single TfT winner within relatively few generations (on average 754 generations).

In the defective-set there is the same two formations winning most of the games. ATfT + AllDtot wins 69% of the plays and ATfT + AllC + AllDtot wins 24% of the plays. AllDtot means that one and just one of the strategies AllD, 99.99D, 99.9D, 99D or 90D is the winning strategy. TfT is a single winning strategy in 2% of the plays, which needs on average 573 generations before winning a play.

In the C-variant set all AllC variants are generous and TfT is even matched. AllD, ATfT and Random are all greedy strategies. In the D-variant set all AllD variants are greedy and TfT is still even-matched. AllC, ATfT and Random are now representing generous strategies.

In the IPD the even-matched TfT is a dominating strategy in both the C- and D-variant set with the greedy AllD as the only primary alternative. So the IPD will end up being a fully cooperative game (TfT) or a fully defecting game (AllD) after relatively few generations. This is the case both for the C-variant set and, within even fewer generations, for the D-variant set.

In ICG there is instead a mixed solution between two or three strategies. In the C-variant ATfT and AllD form a greedy two-strategy set[6]. In the three-strategy variant the generous AllCtot join the other two. In all, generous strategies only constitute about 10% of the mixed strategies. In the D-variant the generous ATfT forms various strategy sets with the greedy AllDtot.

5. DISCUSSION. In our investigation we found ICG to be a strong candidate for being the major cooperate game. ICG seems to facilitate cooperation as much as or even more than IPD, especially under noisy conditions. Axelrod regarded TfT to be a leading cooperative strategy, but in our investigation we found TfT

[5]A game is dominated by a certain strategy if it wins more than 50 out of 100 plays

[6]With just ATfT and AllD left ATfT will behave as a generous strategy even though it starts off as a greedy strategy in the C-variant environment

to have poor success under noisy conditions within ICG. These statements will be further addressed in the discussion below.

If it is true that more cooperating strategies are favored in ICG, we should also expect nice and forgiving strategies to be successful in this game. In the ICG, both players that play defect are faring the worst, which should favor generous strategies. Both ICG and coordination game favors nice, non-revenging, strategies, but unlike coordination game ICG may forgive a defection from the opponent. This makes ICG a primary candidate for being the main cooperative game, favoring both niceness and forgivingness.

Most studies today consider the IPD as a cooperative game where nice and forgiving strategies are successful. A typical winning strategy, like TfT, ends up as an agent playing cooperate all the time. There are contradictory arguments about cooperation within chicken games. The advantage of cooperation may be expected to be stronger, because the cost of defection is higher than in the prisoner's dilemma. Lipman [16] suggests that in ICG, mutual cooperation is less clearly the best outcome because there is no dominant strategy. Each agent prefers the equilibrium in which it defects and the other cooperates, but has no way to force the other agent to cooperate. A mixed strategy or a set of strategies, unlike a single dominant strategy, may favor mutual cooperation. With pure and mixed strategies we here refer to the set of strategies (played by individuals) winning the population tournament. A mixed strategy is a combination of two or more strategies from the given set of strategies i. e. an extended strategy set could include the former mixed strategy as a pure strategy.

In the normalized matrices stochastic memory-0 and memory-1 strategies are used. The main difference between IPD and ICG is best shown by the two strategies TfT and ATfT. TfT does the same as its opponent. This is a successful way of behaving if there is a pure-strategy solution because it forces the winning strategy to cooperate or defect, but not doing both. ATfT is doing very badly in IPD because it tries to jump between playing cooperate and defect.

In ICG we have a totally different assumption because a mixed-strategy solution is favored (at least in the present simulation). ATfT does the opposite as its opponent but cannot by itself form a mixed-strategy solution. It has to rely on other cooperative or defect strategies. In all different ICG ATfT is one of the remaining strategies, while TfT is only occasionally winning a play.

For a simple strategy setting like the cooperative and defective-set, ICG will not find a pure strategy winner at all but a mixture between two or more strategies, while IPD quickly finds a single winner.

Unlike the single play PD, which always favors defect, the IPD will favor playing cooperate. In CG the advantage of cooperation should be even stronger, because it costs more to defect compared to the PD, but in our simulation greedier strategies were favored with memory-0 and memory-1 strategies. We think this new paradox can be explained by a greater robustness of the chicken game. This robustness may be present if more strategies, like the strategies in the two other simulations, are allowed and/or noise is introduced. Robustness is expressed by two or more strategies winning the game instead of a single winner or by a more sophisticated single winner. Such a winner could be cTfT, Pavlov, or Fair in the presence of noise, instead of TfT. Also, with minor exceptions this is also true for noise between 0.02% and 20%.

An interesting exception to the higher success of cooperating strategies within ICG is the poor success under noisy conditions of TfT. The vulnerability of TfT to errors in the implementation of actions within the IPD is well known and has been discussed extensively ([3, 19, 4, 27, 7, 21, 22]). The even poorer ability of TfT to handle noise within the ICG, is however a novel finding. The classical description by Axelrod [3] of a successful strategy in a deterministic (non-noisy) environment is that it should be nice (not be the first to defect), provocable (immediately punish defection), forgiving (immediately reciprocate cooperation), and simple (easily recognizable). Obviously, under noisy conditions TfT either behaves less nice, provocable, forgiving, and simple, or these characteristics are of less value in the ICG. Axelrod and Dion [4] suggested that the difficulty for TfT to handle noise is an inherent consequence of generosity: vulnerability to exploitation. Errors in the implementation of strategies give rise to unconditional cooperation, which undercuts the effectiveness of simple and reciprocating strategies. It also introduces mutual defection among TfT players, reducing their obtained payoffs [22]. In the long run, the average payoffs of two interacting TfT players in a noisy environment converge to that of two interacting Random players [19]. Thus, the main problem for TfT in a noisy environment may be to cope with copies of itself.

A solution to the problem of noise for a strategy is to punish defection in the other player less readily than does TfT. This can be done either by not immediately responding to an opponent's defection or by avoidance of responding to the other player's defection after one has made an unintended defection [19]; see also [27]. Thus, some modified versions of TfT, Contrite tit-for-tat (CTfT) and generous tit-for-tat (GTfT) have proved

to cope much better with noise than the original TfT ([27, 9]). Bendor [6] concludes that uncertainty sometimes affects nice strategies negatively but he also proposes that reciprocating but untrustworthy strategies may start to cooperate because of unintended actions.

Several attempts have been made to classify strategies according to their willingness to play cooperate and defect, respectively, the classical being Axelrod's [1] distinction between nice and mean strategies based on whether a strategy's first draw is cooperate or defect, respectively. Under noisy conditions, the static description of a strategy based on its behavior under non-noisy becomes more or less meaningless. Naturally, a nice strategy then becomes meaner, and a mean strategy becomes nicer, but the actual behavior is difficult to evaluate.

6. CONCLUSION. In our opinion, the discussion about the evolution of cooperative behavior has relied too heavily on analyses within the prisoner's dilemma context. The differences in the outcome of IPD and ICG shown in our study suggest that future game theoretical analyses on cooperation should explore alternative payoff environments. The chicken game was discussed as a special case within the general hawk and dove context by Maynard Smith [18], but for some reason subsequent game theoretical studies has almost exclusively focused on the prisoner's dilemma. This is unfortunate, since the chicken game appears to us to be a very interesting game in explaining the evolution of cooperative behavior. If we give the involved agents the ability to establish trust the difference between the two kinds of games are easier to understand. In the PD establishing credibility between the agents means establishing trust, whereas in CG, it involves creating fear, i. e. avoiding situations where there is too much to lose [25]. This makes ICG a strong candidate for being a major cooperate game together with IPD. We therefore hope that in future studies, more attention will be paid to the role of chicken games in the evolution of agents with cooperative behavior within multi agent systems.

REFERENCES

[1] R. Axelrod, *Effective choice in the prisoner's dilemma*, Conflict Resolution, 24 (1980), pp. 3–25.
[2] R. Axelrod, *More effective choice in the prisoner's dilemma*, Journal of Conflict Resolution, 24 (1980), pp. 379–403.
[3] ———, *The Evolution of Cooperation*, Basic Books Inc., 1984.
[4] R. Axelrod and D. Dion, *The further evolution of cooperation*, Nature, 242 (1988), pp. 1385–1390.
[5] R. Axelrod and H. W.D., *The evolution of cooperation*, Science, 211 (1981).
[6] J. Bendor, *Uncertainty and the evolution of cooperation*, Journal of Conflict Resolution, 37 (1993), pp. 709–734.
[7] J. Bendor, R. Kramer, and S. S., *When in doubt...cooperation in a noisy prisoner's dilemma*, Journal of Conflict Resolution, 35 (1991), pp. 691–719.
[8] N. M. Boerlijst, M.C. and K. Sigmund, *Equal pay for all prisoners. / the logic of contrition*, tech. rep., IIASA Interim Report IR-97-73, 1996.
[9] R. Boyd, *Mistakes allow evolutionary stability in the repeated prisoner's dilemma game*, Journal of Theoretical Biology, 136 (1989), pp. 47–56.
[10] B. Carlsson, *Simulating how to cooperate in iterated chicken game and iterated prisoner's dilemma*, in Agent Engineering, J. Liu, N. Zhong, Y. Tang, and P. Wang, eds., vol. 43 of Machine Perception and Artificial Intelligence, World Scientific, 1998, pp. 285–292.
[11] B. Carlsson and S. Johansson, *An iterated hawk-and-dove game*, in Agents and Multi-Agent Systems, W. Wobcke, M. Pagnucco, and C. Zhang, eds., vol. 1441 of Lecture Notes in Artificial Intelligence, Springer Verlag, 1998, pp. 179–192.
[12] S. Johansson, B. Carlsson, and M. Boman, *Modelling strategies as generous and greedy in prisoner's dilemma like games*, in Simulated Evolution and Learning, B. McKay, X. Yao, C. Newton, J. Kim, and T. Furuhashi, eds., vol. 1585 of Lecture notes in artificial intelligence, Springer Verlag, 1998, pp. 285–292.
[13] J. Koeslag, *Sex, the prisoner's dilemma game, and the evolutionary inevitability of cooperation*, Journal of Theoretical Biology, 189 (1997), pp. 53–61.
[14] L. Lehmann and L. Keller, *The evolution of cooperation and altruism - a general framework and a classification of models*, Journal of Evolutionary Biology, 19 (2006), pp. 1365–1376.
[15] K. Lindgren, *Evolutionary dynamics in game-theory models*, in The Economy as an Evolving, Complex System II, W. Arthur, D. Lane, and S. Durlauf, eds., Addison–Wesley, 1997, pp. 337–367.
[16] B. Lipman, *Cooperation among egoists in prisoner's dilemma and chicken game*, Public Choice, 51 (1986), pp. 315–331.
[17] R. Luce and H. Raiffa, *Games and Decisions*, Dover Publications, 1957.
[18] J. Maynard Smith, *Evolution and the theory of games*, Cambridge University Press, 1982.
[19] P. Molander, *The optimal level of generosity in a selfish uncertain environment*, Journal of Conflict Resolution, 29 (1985), pp. 611–618.
[20] K. Nishimura and S. D.W., *Iterated prisoner's dilemma: Pay-off variance*, Journal of Theoretical Biology, 188 (1997), pp. 1–10.
[21] M. Nowak and K. Sigmund, *Tit for tat in heterogeneous populations*, Nature, 355 (1992), pp. 250–253.
[22] ———, *A strategy of win-stay, lose-shift that outperforms tit-for-tat in the prisoner's dilemma game*, Nature, 364 (1993), pp. 56–58.
[23] A. Rapoport and A. Chammah, *Prisoner's Dilemma A Study in Conflict and Cooperation*, The University of Michigan Press, 1965.

[24] R. Selten, *Reexamination of the perfectness concept for equilibrium points in extensive games*, International Journal of Game Theory, 4 (1975), pp. 25–55.
[25] G. Snyder, *prisoner's dilemma and chicken models in international politics*, International Studies Quartely, 15 (1971), pp. 66–103.
[26] F. Tutzauer, M. Chojnacki, and P. Hoffmann, *Network structure, strategy evolution and the game of chicken*, Social Networks, 28 (2006), pp. 377–396.
[27] J. Wu and R. Axelrod, *How to cope with noise in the iterated prisoner's dilemma*, Journal of Conflict Resolution, 39 (1995), pp. 183–189.

Edited by: Marcin Paprzycki, Niranjan Suri
Received: October 1, 2006
Accepted: December 10, 2006

Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 101–114. http://www.scpe.org
ISSN 1895-1767

A MULTI-AGENT INFRASTRUCTURE FOR ENHANCING ERP SYSTEM INTELLIGENCE

ANDREAS L. SYMEONIDIS[†‡*], KYRIAKOS C. CHATZIDIMITRIOU[†], DIONYSIOS KEHAGIAS[‡], AND PERICLES A. MITKAS[†‡]

Abstract. Enterprise Resource Planning systems efficiently administer all tasks concerning real-time planning and manufacturing, material procurement and inventory monitoring, customer and supplier management. Nevertheless, the incorporation of domain knowledge and the application of adaptive decision making into such systems require extreme customization with a cost that becomes unaffordable, especially in the case of SMEs. In this paper we present an alternative approach for incorporating adaptive business intelligence into the company's backbone. We have designed and developed a highly reconfigurable, adaptive, cost efficient multi-agent framework that acts as an add-on to ERP software, employing Data Mining and Soft Computing techniques in order to provide intelligent recommendations on customer, supplier and inventory management. In this paper, we present the architecture and development details of the developed framework, and demonstrate its application on a real test case.

Key words. ERP systems, Data Mining, Soft Computing, Multi-Agent Systems, Adaptive Decision Making

1. Introduction. *Enterprise Resource Planning* (ERP) systems are business management tools that automate and integrate all company facets, including real-time planning, manufacturing, sales, and marketing. These processes produce large amounts of enterprise data that are, in turn, used by managers and employees to handle all sorts of business tasks such as inventory control, order tracking, customer service, financing and human resources [16].

Despite the support current ERP systems provide on process coordination and data organization, most of them – especially legacy systems – lack advanced Decision-Support (DS) capabilities, resulting therefore in decreased company competitiveness. In addition, from a functionality perspective, most ERP systems are limited to mere transactional IT systems, capable of acquiring, processing, and communicating raw or unsophisticated processed data on the company's past and present supply chain operations [25]. In order to optimize business processes in the tactical supply chain management level, the need for analytical IT systems that will work in close cooperation with the already installed ERP systems has already been identified, and DS-enabled systems stand out as the most successful gateway towards the development of more efficient and more profitable solutions. Probing even further, Davenport [7] suggests that decision-making capabilities should act as an extension of the human ability to process knowledge and proposes the unification of knowledge management systems with the classical transaction-based systems, while Carlsson and Turban [3] claim that the integration of smart add-on modules to the already established ERP systems could make standard software more effective and productive for the end-users.

The benefits of incorporating such sophisticated DS-enabled systems inside the company's IT infrastructure are analyzed by Holsapple and Senna [14]. The most significant, among others, are:

1. Enhancement of the decision maker's ability to process knowledge.
2. Improvement of reliability of the decision support processes.
3. Provision of evidence in support of a decision.
4. Improvement or sustainability of organizational competitiveness.
5. Reduction of effort and time associated with decision-making, and
6. Augmentation of the decision makers' abilities to tackle large-scale, complex problems.

Within the context of Small and Medium sized Enterprises (SMEs) however, applying analytical and mathematical methods as the means for optimization of the supply chain management tasks is highly impractical, being both money– and time–consuming [5, 31]. This is why alternative technologies, such as Data Mining and Agent Technology have already been employed, in order to provide efficient DS-enabled solutions. The increased flexibility of multi-agent applications, which provide multiple loci of control [30] can lead to less development effort, while the cooperation primitives that Agent Technology adopts point to MAS as the best choice for addressing complex tasks in systems that require synergy of multiple entities. Moreover, DM has repeatedly been

*Corresponding author
[†]Electrical and Computer Engineering Dept., Aristotle University of Thessaloniki, GR541 24, Thessaloniki, GREECE
[‡]Intelligent Systems and Software Engineering Laboratory, Informatics and Telematics Institute - CERTH, GR570 01,Thessaloniki, GREECE, {asymeon,diok}@iti.gr, kyrxa@ee.auth.gr, mitkas@auth.gr

used for Market Trend Analysis, User Segmentation, and Forecasting. Knowledge derived from the application of DM techniques on existing ERP historical data can provide managers with useful information, which may enhance their decision-making capabilities.

Going briefly through related work, we see that DM and MAS have been used separately for efficient enterprise management and decision support. Rygielski et. al. [24] have exploited DM techniques for Customer Relationship Management (CRM), while Choy et. al. [4, 5] have used a hybrid machine learning methodology for performing Supplier Relationship Management (SRM). On the other hand, MAS integrated with ERP systems have been used for production planning [22], and for the identification and maintenance of oversights and malfunctions inside the ERP systems [15].

Elaborating on previous work, we have integrated AT and DM advantages into a versatile and adaptive multi-agent system that acts as an add-on to established ERP systems. Our approach employs Soft Computing, DM, Expert Systems, standard Supply Chain Management (SCM) and AT primitives, in order to provide intelligent recommendations on customer, supplier, and inventory issues. The system is designated to assist not only the managers of a company – "Managing by wire" approach [12] –, but also the lower-level, distributed decision makers – "Cowboys" approach [18]. Our framework utilizes the vast amount of corporate data stored inside ERP systems to produce knowledge, by applying data mining techniques on them. The extracted knowledge is diffused to all interested parties via the multi-agent architecture, while domain knowledge and business rules are incorporated into the system by the use of rule-based agents. It merges the, already proven capabilities of data mining with the advantages of multi-agent systems in terms of autonomy and flexibility, and therefore promises a great likelihood of success.

The rest of the paper is organized as follows. Section 2 presents the extensive Recommendation Framework in detail and describes the functional characteristics of the different types of agents that comprise it. Section 3 illustrates the basic functional operations of IPRA, an already developed add-on in a real enterprise environment. Finally, Section 4 summarizes the work presented, and concludes this paper.

2. The Intelligent Recommendation Framework. The arrival of a new customer order designates the initialization of the Intelligent Recommendation Framework (IRF) operation. All customer order preferences are, at first, gathered by the system operator via a front-end agent and are then transferred to the backbone (order) agents for processing. The order processing agents are of different types, each one related to a specific entity of the supply chain (company, customers, suppliers, products), and manage entity-specific data. In order to establish connectivity to the ERP system's database and access ERP data, another agent has also been implemented. By the use of DM techniques, all related entities' profiles are constructed for the recommendation procedure to be based on. When all processes are finalized, the front-end agent returns to the operator the intelligent recommendations produced by the framework, along with an explanatory memo. These recommendations are not designed to substitute the human operator, rather to aid him/her and the company to increase profit and efficiently manage customer orders and company supplies.

2.1. IRF Architecture. The general IRF architecture is illustrated in Figure 2.1. The IRF agents belong to one of six different agent types ($Q_1 - Q_6$) and are listed in Table 2.1. The main characteristics and the functionality of each type are discussed in the following paragraphs.

Table 2.1
The IRF agent types and their functionality

Agent type	Name	Functionality
Q_1	COA – Customer Order Agent	GUI agent
Q_2	RA – Recommendation Agent	Organization & Decision Making agent
Q_3	CPIA – Customer Profile Identification Agent	Knowledge Extraction agent
Q_4	SPIA – Supplier Profile Identification Agent	Knowledge Extraction agent
Q_5	IPIA – Inventory Profile Identification Agent	Knowledge Extraction agent
Q_6	ERPA – Enterprise Resource Planning Agent	Interface agent

2.1.1. Customer Order Agent type (COA). COA is an interface agent that may operate at the distribution points, or at the telephone center of an enterprise. COA enables the system operator to: a) transfer information into and out of the system, b) input order details into the system, and c) justify, by means of visualization tools, the proposed recommendations. When an order arrives into the system, COA provides the

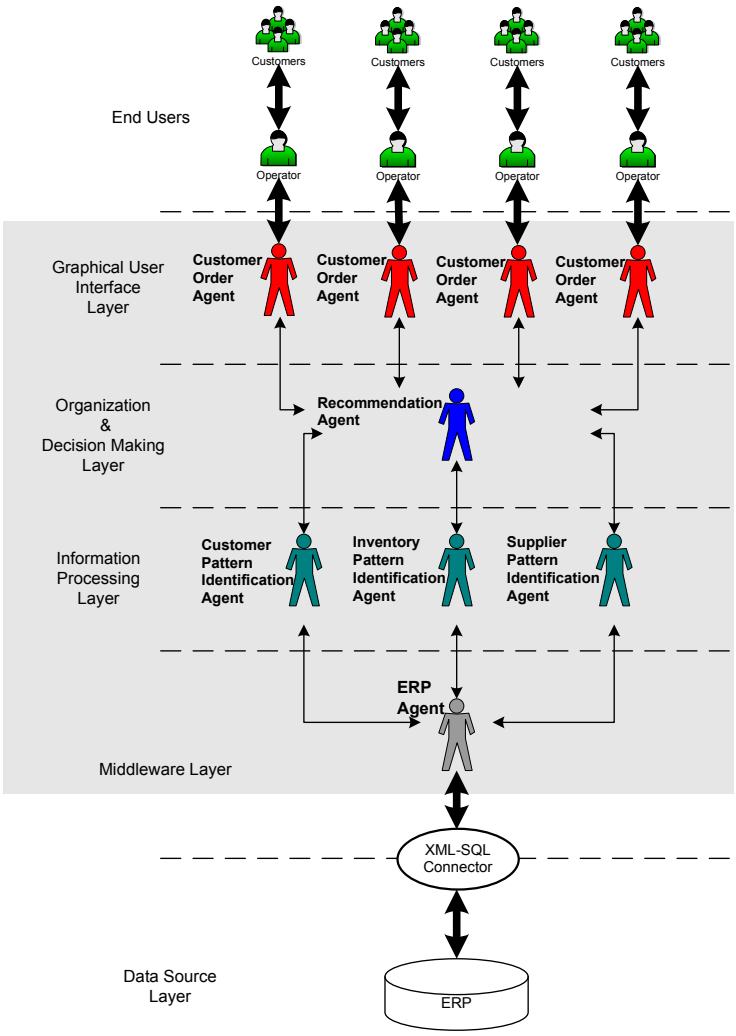


FIG. 2.1. *The IRF architectural diagram*

human agent with basic functionalities for inserting information on the customer, the order details (products and their corresponding quantities), payment terms (cash, check, credit etc.), backorder policies and, finally, the party (client or company) responsible for transportation costs. COA also encompasses a unit that displays information in various forms to explain and justify the recommendations issued by the RA.

2.1.2. Recommendation Agent type (RA). The RA is responsible for gathering the profiles of the entities involved in the current order and for issuing recommendations. By distributing the profile requests to the appropriate *Information Processing Layer* agents (CPIA, SPIA and IPIA - each one of them operating on its own control thread), and by exercising concurrency control, this agent diminishes the cycle-time of the recommendation process. RA is a rule-based agent implemented using the Java Expert System Shell (JESS) [9]. Static and dynamic business rules can be incorporated into the RA. The latter must be written into a document that the agent can read during its execution phase. In this way, business rules can be modified on-the-fly, without the need of recompiling, or even restarting the application.

2.1.3. Customer Profile Identification Agent Type (CPIA). CPIA is designed to identify customer profiles, utilizing the historical data maintained in the ERP system. The process can be described as follows: Initially, managers and application developers produce a model for generating the profiles of customers. They select the appropriate customer attributes that can be mapped from the data residing in the ERP database; these are the attributes that are considered instrumental for reasoning on customer value. Then, they decide

on the desired classification of customers, i.e., added-value to the company, discount due to past transactions etc. CPIA, by the use of clustering techniques, analyzes customer profiles periodically, and stores the outcome of this analysis into a profile repository for posterior retrieval. When a CPIA is asked to provide the profile of a customer, the current attributes of the specific customer are requested from the ERP database and are matched against those in the profile repository, resulting into the identification of the group the specific customer belongs to. During the development phase, one or more CPIA agents can be instantiated, and the distinction of CPIAs into training and recommendation ones, results to quicker response times when learning and inference procedures overlap.

2.1.4. Supplier Pattern Identification Agent Type (SPIA). SPIA is responsible for identifying supplier profiles according to the historical records found in the ERP database. In a similar to CPIA manner, managers identify the key attributes for determining a supplier's value to the company and their credibility. SPIA then generates supplier profiles and updates them periodically. For every requested item in the current order, the RA identifies one or more potential suppliers and requests their profiles from the SPIA. SPIA has to retrieve the current records of all the suppliers, identify for each one the best match in the profile repository, and return the corresponding profiles to the RA. Then RA can select the most appropriate supplier combination (according to its rule engine), and recommend it to the human operator. SPIA is also responsible for fetching to RA information about a specific supplier, such as statistical data on lead-times, quantities to be procured etc.

2.1.5. Inventory Profile Identification Agent Type (IPIA). IPIA is responsible for identifying product profiles. Product profiles comprise raw data from the ERP database (i.e., product price, related store, remaining quantities), unsophisticated processed data (for example statistical data on product demand) and intelligent recommendations on products (such as related products that the customer may be willing to purchase). Once more, managers and application developers have to identify the company priorities and map the profile to the data maintained by the ERP. Besides the directly–derived data, IPIA is responsible for identifying buying patterns. Market basket analysis can be performed with the help of association rule extraction techniques. Since this process is, in general, time-consuming, two or more IPIAs can be instantiated to separate the recommendation from the learning procedure.

2.1.6. Enterprise Resource Planning Agent Type (ERPA). ERPAs provide the middleware between the MAS application and the ERP system. These agents behave like transducers [11], because they are responsible for transforming data from heterogeneous applications into message formats that agents can comprehend. An ERPA handles all queries posted by CPIAs, IPIAs, and SPIAs by connecting to the ERP database and fetching all the requested data. It works in close cooperation with an XML connector which relays XML-SQL queries to the ERP and receives data in XML format. ERPA is the only IRF agent type that needs to be configured properly, in order to meet the connection requirements of different ERP systems.

2.1.7. Technologies adopted. IRF has been developed with the use of Agent Academy (AA) [20, 27] a platform for developing MAS architectures and for enhancing their functionality and intelligence through the use of DM techniques. All the agents are developed over the Java Agent Development Framework (JADE) ([2], which conforms to the FIPA specifications [28], while the required ontologies have been developed through the Agent Factory module (AF) of AA. Data mining has been performed on ERP data that are imported to AA in XML format, and are forwarded to the Data Miner (DM) of AA, a DM suite that expands the Waikato Environment for Knowledge Analysis (WEKA) tool [29].

The extracted knowledge structures are represented in PMML (Predictive Model Markup Language), a language that efficiently describes clustering, classification and association rule knowledge models [6]. The resulting knowledge has been incorporated into the agents by the use of the Agent Training Module (ATM) of AA. All necessary data files (ERP data, agent behavior data, knowledge structures, agent ontologies) are stored into AA's main database, the Agent Use Repository (AUR). Agents can be periodically recalled for retraining, since appropriate agent tracking tools have been incorporated into Agent Academy, in order to monitor agent activity after their deployment.

2.2. Installation and Runtime Workflows. Once a company chooses to add IRF to its already operating ERP system, a few important steps have to be performed. The installation procedure of the IRF is shown in Figure 2.2.

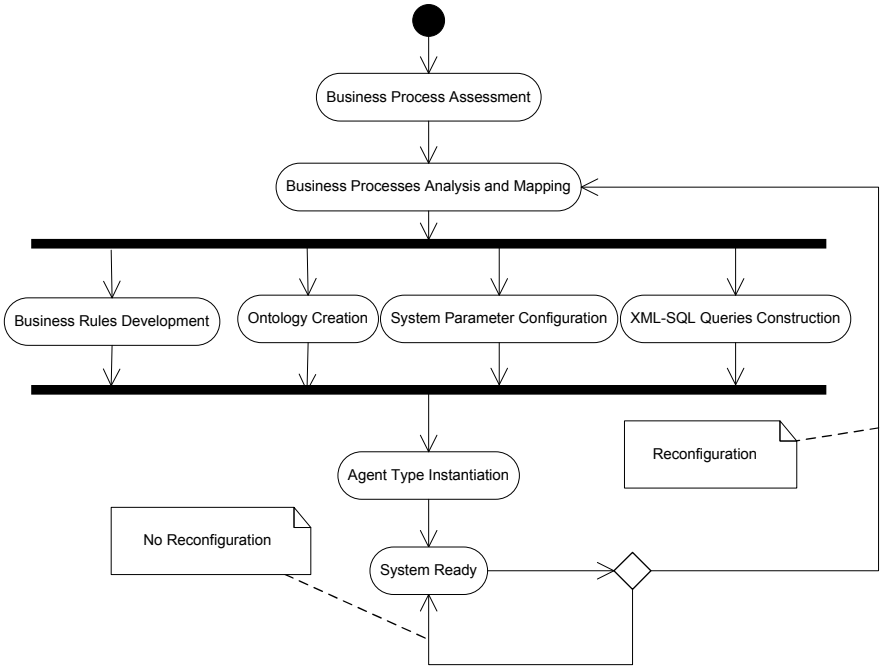


FIG. 2.2. *Installing IRF on top of an existing ERP*

At first, the company's business process expert, along with the IRF application developers have to make a detailed analysis and assessment of the current customer order, inventory and products procurement processes. The results are mapped to the recommendation process of the add-on and the relevant datasets are delineated in the ERP.

After modeling the recommendation procedure according to the needs of the company, parallel activities for producing required documents and templates for the configuration of the MAS application follow. Fixed business rules incorporating company policy are transformed to expert system rules, XML-SQL queries are built and stored in the XML documents repository, ontologies (in RDFS format) are developed for the messages exchanged and for the decision on the workflow of the agents, agent types instantiation requirements are defined (at different workstations and cardinalities) and other additional parameters are configured (i.e., simple retraining time-thresholds, parameters for the data-mining algorithms, such as support and confidence for market basket analysis etc).

Once bootstrapped, reconfiguration of the system parameters is quite easy, since all related parameters are documents that can be conveniently re-engineered. Figure 2.3 illustrates the workflow of the SPIA, where all the tasks described earlier in this section, can be detected. In case IRF needs to be modified due to a change in the company processes, the reconfiguration path must be traversed. The IPIA and CPIA workflows are similar and, thus, they are omitted.

2.3. System Intelligence.

2.3.1. Benchmarking customer and suppliers. In order to perform customer and supplier segregation, CPIA and SPIA use a hybrid approach that combines data mining and soft computing methodologies. Clustering techniques and fuzzy inferencing are adopted, in order to decide on customer and supplier "quality". Initially, the human experts select the attributes on which the profile extraction procedures will be based on. These attributes can either be socio-demographic, managerial or financial data, deterministic or probabilistic. We represent the deterministic attributes, which are directly extracted from the ERP database by ERPA, as Det_i, $i = 1, ...n$, where n is the cardinality of the selected deterministic attributes. On the other hand, we represent the average (AVG) and standard deviation values (STD) of probabilistic variables, which are calculated by ERPA, as AVG_j and STD_j, $j = 1..m$, where m is the cardinality of the selected probabilistic attributes P_j.

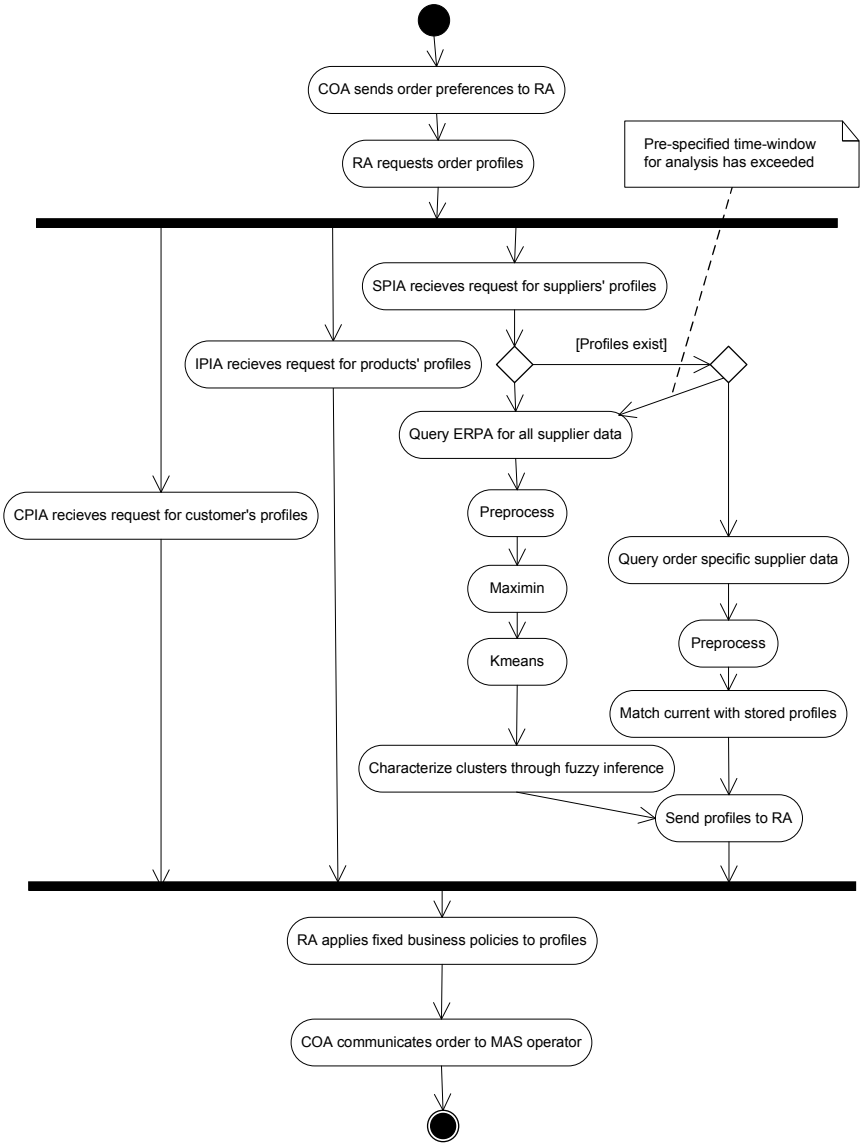


FIG. 2.3. *The Workflow of SPIA*

Each customer/supplier is thus represented by a tuple:

$$< Det_1, ..., Det_n, AVG_1, STD_1, ..., AVG_m, STD_m) > \tag{2.1}$$

where $i = 1..n$, $j = 1..m$, $i + j > 0$. Since real-world databases contain missing, unknown and erroneous data [13], ERPA preprocesses data prior to sending the corresponding datasets to the Information Processing Layer Agents. Typical preprocessing tasks are tuple omission and filling of missing values.

After the datasets have been preprocessed by ERPA, they are forwarded to CPIA and SPIA. Clustering is performed in order to separate customers/suppliers into distinct groups. The Maximin algorithm [17] is used to provide the number of the centers K that are formulated by the application of the K-means algorithm [19]. This way K disjoint customer/supplier clusters are created.

In order to decide on customer/supplier clusters' added-value, CPIA and SPIA employ an Adaptive Fuzzy Logic Inference Engine (AFLIE), which characterizes the already created clusters with respect to an outcome defined by company managers, i.e., supplier credibility. Domain knowledge is incorporated into AFLIE [8], providing to IRF the capability of characterization.

The attributes of the resulting clusters are the inputs to AFLIE and they may have positive ($\nearrow$) or negative ($\searrow$) preferred tendencies, depending on their beneficiary or harmful impact on company revenue. Once domain knowledge is introduced to AFLIE in the form of preferred tendencies and desired outputs, the attributes are fuzzified according to Table 2.2.

TABLE 2.2
Fuzzy variable definition and Interestingness of dataset attributes

Variable		**Fuzzy Tuple**
Input	**Preferred Tendency**	
Det_i	$\nearrow$	$\langle Det_i, [LOW, MEDIUM, HIGH], [Det_{i_1}, Det_{i_2}], Triangular\rangle$
Det_i	$\searrow$	$\langle Det_i, [LOW, MEDIUM, HIGH], [Det_{i_1}, Det_{i_2}], Triangular\rangle$
AVG_j	$\nearrow$	$\langle AVGj, [LOW, MEDIUM, HIGH], [AVG_{j_1}, AVG_{j_2}], Triangular\rangle$
AVG_j	$\searrow$	$\langle AVGj, [LOW, MEDIUM, HIGH], [AVG_{j_1}, AVG_{j_2}], Triangular\rangle$
Output	**Value Range**	
Y	Varies from Y_1 to Y_2 with a step of x	$\langle Y, [\#(Y_2 - Y_1)/x$ Incremental Fuzzy Values], $[Y_1, Y_2], Triangular\rangle$

The probabilistic variables are handled in an adaptive way and are used as inputs only when Chebyshev's inequality (Eq. 2.2) is satisfied [21]:

$$P\{|P_j - AVG_j|\epsilon\} \leq \frac{(STD_j)^2}{\epsilon^2}, \text{for any } \epsilon > 0 \tag{2.2}$$

Eq. 2.2 ensures the concentration of probabilistic variables near their mean value, in the interval $(AVG_j - \epsilon, AVG_j + \epsilon)$. No attributes with high distribution are taken as inputs to the final inference procedure, avoiding therefore decision polarization.

The formulation of the inputs (3 values: $[LOW, MEDIUM, HIGH]$) leads to 3^ν *Fuzzy Rules* (FR), where ν is the number of AFLIE inputs. FRs are of type:

If X_1 is $LX_1(k)$ and X_2 is $LX_2(k)$ and...and X_n is $LX_n(k)$
Then Y is $LY(l)$, $k = 1..3$, $l = 1..q$,

where q is the cardinality of the fuzzy values of the output. Triangular membership functions are adopted for all the inputs and outputs, whereas maximum defuzzification is used for crisping the FRs.

All inputs are assigned a *Corresponding Value* (CV), ranging from -1 to 1, according to their company benefit criterion (Table 2.2). The *Output Value* (OV) of Y is then calculated for each FR as:

$$OV = \sum_{i=1..n+m} w_i \cdot CV_i \tag{2.3}$$

where w_i is the weight of importance ($0 \leq w_i \leq 1$) of the i^{th} input attribute.

The OVs are mapped to Fuzzy Values (FV), according to the degree of discrimination of the output decision variables. By categorizing the range of the output into q fuzzy values, the $OV \longrightarrow FV$ mapping is based on the following formula:

$$FV(OV) = RND\left[OV \cdot [\frac{2(n+m)}{q}]\right] \tag{2.4}$$

where $RND(x)$ is the rounding function of x to the closest integer (i. e., $MEDIUM$ for $x = 3$, $MEDIUM\_HIGH$ for $x = 4$ etc).

After all clusters have been characterized, the corresponding OVs, along with the cluster centers, are stored inside a profile repository for posterior retrieval. This process signals the end of the training phase of CPIA and SPIA.

In real time, when a new order comes into the system, RA requests the corresponding customer profile and the profiles of the suppliers that are related to the ordered products. CPIA and SPIA request, in turn, the attributes of these entities from ERPA, and match them against the profiles stored inside the profile repository, by the use of the Assigned Cluster (AC) criterion. AC is a closeness-to-cluster-center function, given by the following equation:

$$AC = \min_{i=1..k} \{ \sqrt{\sum_{i=1}^{n+m} (c_i - xc_{ji})} \} \tag{2.5}$$

where k is the number of clusters, n the number of attributes, c_i is the i^{th} attribute value of the cluster center vector $c = (c_1, c_2, ..., c_n)$, and xc_{ij} the i^{th} attribute value of the j^{th} current vector $xc_j = (xc_{j1}, xc_{j2}, ..., xc_{jn})$. The winning cluster along with its OV is returned to RA.

2.3.2. IPIA products profile. The IPIA plays a dual role in the system:

1. It fetches information on price, stock, statistical data about demand faced by the ordered products, and
2. It provides recommendations on additional items to buy, based on association rule extraction techniques.

In order to provide adaptive recommendations on ordering habits, IPIA incorporates knowledge extracted by the Apriori algorithm ([1, 10]. The association rules extracted are stored inside the profile repository for later retrieval.

Special attention should be drawn to the fact that the transactions included into the dataset to be mined may span several different customer order periods. XML-SQL queries can be adapted to perform data mining either to the whole dataset or the datasets of specific periods. Thus, IPIA is highly adaptable, both for companies in the general merchandize domain, but also for companies that sell seasonal goods (for example toys). The recommendations of IPIA, as well as the information concerning stock availability and price, are sent to the RA.

2.3.3. RA Intelligence. As mentioned earlier, RA is an expert agent that incorporates fixed business policies applied to customers, inventories, and suppliers. These rules are related, not only to raw data retrieved from the ERP database and order preferences provided by customers, but also to the extracted knowledge provided by the Information Processing agents. There are three distinct rule types that RA can realize:

1. Simple $\langle If \ldots Then \ldots \rangle$ statements,
2. Rules describing mathematical formulas, and
3. Rules providing solutions to search problems and constraint satisfaction problems.

An example is provided below for each one of these rule types:

Example 1: Simple Rules

Additional discounts or burdens to the total price of an order can be implemented by the use of simple rules (knowledge extracted is denoted in bold):

1. IF ($TotalOrderRevenue >= 100$) AND (**CustomerValue** $= LOW$)
   THEN $TotalDiscount+ = 5\%$;
2. IF (**CustomerValue** $= LOW$) THEN $TotalDiscount- = 5\%$;
3. IF ($ProductType = ChristmasProducts$) AND ($TotalQuantity >= 100$)
   THEN **ProductDiscount**$+ = 10\%$;
4. IF (**RecommendedProductsPurchased** $= True$)
   THEN $ProductDiscount+ = 5\%$;

Example 2: Mathematical Formulas

(a) Re-order/Order-up-to-level metric sS

The re-order/order-up-to-level-point metric (sS) provides efficient inventory management for either no-fixed cost orders or fixed cost orders [16]. In the case of no-fixed cost orders (where $s = S$), the reorder point is calculated as:

$$sS = AVGD \cdot AVGL + z \cdot \sqrt{AVGL \cdot STDD^2 + AVGD^2 \cdot STDL^2} \tag{2.6}$$

where z is a constant chosen from statistical tables to ensure the satisfaction of a pre-specified value for the company's service level. Table 2.3 illustrates the value of z in correlation with the desired service level. In most legacy ERP systems such attributes have to be provided by users and cannot be derived automatically.

TABLE 2.3
Service Level and corresponding z Value

Service Level	90%	91%	92%	93%	94%	95%	96%	97%	98%	99%	99.9%
z	1.29	1.34	1.41	1.48	1.56	1.65	1.75	1.88	2.05	2.33	3.08

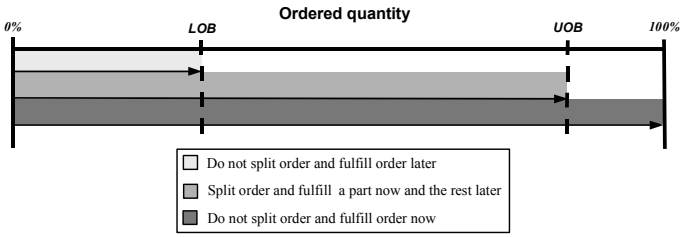


FIG. 2.4. *RA order splitting policy*

(b) Splitting Policy

A splitting policy is applied when company stock availability cannot satisfy order needs. Upon arrival of a new order, the quantity of ordered items and available stock are cross-checked. If the requested quantities are available, the order is fulfilled immediately. Otherwise, the final supplying policy that the RA recommends is set according to the schema illustrated in Figure 2.4.

The LOB and UOB thresholds depend on the estimated customer value. In case we choose to incorporate product discount and customer priority into our splitting policy (for example, customers that enjoy better discount and have a higher priority to have a lower LOB and an higher UOB), we may adjust LOB and UOB according to the following equations:

$$LOB = \alpha_l \cdot \exp[-(b_{pl}\hat{p} + b_{dl}\hat{d})] \tag{2.7}$$

$$UOB = \alpha_u \cdot \exp(b_{pu}\hat{p} + b_{du}\hat{d}) \tag{2.8}$$

where $\hat{p}$ is the priority normalized factor, $\hat{d}$ is the discount normalized factor, while the weighting factors $\langle \alpha_l, b_{pl}, b_{dl}, \alpha_u, b_{pu}, b_{du} \rangle$ are estimated in order to satisfy minimal requirements on LOB and UOB range.

If available stock is below $LOB\%$ of the ordered quantity, the entire order is put on hold until the company is supplied with adequate quantities of the ordered item. When item availability falls within the $[LOB-UOB]\%$ range of the ordered quantity, the order is split. All available stock is immediately delivered to the customer, whereas the rest is ordered from the appropriate suppliers. Finally, in case the available stock exceeds $UOB\%$ of the ordered quantity, the order is immediately preprocessed and the remaining order percentage is ignored.

Example 3: Problem Searching

(a) Problems that require heuristics application and/or constraint satisfaction

Based on raw data from the ERP and on knowledge provided by SPIA, Recommendation Agents can yield solutions to problems like the selection of the most appropriate supplier with respect to their added-value,

proximity to the depleted company store, or the identification and application of an established contract.

(b) Enhanced Customer Relationship Management

Using the knowledge obtained by customer clustering, RA can implement a variety of targeted discount strategies in the form of crisp rules. Thus, the company has additional flexibility in its efforts to retain valuable customers and entice new ones with attractive offers [23].

TABLE 2.4
IPRA inputs and outputs

CPIA		SPIA		IPIA	RA
Input	**Preferred Tendency**	**Input**	**Preferred Tendency**	**Input**	**Input**
Account balance	↘	Account balance	↘	Stock Availability	Ordered Quantity
Credit Limit	↗	Credit Limit	↗	Item price	Stock Availability
Turnover	↗	Turnover	↗	Supplier ids	Re-order metric
Average Order Periodicity	↘	Average Order Completion	↘	Average Item Turnover (AIT) for the last two years	Supplier Geographic Location
Standard deviation of Order	-	Standard deviation of Order	-	Monthly Standard Deviation of	Lower Order Break-point
Average Order Income	↗	Average Payment Terms	↘		Upper Order Break-point
Standard deviation of Order Income	-	Standard deviation of Payment Terms	-		Customer Geographic Location
Average Payment Terms	↘	Supplier Geographic Location	↘		
Standard deviation of Payment Terms	-				
Customer Geographic Location	↘				

IPRA Outputs					
Output	**Value Range**	**Output**	**Value Range**	**Output**	**Output**
DISCOUNT	Varies from 0 – 30%, using a step of 5%	CREDIBILITY	Ranging from 0 – 1, using a step based on the number of supplier clusters	PROPOSED ORDER ITEMS	SPLITTING POLICY
PRIORITY	Varies from 0 – 3, using a step of 1				ADDITIONAL DISCOUNT
					CUSTOMER STATISTICS

3. An IRF Demonstrator. In order to demonstrate the efficiency of IRF, we have developed IPRA [26], an Intelligent Recommender module that employs the methodology presented in Chapter 5. The system was integrated into the IT environment of a large retailer in the Greek market, hosting an ERP system with a sufficiently large data repository. IPRA was slightly customized to facilitate access to the existing Oracle™ database.

Our system proved itself capable of managing over 25.000 transaction records, resulting in the extraction of truly "smart" suggestions. The CPIA and the SPIA performed clustering of over 8.000 customers (D_{IQ_3} dataset) and 500 suppliers (D_{IQ_4} dataset), respectively, while IPIA performed association rule extraction on

14125 customer transactions (D_{IQ_5} dataset).

All the attributes used by the Information Processing agents as inputs for DM, their corresponding preferred tendency, the inputs of the RA JESS engine, as well as the outputs of the IPRA system and their value range, are listed in Table 2.4.

The Information Processing agents of IPRA, in order to provide RA with valid customer and supplier clusters, as well as interesting additional order items, performed DM on the relevant datasets. For the specific company, CPIA and SPIA have identified each five major clusters representing an equal number of customer and supplier groups, respectively. Resulting customer (supplier) clusters, as well as the discount and priority (credibility), calculated by the CPIA (SPIA) Fuzzy Inference Engine for each cluster, are illustrated in Table 3.1 and Table 3.2.

TABLE 3.1
The resulting customer clusters and the corresponding Discount and Priority values

Center ID	Population (%)	Discount (%)	Priority
0	0.002	20	High
1	10.150	10	Medium
2	46.600	15	Medium
3	22.240	10	Medium
4	20.830	5	Low

TABLE 3.2
The resulting supplier clusters and the corresponding Supplier Value towards the company

Center ID	Population (%)	Value
0	15.203	Low
1	10.112	Medium
2	25.646	Low
3	34.521	Medium
4	13.518	High

TABLE 3.3
The generated association rules with the predefined support and confidence thresholds.

Generated Rules	Support	Confidence
25	2%	90%
10	4%	90%

IPIA, on the other hand, has extracted a number of association rules from the records of previous orders, as shown in Table 3.3.

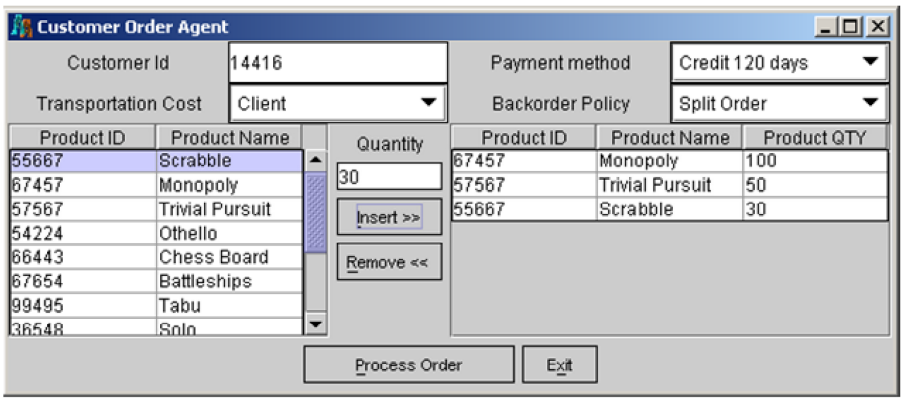


FIG. 3.1. *GUI of Customer Order Agent with information on the new order*

As already mentioned, upon receiving an order, the human agent collects all the necessary information, in order to provide IPRA with input. Data collected are handled by COA, the GUI agent of the system. An instance of the GUI is illustrated in Figure 3.1.

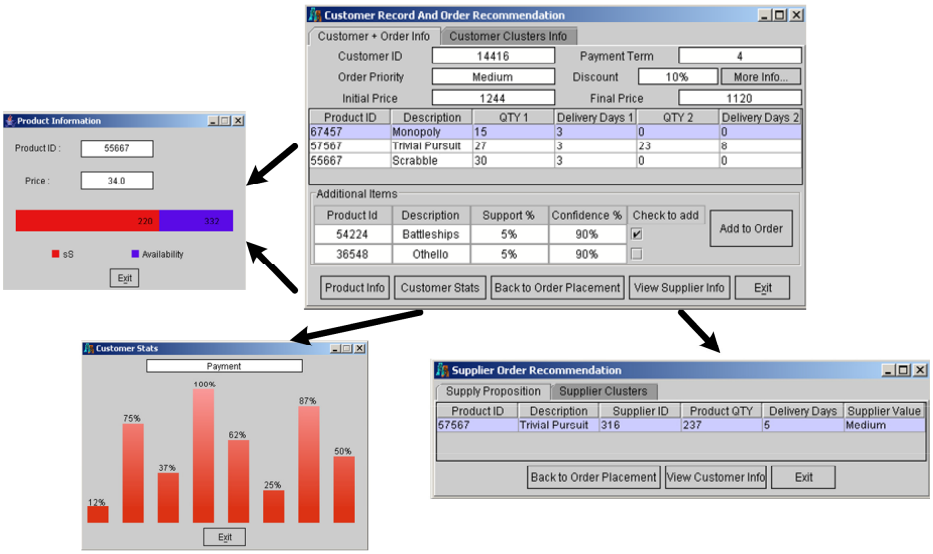


Fig. 3.2. *The final IPRA Recommendation*

All information on items and quantities to be ordered, backorder policy, payment method, and transportation costs are given as input to IPRA. When the order process is initialized, COA forwards to the CPIA, SPIA, IPIA and RA respectively the already collected information. CPIA checks on the cluster the client falls into, SPIA decides on the best supplier, (according to his/her added-value), in case an order has to be placed to satisfy customer demand, IPIA proposes additional items for the customer to order, and all these decisions are passed on to the RA, which decides on the splitting policy, (if needed) and on additional discount.

Figure 3.2 illustrates the final recommendation created. Detailed information on the order and its products, customer suggested priority and discount, customer clusters, supplier suggested value and supplier clusters, additional order items, suggested order policy and statistics, are at the disposal of the human agent, to evaluate and realize the transaction at the maximal benefit of the company.

4. Conclusions. An ERP system, although indispensable, constitutes a costly investment and the process of updating business rules or adding customization modules to it is often unaffordable, especially for SMEs. The IRF methodology aspires to overcome the already mentioned deficiencies of non DS-enabled ERP systems, in a low-cost yet efficient manner. Knowledge residing in a company's ERP can be identified and dynamically incorporated into versatile and adaptable CRM/SRM solutions. IRF integrates a number of enhancements into a convenient package and establishes an expedient vehicle for providing intelligent recommendations to incoming customer orders and requests for quotes. Recommendations are independently and perpetually adapted, without an adverse impact on IRF run-time performance. IRF architecture ensures reusability and re-configurability, with respect to the underlying ERP. Table 3.4 summarizes the key enhancements provided by the augmentation of ERP systems with the IRF module.

REFERENCES

[1] A. Amir, R. Feldman, and R. Kashi, *A new and versatile method for association generation*, Information Systems, 22 (1999), pp. 333–347.
[2] F. Bellifemine, A. Poggi, and R. Rimassa, *Developing multi-agent systems with JADE*, Lecture Notes in Computer Science, 1986 (2001), pp. 89–101.
[3] C. Carlsson and E. Turban, *Dss: directions for the next decade*, Decision Support Systems, 33 (2002), pp. 105–110.
[4] K. L. Choy, B. Lee, and V. Lo, *Design of an intelligent supplier relationship management system: a hybrid case based neural network approach*, Expert Systems with Applications, 24 (2003), pp. 225–237.

TABLE 3.4
IRF enhancements to ERPs

	IRF + ERP	Legacy ERPs
Static Business Rules	**Yes** Provided as rule documents changed on the fly.	**Yes** Hard-coded by the ERP vendor.
Dynamic Business Rules	Applied to data + knowledge	Applied only to data
Market Basket Analysis	**Yes** Added online to the recommendation procedure	**No** (Unless external MBA is performed)
Recommendation Procedure	Automatically generated	Through reports
Inventory Management	Thresholds automatically adapted	Thresholds inserted manually if applicable (Unless SCM module incorporated)
Decision cycle-time	**Short** (Not related to database size)	**Long** (Related to database size)
Distributed Decision Making	**Yes** Recommendations can be used by lower level personnel	**No**
Adaptability	**High**	**Low**
Autonomy	**Yes**	**No**
Customers Intelligent Evaluation	**Yes**	**No** (Unless CRM module incorporated)
Suppliers Intelligent Evaluation	**Yes**	**No** (Unless SRM module incorporated)
Information Overload Reduction	**High**	**Small** (Through reports)
Cost of enhancement	**Low** (Use of AA platform)	**High** (Customization/third party DS COTS)

[5] K. L. CHOY, W. B. LEE, AND V. LO, *Development of a case based intelligent customer-supplier relationship management system*, Expert Systems with Applications, 23 (2002), pp. 281–297.

[6] DATA MINING GROUP, *Predictive model markup language specifications (pmml), ver. 2.0.*, tech. report, The DMG Consortium, 2001.

[7] T. H. DAVENPORT, *The future of enterprise system-enabled organizations*, Information Systems Frontiers, 2 (2000), pp. 163–180.

[8] A. A. FREITAS, *On rule interestingness measures*, Knowledge-Based Systems, 12 (1999), pp. 309–315.

[9] E. J. FRIEDMAN-HILL, *Jess, The Java Expert System Shell*, Sandia National Laboratories, Livermore, CA, USA, 1998.

[10] V. GANTI, J. GEHRKE, AND R. RAMAKRISHNAN, *Mining very large databases*, Computer, 32 (1999), pp. 38–45.

[11] M. R. GENESERETH AND S. KETCHPEL, *Software agents*, Communications of the ACM, 37 (1994), pp. 48–53.

[12] S. H. HAECKEL AND R. NOLAN, *Managing by wire*, Harvard Business Review, October 1994, pp. 122–132.

[13] J. HAN AND M. KAMBER, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, 2001.

[14] C. W. HOLSAPPLE AND M. P. SENA, *Erp plans and decision-support benefits*, Decision Support Systems, (2004).

[15] O. B. KWON AND J. J. LEE, *A multi agent intelligent system for efficient erp maintenance*, Expert Systems with Applications, 21 (2001), pp. 191–202.

[16] S. D. LEVI, P. KAMINSKY, AND S. E. LEVI, *Designing and managing the supply chain*, McGraw-Hill, 2000.

[17] C. G. LOONEY, *Pattern Recognition Using Neural Networks: Theory and Algorithms for Engineers and Scientists*, Oxford University Press, 1997.

[18] T. W. MALONE, *Inventing the organizations of the twentieth first century: control, empowerment and information technology*, in Harvard Business Review Sept/Oct 1998, Harvard Business School Press, 1998, pp. 263–284.

[19] J. B. MCQUEEN, *Some methods of classification and analysis of multivariate observations*, in Proceedings of Fifth Berkeley Symposium on Mathematical Statistics and Probability, L. M. L. Cam and J. Neyman, eds., 1967, pp. 281–297.

[20] P. A. MITKAS, D. KEHAGIAS, A. L. SYMEONIDIS, AND I. ATHANASIADIS, *A framework for constructing multi-agent applications and training intelligent agents*, in Proceedings of the 4th International Workshop on Agent-Oriented Software Engineering, Springer-Verlag, 2003, pp. 1–16.

[21] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, EDITION:2nd; McGraw-Hill Book Company; New York, NY, 1984.
[22] Y. Peng, T. Finin, Y. Labrou, B. Chu, W. Tolone, and A. Boughannam, *A multi agent system for enterprise integration*, Applied Artificial Intelligence, 13 (1999), pp. 39–63.
[23] R. T. Rust, V. A. Zeithaml, and K. Lemon, *Driving customer Equity: How customer lifetime value is reshaping corporate strategy*, The Free Press, 2000.
[24] C. Rygielsky, J. C. Wang, and D. C. Yen, *Data mining techniques for customer relationship management*, Technology in Society, 24 (2002), pp. 483–502.
[25] J. Shapiro, *Bottom-up vs. top-down approaches to supply chain modeling*, Kluwer, 1999, pp. 737–759.
[26] A. L. Symeonidis, D. Kehagias, and P. A. Mitkas, *Intelligent policy recommendations on enterprise resource planning by the use of agent technology and data mining techniques*, Expert Systems with Applications, 25 (2003), pp. 589–602.
[27] A. L. Symeonidis, P. A. Mitkas, and D. Kehagias, *Mining patterns and rules for improving agent intelligence through an integrated multi-agent platform*, in Proceedings of the 6th IASTED International Conference on Artificial Intelligence and Soft Computing, 2002.
[28] The FIPA Foundations, *Fipa-sl specifications, 2000, fipa sl content language specification*, tech. report, The FIPA Consortium, March 2000.
[29] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufman, 2000.
[30] M. Wooldridge, *Intelligent agents*, in Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, G. Weiss, ed., The MIT Press, Cambridge, MA, USA, 1999, ch. 1, pp. 27–78.
[31] J. H. Worley, G. R. Castillo, L. Geneste, and B. Grabot, *Adding decision support to workflow systems by reusable standard software components*, Computers in Industry, 49 (2002), pp. 123–140.

Edited by: Marcin Paprzycki, Niranjan Suri
Received: October 1, 2006
Accepted: December 10, 2006

Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 115–130. http://www.scpe.org
ISSN 1895-1767

DATA MANAGEMENT IN DISTRIBUTED SYSTEMS: A SCALABILITY TAXONOMY

A VIJAY SRINIVAS AND D JANAKIRAM*

Abstract.
Data management is a key aspect of any distributed system. This paper surveys data management techniques in various distributed systems, starting from Distributed Shared Memory (DSM) systems to Peer-to-Peer (P2P) systems. The central focus is on scalability, an important non-functional property of distributed systems. A scalability taxonomy of data management techniques is presented. Detailed discussion of the evolution of data management techniques in the different categories as well as the state of the art is provided. As a result, several open issues are inferred including use of P2P techniques in data grids and distributed mobile systems and the use of optimal data placement heuristics from Content Distribution Networks (CDNs) for P2P grids.

1. Introduction. Data management is an important facet of distributed systems. Data management encompasses the ability to describe data, handle multiple copies (replication or caching) of data objects or files, support for meta-data as well as data querying and accessing. Different approaches for data management have given importance to these different aspects and provide explicit support, while other aspects are implicitly or indirectly supported. For instance, Distributed Shared Memory (DSM) systems and shared object spaces handled consistency of replicated data, but supported meta-data indirectly through object lookups.

Orthogonal to the above mentioned issues of managing data, the main non-functional challenges are fault-tolerance, scalability and security, as illustrated in [32]. We survey various distributed systems from the perspective of scalability of data management solutions and provide a scalability taxonomy. We classify data management approaches into three categories: Centralized/Naively Distributed (CND) techniques, Sophisticated/Intermediate Data (SID) management techniques and Large Scale Data (LSD) management techniques. We give a brief view of the evolution of data management in each of the categories.

CND techniques for data management were used by DSM systems such as TreadMarks [10], Munin [25] and shared object spaces such as Linda [24], Orca [36] and T Spaces [4]. Many of these systems provide application transparent replica consistency management. They use centralized or naively distributed components to achieve the same. For instance, T Spaces uses a centralized server for consistency maintenance and for object lookups, while Java Spaces [81] uses a centralized transaction coordinator.

SID techniques have been used mainly in data management in grid computing systems such as [51], which provides a Replica Management Service (RMS). Some of these systems are characterized by data sharing across autonomous organizations at intermediate scale (possibly thousands of nodes). These approaches mainly manage replicated data in a grid computing environment. Data grids [27] handle data management as first class entities in addition to computation issues. They are characterized by the size of the data sets, which could be order of gigabytes or even terabytes. High Energy Physics (HEP) applications such as GriPhyN [31] and CERN [79] are examples of data grids. Other approaches that use SID techniques include Content Distribution Networks (CDNs) and data management in distributed mobile systems. CDNs such as Akamai [43] have been proposed to deliver web content to users from closer to the edge of the Internet, enabling web servers to scale up. Data management in distributed mobile systems are characterized by data sharing in the presence of mobile nodes, exemplified by systems such as Coda [74]. The common feature across these different systems is the scale of operation (thousands of nodes) that distinguishes SID techniques for data management. Many of these systems assume that failures are rare and reliable servers (distributed, not centralized) are available.

LSD management techniques do not assume reliable servers. The distinguishing feature of LSD techniques is that the execution of services is delegated to the edges of the Internet, resulting in high scalability and fault-tolerance. LSD techniques work well over the Internet and could handle millions of nodes/data entities. Peer-to-Peer file sharing systems such as Napster [57] and Gnutella [33], P2P file storage management systems such as PAST [15] and Oceanstore [49] as well as P2P extensions to Distributed DataBase Management Systems (DDBMS) such as PIER [38] and PeerDB [60] all fall into the LSD category.

A taxonomy of data grids has been provided in [87]. It compares data grids with related data management approaches such as CDNs, DDBMS and P2P systems. A functional perspective of data management that focuses on data location, integration, sharing and query processing as well as the different P2P systems that

*Distributed & Object Systems Lab, Dept. of Computer Science & Engg., Indian Institute of Technology, Madras, India, http://dos.iitm.ac.in,{avs, djram@cs.iitm.ernet.in}

address these functionalities is given in [50]. A survey of P2P content distribution has been provided in [77]. It examines P2P architectures from the perspective of non-functional properties such as performance, security, fairness, fault-tolerance and scalability. Our survey is broader and tries to provide the equivalent survey for grids, P2P systems, CDNs and DDBMS. We also provide a scalability taxonomy that distinguishes our survey from others. Further, we discuss state of the art in several of these areas and discuss how ideas/concepts/techniques from one area can be applied to others. The reader must keep in mind that though the authors have made an effort to be unbiased, the survey has limitations as it is perceived through their looking glass.

The rest of the paper is organized as follows. Section 2 discusses the CND techniques for data management and includes DSMs and shared object spaces. Section 3 discusses the SID techniques and includes data management in grids, CDNs, and distributed mobile systems. Section 4 discusses P2P data management techniques. Section 5 explores the state of the art data management techniques in distributed systems. Section 6 concludes the paper and includes a taxonomy figure and gives directions for future research.

2. CND Techniques: Data Replication in DSMs and Shared Object Spaces. DSM provides an illusion of globally shared memory, in which processors can share data, without the application developer needing to specify explicitly where data is stored and how it should be accessed. DSM abstraction is particularly useful for parallel computing applications, as demonstrated by TreadMarks [10]. Collaborative applications such as on-line chatting and collaborative browsing would be easier to develop over a DSM.

Page based DSMs can be more efficient, due to the availability of hardware support for detecting memory accesses. But due to the larger granularity of sharing, page based DSMs may suffer from false sharing. Relaxed consistency models including Release Consistency (RC) and its variants such as lazy RC allow false sharing to be hidden more efficiently than strict consistency models [64]. Munin [25] was an early DSM system which focused on reducing the communication required for consistency maintenance. It provides software implementation of RC. TreadMarks [10] is another DSM system that provides an implementation of release consistency. Java/DSM [91] provides a Java Virtual Machine (JVM) abstraction over TreadMarks. It is an example of page based DSMs, similar to Munin and TreadMarks.

Release consistency is a widely known relaxed consistency model for DSMs. Memory accesses are divided into synchronization (sync) and non-synchronization (nsync) operations. The nsync operations are either data operations or special operations not used for synchronization. The sync operations are further divided into acquire and release operations. An acquire is like a read operation to gain access to a shared location. A release is the complementary operation performed to allow access to the shared location. Acquire and release operations can be thought of as conventional operations on locks. There are two variations of RC, RC_{sc}—which realizes sequential consistency and RC_{pc}—which realizes processor consistency. RC_{sc} maintains program order from an acquire to any operation that follows it, from an operation to a release and between special operations. RC_{pc} is similar, except that write to read program order is not maintained for special operations. Eager RC, as the original RC became subsequently known [48], requires ordinary shared memory access to be performed only when a subsequent release operation is due by the same processor. Lazy RC (LRC) is a variation of RC in which processors further delay performing modifications until subsequent acquires by other processors and modifications are made only by the *acquiring* processor. LRC intuitively assumes competing shared accesses to be separated by synchronization operations.

2.1. Shared Object Spaces. Object based DSMs (also known as shared object spaces) alleviate the false sharing problem by letting applications specify granularity of sharing. Examples of object based DSMs include Linda [24], Orca [36], T Spaces [4], JavaSpaces [81] as well as an object based DSM in the .NET environment [75]. Orca relies on an update mechanism based on totally ordered group communication to serialize access to replicas. Even though a study has shown that the overhead of totally ordered group communication affects application performance minimally [37][1], the study was done on a Myrinet cluster. Orca has not been evaluated on the Internet scale. T spaces is a shared object space from IBM [4] that adds database functionality to Linda tuplespace [24] and is implemented in Java to take advantage of its wider usability. In addition to the traditional Linda primitives of *in, out, read*, T spaces supports set oriented operators and a novel rendezvous operator called *rhonda*. Global shared objects [90] allows heap objects in a JVM to be shared across nodes. Based on memory access patterns of applications, it also proposes various consistency mechanisms to be realized efficiently. However, it uses locks and per-object lock managers for keeping replicas consistent. It does not address failures of the lock manager. Java Spaces specification from Sun [81] provides a distributed persistent

[1] This is due to its choice of which objects to replicate—those with high read/write ratios and efficient implementation of totally ordered group communication.

shared object space using Java RMI and Java serialization. It provides Linda-like operations on the tuple space and uses Jini's transaction specification to achieve serializability of write operations. It also does not address fault tolerance, an important issue for Internet scale systems.

2.1.1. Globe. Globe [3] attempted to address the challenges of building software infrastructure for developing applications over the Internet. A key design objective of Globe was to provide a uniform model for distributed computing. This means that Globe provides a uniform way to access common services (such as naming, replication and communication) without sacrificing distribution transparency. Objects in Globe encapsulate policies for replication, migration, etc. Each object comprises multiple sub-objects, allowing an object to be physically distributed. The different sub-objects of an object include one each for semantics (functionality), communication (sending/receiving messages), replication and control flow. This helps the programmer to separate functionality from orthogonal non-functional properties such as replication. Objects also help in realizing distribution transparency by hiding implementation details behind well defined interfaces. The implementation framework of Globe is flexible, meaning that different implementations of the same interfaces are possible. It also provides an efficient mechanism for object lookups by using a tree based hierarchical naming space. It must be observed that distributed object middleware such as CORBA [61] also provide similar services such as naming and trading. But they cannot provide object-specific policies that can be provided in Globe.

2.2. Software Availability and Usage Summary. To the knowledge of the authors, T spaces and Java Spaces are widely used and are available as open source software. Linda is a specification and has been implemented by several groups. Orca and Globe are research prototypes, information on their deployment and use is not available.

2.3. Observations. We have proposed a generic scalability model for analyzing distributed systems in [6]. It takes the view that scalability of distributed systems should be analyzed considering related issues such as consistency, synchronization, and availability. We give below the essence of the model.

$scalability = f(avail, sync, consis, workload, faultload)$

- *avail* is availability—can be quantified as the ratio of the number of transactions accepted versus those submitted.
- *consis* is consistency, itself a function of update ordering and consistency granularity. Update ordering refers to the update ordering mechanisms across replicas of an object and can be one of causal, serializable or PRAM. Consistency granularity refers to the grain size at which consistency needs to be maintained.
- *sync* refers to synchronization among the replicas. The two dimensions of synchronization are how often the replicas are synchronized and the mode of synchronization (push/pull).
- *workload* can be broken down into workload intensity (number of transactions per second or number of clients) and workload service demand characterization (CPU time for operations).
- *faultload* refers to the failure sequences and the number as well as location of the replicas.

The scalability model given above is useful to identify bottlenecks in distributed systems. By applying the scalability model on shared object spaces, we have identified the key bottlenecks that inhibit existing shared object spaces (with the exception of Globe) from scaling up to the Internet:

- Centralized Components
  Many existing DSMs and shared object spaces have some centralized components that affect their scalability. For instance, Orca has a sequencer for realizing totally ordered group communication, while others like T Spaces [4] have a centralized component for object lookups.
- Failures
  Existing shared object spaces do not handle failures. For instance, JavaSpaces and global shared objects do not handle failures of transaction coordinator, while Orca does not handle failure of the sequencer.
- Object Lookup
  Given an object identifier (id), efficient mechanisms must exist that maps the id to the node that either stores a replica or stores meta-data about the replica. Existing shared object spaces such as T Spaces use centralized lookup mechanisms. Object lookup mechanisms in distributed object middleware such as CORBA and DCOM also have difficulty in handling failures and scaling up.
- Consistency
  Several existing DSM systems such as TreadMarks, Munin and shared object spaces such as JavaSpaces provide relaxed consistency mechanisms such as release consistency and entry consistency. Relaxed consistency mechanisms have also been explored in other areas [66, 52]. However, to our knowledge,

these mechanisms have not been evaluated in Internet scale systems. Peer-to-Peer (P2P) systems which have been scaled to the Internet, such as Pastry [69] and Tapestry [17] assume replicas are read-only.

3. SID Techniques for Data Management.

3.1. Computing Grids. Globus [39] a de-facto standard toolkit for grid computing systems, relies on explicit data transfers between clients and computing servers. It uses the GridFTP protocol [19] that provides authentication based efficient data transfer mechanism for large grids. Globus also allows data catalogues, but leaves catalogue consistency to the application. The paper [51] explores the interfaces required for a Replica Management Service (RMS) that acts as a common entry point for replica catalogue service, meta-data access as well as wide area copy. It does not address consistency issues per se. Further, the RMS is centralized and may not scale up. The other grid paper that has addressed data management issues [29] outlines possible use-cases and gives higher level view of the data management requirements in a grid. The quorum scheme it describes for handling read-write may have to be modified in an Internet kind of an environment to handle quorum dynamics. Further, it does not address various granularities of replication and uses locks for synchronization. The paper [78] also addresses read-write data consistency in a grid environment based on a lazy update propagation algorithm. The update propagation algorithm is based on timestamps and may not scale up to work in a large scale grid environment (Update conflicts are handled manually by application programmer - non-trivial task). Attempts have also been made to extend the existing 2Phase Commit (2PC) based algorithms [82]. These would need global agreement and may be expensive in an Internet setting.

3.2. Data Grids. A generic architecture for handling large data sets in grid computing environments has been proposed in [27]. It describes the way data grid services such as replication and replica selection can be built over basic services of data and meta-data access. It assumes that replicas (file instances) are read-only.

GriPhyN [31] attempts to support large-scale data management in High Energy Physics (HEP) applications as well as for astronomy and gravitational wave physics. GriPhyN provides users transparent access to both raw and processed data (The term virtual data is used to refer to both). It can convert raw data to processed data by scheduling required computations and data transfers. GriPhyN is built on top of Globus. It takes application meta-data and maps it into a Directed Acyclic Graph (DAG), which is an abstract representation of the required actions on data sets. A request planner takes the DAG and transforms it into a concrete DAG, which can be executed by a grid scheduling system such as Condor-G [42].

CERN, the European organization for nuclear research, is also involved in handling computation on large data sets in the HEP area. Object level as well as file level replication for data grids has been explored in [79], a CERN effort. It also assumes files are read only and can be replicated without need for consistency protocols. They support replica catalogs to handle meta-data. Actual file/object transfers are achieved using GridFTP [19].

Data related activities on the grid such as queuing, monitoring and scheduling need to be carefully managed, as data could become bottleneck for data intensive applications. Currently, these data related tasks are performed manually or by simple scripts. The main goal of Stork [85] was to make data a first class citizen on the grid. Data placement jobs have different characteristics from compute intensive jobs and so, may have to be treated differently. Stork is a separate scheduler for scheduling and managing data intensive jobs on grid. Data related activities are represented in the form of a DAG. Stork can interact with higher level planners such as Directed Acyclic Graph Manager (DAGman) which is a part of CondorG. Enhancements have been made to DAGman to make it submit compute intensive jobs to grid schedulers such as CondorG and data intensive jobs to Stork. Stork also supports different heterogeneous storage systems and various data transfer protocols. Case studies have demonstrated the use of Stork as a pipeline between two heterogeneous storage systems and for runtime adaptation of data transfers.

3.3. Content Distribution Networks. Web servers had difficulty in handling the *flash crowd* problem. The *flash crowd* problem refers to a large number of requests coming in suddenly, overwhelming the server's bandwidth, or CPU or back-end transaction infrastructure. Web servers have bursty request nature, for instance during a football match in World Cup or during an election counting process, resulting in the flash crowd problem. Content Distribution Networks (CDNs) such as Akamai [43] have been proposed to handle this problem and to enable web servers to scale up. A separate infrastructure of dedicated servers spread across the Internet was built by several companies to offload content distribution from web servers or to deliver content from the edge of the Internet. Akamai's CDN consists of over twelve thousand servers across thousand different networks. They use either URL rewriting or DNS interposition to redirect client requests to the proximal CDN server.

Studies have shown that caching is beneficial in CDNs as they mainly deliver images or videos (static content) [44]. Akamai CDNs achieved cache hit rates of nearly 88% in another study that compared the CDNs with P2P file sharing systems for distributing content [76]. This shows that CDNs are beneficial for content delivery and can reduce response time for clients. However, another study has shown that the average response time for clients is not affected by employing CDNs [44]. But they avoid worst case of badly performing servers rather than routing client requests to an optimal CDN server.

Cache consistency becomes a challenging issue in order to deliver non-static content to clients. Traditional caching mechanisms such as leasing [22] may not be directly applicable to CDNs. Origin servers would have to keep track of each CDN proxy that caches an object (web document) from the server. It must also manage the lease related issues for that CDN proxy, including notifying the CDN proxy on updates to the object. The CDN proxy has to renew the lease to receive further notifications. Mechanisms for CDNs must be scalable, requiring the CDN proxies to cooperatively maintain consistency. Cooperative leases has been proposed as a scalable mechanism for maintaining cache consistency in CDNs. [12, 11]. Each object is assigned a Δ parameter, which indicates the time or the rate $1/\Delta$ at which an origin server notifies interested CDN proxies of updates to that object. This allows consistency to be relaxed implying that CDN proxy can be notified only once every Δ time units, instead of after every update. Leases are cooperative, meaning that a CDN proxy acts as a leader for a CDN proxy group for lease related interactions with an origin server. The leader is responsible for notifying the other CDN proxies. This reduces both the state maintained at the origin server and the number of updates it must send.

3.4. Data Management in Distributed Mobile Systems. Distributed Mobile Systems (DMS) are distributed systems in which some nodes may be mobile and may have constraints. These constraints could be battery or memory or computing power related. Data could either be stored on or be accessed from mobile devices. Different kinds of management have been identified, with respect to the level of transparency to applications in [54]. Client transparent adaptation allows applications to seamlessly access data without being aware of mobility, with the system providing complete support. The other extreme is a *laisse-faire* model in which adaptation is entirely at user level, with the system providing no support. There are a wealth of strategies between the two extremes, that allow applications to be aware of mobility in varying degrees including application aware adaptation and extended client server models.

Coda [74] was one of the early file systems that allows clients to seamlessly access information, an example of client transparent adaptation. The main goal of Coda was to enable operations to be performed on a shared data repository, even in the face of disconnected operations. Disconnections may be frequent in DMS. *Venus* is the cache manager on each client that manages the cache, hiding mobility from the application. Venus caches volume mappings, with a volume referring to a subtree of the Coda namespace. In the face of connected operations, Coda uses server replication and callback based cache coherence to ensure session semantics (contents will be latest when a session is starting and after it ends) for applications. During disconnections, Venus relies on cache contents and propagates failure to application when a cache miss occurs. When disconnection ends, Coda reverts back to server replication by using reintegration operations using logs.

Application aware adaption has been used in the Odyssey system [21]. Odyssey provides a clean separation between the concerns of the system and the application: system monitors resource dynamics and notifies applications if required, but retains control of resource allocation mechanism; while applications specify mapping of resource levels to *fidelity* levels. Fidelity is defined as the degree to which client data matches with server's. It has multiple dimensions of consistency, frame rate and image quality for video data as well as resolution for spatial data. Building a system that allows diverse fidelity levels necessitates type awareness - client code is responsible for handling particular data types. This is achieved through the use of *wardens*, which are specialized code components that encapsulate system level support at the client. Wardens are subordinate to *Viceroy*, which is responsible for centralized resource management.

Odyssey is an example of client based application aware adaptation. Rover [13] is a system that allows client-server adaptation. This means that some code required for adaption would also reside in server. Rover uses the concept of Relocatable Dynamic Objects (RDOs) for data types handled by the application. The application programmer splits the program containing RDOs into those that reside on the client and those that run on servers. This requires that the adaptation code be resident on origin servers. Another approach has been taken to avoid this, named as proxy based adaptation. The adaptation is done by the proxy, which acts on behalf of clients. The Barwan project [30] is an example. Flexible client server model for application aware adaptation has been proposed in the Bayou system [84]. It allows clients to read/write shared data. Conflicts resolution is handled by using application specific dependency checks and merge procedures. It provides eventual

consistency, an unbounded consistency mechanism that allows replicas to diverge, but be consistent after an unspecified time.

3.5. Software Availability and Usage Summary. Globus is a widely used toolkit and is available as an open source software. Stork is a research prototype, while GriPhyN and CERN have been deployed and used. Akamai's CDNs are widely deployed and used, while cooperative leases [12] is a research prototype. Coda and Odyssey are the distributed mobile systems software that are widely deployed and used.

4. Large Scale Data Management Techniques.

4.1. P2P Data Management. We first give an overview of P2P file sharing systems starting from the initial unstructured P2P systems such as Napster to super-peer systems such as Kazaa before discussing structured P2P systems. We go on to discuss P2P storage management systems such as Oceanstore.

4.1.1. P2P File Sharing Systems. P2P as an area became popular only after the advent of Napster, a file sharing system. Napster [57] was used for sharing music files. Meta-data about files is stored in a global directory, which is stored in a centralized server. The meta-data stored information about music files themselves, which were downloaded from peers. Gnutella [33] came up with a decentralized search protocol for file sharing applications. Gnutella can be seen to be a purely decentralized unstructured P2P system. The term "unstructured" refers to the lack of structure in the overlay, which is mostly a random graph. Search was achieved by flooding the network or by using random walks. Freenet added a mechanism to *route* requests to possible content locations, based on best effort semantics. Freenet also adds a notion of anonymity to the data shared. The main advantage of the unstructured P2P systems was that complex queries could be easily handled. By complex queries, we mean queries such as "get all nodes with processing speed $>$ 3GHz and RAM $>$ 1GB and storage $>$ 100GB". This is because the query is sent to each node and evaluated explicitly. However, deterministic guarantees for searching are difficult to provide in these systems.

Initial attempts at introducing structure to the overlay in P2P systems resulted in super-peer systems, with some nodes (which have better capabilities) acting as super-peers. The other nodes act as clients to the super-peers, which form a P2P overlay among themselves. Super-peers made searching more efficient for complex queries, by exploiting the heterogeneous nature of nodes (some nodes have better capabilities and more importantly, better connectivity than others). An example of a popular super-peer system is Kazaa (`http://www.kazaa.com`). However, handling super-peer failures requires replicating super-peers (otherwise the clients may become disconnected). K-replicas can be created in each cluster, resulting in reduced load on the super-peers [93]. However, this may make replicas client aware. Other design issues in super-peer systems include cluster size and dynamic layer management. A large cluster size is good for aggregate bandwidth, but may create bottlenecks. A small cluster size avoids bottlenecks, but may reduce search efficiency. Dynamic layer management allows nodes to play super-peer or client nodes adaptively, thereby making the super-peer network more efficient [95].

The third generation of P2P systems introduced structure in the overlay network. The motivation came from providing deterministic search guarantees, partitioning the load over the available machines effectively, scaling to large numbers and achieving fault-tolerance. The Distributed Hash Table (DHT) was mainly used as the structure for overlay formation. It was based on the Plaxton data structure [23]. Nodes are given identifiers (ids) from an id space. Application objects are also given ids from the same space. The DHT provides a mapping from the application object id (key) to the node id that is responsible for that key. Each node has a routing table consisting of neighbours and performs routing functions to lookup objects. Various DHTs have been proposed, each having different routing algorithms and routing table maintenance. Geometric interpretations of DHTs have been given in [45] (but the focus of that paper was mainly to study the static resilience of DHTs). Chord [40] is based on a ring, while Content Addressable Network (CAN) is based on a hypercube, Plaxton data structure is based on a tree, while Pastry [69] is a hybrid geometry combining the tree and the ring. We discuss some of these structured P2P systems in more detail below.

Chord provides the lookup abstraction of DHTs through the method: lookup(key) which maps a key to a node responsible for it. Chord uses consistent hashing to assign m-bit identifiers to both Chord nodes and application objects. The ids are arranged in a ring fashion (modulo 2^m). A key k maps to the first node whose id is equal to or follows k in the identifier space (this node is known as successor(k)). Each node maintains a pointer to its successor in the ring. Routing proceeds along the ring till a key is straddled between two node ids, with the second node id being the destination. Each node also maintains information on $O(\log(N))$ (for N nodes) other nodes in the form of a *finger table* in order to speed up routing. Even if nodes in the finger

table were to fail, only efficiency is affected, but not correctness. As long as each node is able to connect to its successor, routing is guaranteed to finish in $O(\log(N))$ time.

CAN routes over a hypercube. Each CAN node stores a chunk (or zone) of the hash table. Each node also stores information on adjacent zones in the table. This is again to speed up routing. Lookup requests for a particular key are routed towards a CAN node whose zone contains that key. Requests are routed by correcting bits (n bits for a n-dimensional hypercube). Generally tree based DHTs such as the Plaxton data structure allow bits to be corrected in order (from MSB to LSB of key), while hypercube based DHTs allow bit correction in any order. This makes routing more resilient to node/link failures.

Pastry can be viewed as having a hybrid geometry due to its use of tree based routing and ring like neighbour formation. It provides a *route* abstraction to applications. The *route(msg, key)* ensures that the message with a given id is routed to a node with the closest matching id as key among all live nodes. Each node keeps track of its immediate neighbours in the node id space by maintaining leaf sets. They also store information about a few other nodes that have prefix matching ids in the form of a routing table. Pastry takes into account network locality in routing. This means that a given message will be routed to the nearest node that is alive and that has the closest matching id as the key. Routing takes place by prefix matching, with each hop taking the message one bit closer in the node id space, resulting in $O(\log(N))$ hops.

4.1.2. P2P File Storage Systems. Ivy [56] is a read/write P2P file system that provides an NFS-like abstraction for programmers. Ivy provides NFS-like semantics in a failure free environment. Under network partitions and failures, Ivy uses logs to allow applications to detect and resolve conflicts. Ivy logs are specific to each participant and host. The logs are stored in DHash, a DHT based P2P block storage system over which Ivy is built. Participants can *read* other logs, but write only his/her log while updating the file system. Ivy uses versioning vectors to detect conflicting updates and provides information to application level conflict resolvers. Ivy system demonstrated a performance within 2-3 factor of NFS performance in a WAN testbed.

PAST [15] is an Internet based P2P storage utility. It offers persistent storage services, availability, security and scalability. PAST provides *insert*, *reclaim* and *retrieve* operations on files. Since a file cannot be inserted multiple times, files are assumed to be immutable in PAST. It must be noted that PAST is an extension of Pastry to provide a file storage system. On insertion of a file into PAST, the file is routed by Pastry to k-nodes with closest matching ids as the file id and that are alive. The set k will be diverse with respect to location, capabilities and connectivity due to the randomization of the identifier space. File availability is ensured as long as all k nodes do not fail simultaneously. It provides security using optional smartcards that are based on a public-key cryptosystem.

Oceanstore [49] is an Internet based file system that provides persistence and availability of files by using a two-tiered system. The upper tier consists of capable machines with good connectivity. These machines act as an *inner* circle of servers for serializing updates. The lower tier consists of less capable machines which only provide storage resources to the system. Pond [67] is an Oceanstore realization that provides fault tolerant durable storage to applications. It uses erasure coding to store data. Erasure coding [20] is a technique that allows a block to be split into m fragments, which are encoded into n fragments ($n > m$). The key property of erasure coding is that it ensures that the block can be reconstructed from any m of the n coded fragments. Oceanstore uses Tapestry [17], another DHT, to store the erasure coded fragments (based on fragment number + block id). Oceanstore uses primary copy replication to ensure consistency of file blocks. It handles read/write data by a versioning mechanism in which any write operation creates a new version of the data. The problem is then reduced to one of finding the most recent version of the file.

4.1.3. Observations. Ivy has the disadvantage that it leaves write conflict resolution to the application, limiting the scalability. PAST provides a persistent caching and storage management layer on top of Pastry. It provides *insert*, *lookup* and *reclaim* operations on files. However, it also assumes files are immutable, as files cannot be inserted multiple times with the same id. Oceanstore's versioning mechanism has not been proved scalable. The evaluations on Oceanstore and Pond [67] have not considered conflicting write operations and have assumed there is a single write per data block. Moreover, Oceanstore assumes an inner circle of reliable servers to ensure consistency. Further, all the three storage systems (Ivy, PAST and Oceanstore) have been built over DHTs. DHTs provide support for only limited queries (exact matching kind) and may not allow application specific criterion for data placement. In the words of [47], virtualization (through DHTs) "destroys locality and application specific information". However, there have been recent efforts that enable DHTs to handle advanced queries such as those handled in [18].

4.2. P2P Extensions to DDBMS. A simplistic view of a traditional distributed database management system is that it uses a centralized server to provide a global schema and ACID properties through transactions. Several approaches have extended these techniques to work in a decentralized manner, to apply to Internet or P2P systems. Active XML [9] provides dynamic XML documents over web services for distributed data integration. It is a model for replicating (whole file) and distributing (parts of a file) XML documents by introducing location aware queries in X-Path and X-Query. It also provides a framework by which peers perform decentralized query processing in the presence of distribution and replication. It allows peers to optimize localized query evaluation costs, by a series of replication steps.

Edutella [58] attempts to design and implement a schema based P2P infrastructure for the semantic web. It uses W3C standards RDF and RDF Schema as the schema language to annotate resources on the web. It uses RDF-QEL as an expressive query exchange language to retrieve the data stored in the P2P network. It uses super-peer routing indices that include schema and other index information.

Piazza [83] is a peer data management system that facilitates decentralized sharing of heterogeneous data. Each peer contributes schemas, mappings, data and/or computation. Piazza provides query answering capabilities over a distributed collection of local schemas and pairwise mappings between them. It essentially provides a schema mediation mechanism for data integration over a P2P system.

P2P Information Exchange and Retrieval (PIER) [38] is a P2P query engine for query processing in Internet scale distributed systems. PIER provides a mechanism for scalable sharing and querying of finger print information, used in network monitoring applications such as intrusion detection. It provides best effort results, as achieving ACID properties may be difficult in Internet scale systems. The query engine does not assume data is loaded into databases on all peers, but is available in their *natural habitats* in file systems. PIER is realized over CAN, the hypercube based P2P system.

PeerDB [60] is an object management system that provides sophisticated searching capabilities. PeerDB is realized over BestPeer [59], which provides P2P enabling technologies. PeerDB can be viewed as a network of local databases on peers. It allows data sharing without a global schema by using meta-data for each relation and attributes. The query proceeds in two phases: in the first phase, relations that match the user's search are returned by searching on neighbours. After the user selects the desired relations, the second phase begins, where queries are directed to nodes containing the selected relations. Mobile agents are dispatched to perform the queries in both phases.

4.3. Software Availability and Usage Summary. Gnutella and Napster have been widely deployed and used. Chord is a research prototype that is also available as an open source software. Pastry is also available as an open source software and has also been used widely. CAN and Ivy are research prototypes about which deployment information is not available. PAST and Oceanstore are research prototypes that have been deployed and used in the Planetlab testbed.

Edutella is available as an open source software. The authors do not have information on the deployment/availability on other research prototypes Piazza, PeerDB and Active XML. PIER has been deployed in the Planetlab testbed.

5. State of the Art Data Management.

5.1. SID Techniques: State of the Art.

5.1.1. P2P Techniques in Grids. JuxMem [2] provides a data sharing service for grids by integrating DSM concepts with P2P systems. It is realized over (Juxtapose) JXTA [34], an emerging framework for developing P2P applications. JuxMem uses cluster advertisements to advertise the amount of memory each peer can provide to the global storage. It is organized into a federation of clusters, with each cluster having a Cluster Manager (CM). The CM is responsible for storing all cluster advertisements in its group. The CMs across clusters form a DHT. Actually, the amount of memory provided in the cluster advertisement is hashed and the CM with the closest matching id in the DHT stores this advertisement. When a client asks for a block of memory with a given rounded size (fixed sized blocks can only be supported), the size is hashed and the cluster advertisement which provides that size is retrieved from the CM with the closest matching id. The cluster advertisement has the details of the actual storage provider. Recent extensions to JuxMem [14] provide mechanisms to decouple consistency protocols from fault-tolerance mechanisms. This allows the use of standard DSM consistency protocols to integrate fault-tolerance components. In particular, DSM consistency schemes such as home based consistency [41] which assume a single home node for serializing concurrent writes, can be made fault-tolerant by having a group of nodes as the home node. This requires group membership protocols, as

well as an atomic multicast protocol, which is achieved by using consensus protocols based on Failure Detectors (FDs) [26]. The data sharing mechanisms of JuxMem have only been evaluated at the cluster level.

The replica location problem has been addressed in grids using P2P concepts in [5]. It proposes a P2P realization of the Replica Location Service (RLS), a key component of data grids. The Logical File Name (LFN) is hashed to give the identifier for a replica. The node with the closest matching id as the LFN hash contains the LFN to Physical File Name (PFN) mapping. This is the meta-data stored in RLS for file lookup. It also proposes an update protocol to handle consistency of meta-data. The RLS realization is based on Kademlia [63]. Kademlia is a structured P2P system that uses a novel XOR metric for routing—distance between two nodes is defined as the eXclusive OR (XOR) of their numeric ids. A Kademlia node forms log(n) neighbours, where neighbour i is at XOR distance $[2^i, 2^{i+1}]$. The neighbour set is same as that formed by a tree based DHT PRR [23]. Even the failure-free routing in Kademlia is similar to PRR, in that bits are corrected from left to right. However, in the case of failures, XOR metric allows bits to be corrected in any order. This implies that the static resilience[2] of Kademlia is better compared to PRR [45].

5.1.2. Replica Placement in CDNs. Optimal placement of replicas in CDNs is a non-trivial task and has not been addressed. QoS aware replica placement was proposed in [92] to meet QoS requirements of clients with the objective of minimizing the replication cost. The replication cost includes cost of storage and consistency management, while QoS is specified in terms of distance metrics such as hop count. Two problems are formulated: Replica-aware and Replica-blind. In replica-aware model, the CDN servers are aware of where object replicas are stored in the CDN network. This helps the servers to redirect client requests to the nearest replica. In the replica blind model, application or network level routing ensures client requests are routed to CDN servers, with servers being transparent to replica location. Each replica (CDN server) serves requests coming to it. Dynamic programming techniques are used to arrive at near optimal solutions for the optimal replica placement problem, which is shown to be NP-complete.

5.1.3. Distributed Mobile Storage System. Segank [80] provides an abstraction of a shared storage system for heterogeneous storage elements. The motivation was that traditional mechanisms for managing data in distributed mobile environments such as Coda and Bayou, have time consuming merge operations. In Coda, updates are released to the server before becoming visible on clients. If servers are physically far away, this could increase the time after which updates become visible. Bayou uses full replication, leading to potentially expensive merge operations. Segank handles data location problem when data could be located on any subset of devices, by using a location and topology sensitive multicast-like (named as segankcast) operation. It allows lazy P2P propagation of invalidation information to handle consistency of replicated data. It also uses a distributed snapshot mechanism to ensure a consistent image across all devices for backup. It must be observed that Segank uses only unstructured P2P system concepts. This implies that Segank cannot provide deterministic search guarantees.

5.2. Large Scale Data Management: State of the Art. We shall explain the current state of the art in P2P data management along four directions: integrating structured and unstructured P2P systems providing Quality of Service (QoS) guarantees in P2P systems, composable consistency for P2P systems and large scale DHT deployment. We also explain the state of the art in P2P DBMS.

5.2.1. Integrating Structured and Unstructured P2P Systems. An attempt has been made in [55] to improve structured P2P systems along three directions where they were traditionally known to perform worse compared to unstructured P2P systems: handling churn, exploiting heterogeneity and handling complex queries. In P2P systems, node/network dynamics resulting in routing-table updates and/or data movement is known as churn. The paper [55] shows that MS Pastry, an implementation of Pastry, can handle churn well by using a periodic routing table maintenance protocol. This protocol updates failed routing table entries. It also has a passive routing table repair protocol. They demonstrate that by exploiting structure, MS Pastry can handle churn better than unstructured P2P systems. Heterogeneity is difficult to handle in structured P2P systems due to constraints on data placement and neighbour selection. MS Pastry handles heterogeneity in two ways: one by using super-peer concepts; second, by modifying neighbour selection to handle capacity. MS Pastry is also extended to handle complex queries by introducing new techniques for flooding or random walks. Flooding is achieved by sending the message to all nodes in the routing table. Random walk is achieved by using a tag containing the set of nodes to visit, a queue of nodes in the routing table row and a bound on number of rows to traverse. A few other efforts have also been made recently to make structured P2P systems handle

[2]Static Resilience measures the goodness of a DHT routing algorithm before recovery mechanisms take effect

range queries [16], multi-dimensional queries [65] as well a query algebra [73]. A Scalable Wide Area Resource Discovery (SWORD) [62] has been built to realize resource discovery over WANs by supporting multi-attribute range queries over DHTs.

Another approach to integrate structured and unstructured P2P systems has been made in the Vishwa computing grid middleware [53]. Vishwa uses the task management layer to handle initial task deployment and load adaptability of the tasks. The task management layer is realized using unstructured P2P concepts and allows capability based resource clustering. The reconfiguration layer of Vishwa is realized as a structured P2P layer and stores information needed to handle node/network failures. The two layered architecture has also been used for data management in Virat [1, 7]. Virat provides a shared object space abstraction over a wide-area distributed system. Virat has been extended to a replica management middleware for P2P systems [8]. The unstructured layer forms neighbours based on node capabilities (in terms of processing power, memory available, storage capacity and load conditions). A structured DHT is built over this unstructured layer by using the concept of virtual nodes. Virat achieves dynamic replica placement on nodes with given capabilities, which would be very useful in computing/data grids. Detailed performance comparison is also made with a replica mechanism realized over OpenDHT [68], a state of the art structured P2P system. It has been demonstrated that the 99th percentile response time for Virat does not exceed 600 ms, whereas for OpenDHT, it goes beyond 2000 ms in an Internet testbed.

5.2.2. Composable Consistency for P2P Systems. A flexible consistency model known as composable consistency suitable for a variety of P2P applications has been proposed in [72]. The authors have initially surveyed consistency requirements for P2P applications such as personal file access, real time collaboration and database or directory services. The survey showed that different applications need different semantics for read/write and for replica divergence. The main contribution of [72] is the classification of consistency requirements along five orthogonal dimensions: concurrency—degree of conflicting read/write access; replica synchronization—degree of replica divergence; failure handling—data access semantics in the presence of inaccessible replicas; update visibility - time after which local updates may be made globally visible; view isolation—time after which remote updates must be made locally visible. A rich collection of consistency semantics for shared data can be *composed* by combining the above five options. Performance studies have shown that composable consistency in the Swarm system outperforms CoDA [74] in a file sharing scenario, while for a replicated BerkeleyDB database, it provides different consistency mechanisms from strong to time-based.

5.2.3. Providing QoS Guarantees in P2P Systems. Guaranteeing Quality of Service (QoS) parameters such as response time or throughput in P2P systems is a challenging task. An initial attempt was made in [70] at using P2P system concepts for Domain Name System (DNS), which requires efficient data location. It showed that though P2P DNS could provide better fault-tolerance than conventional DNS, lookup performance of O(log(N)) provided by DHTs was far worse compared to conventional DNS. Cooperative DNS (CoDoNS) [89] was proposed to tackle three problems of conventional DNS: susceptibility to Denial of Service (DoS) attacks; lookup delays, especially for flash crowds; lack of cache coherency, preventing quick service relocation in emergencies. CoDoNS has been proposed as a backward compatible replacement for conventional DNS. It provides O(1) lookup time by using the proactive caching layer of Beehive [88]. Beehive enables DHTs to achieve O(1) lookup performance by proactive replication. Traditionally, prefix matching DHTs store an application object at the closest matching node, with each routing step successively matching prefixes, resulting in O(log(N)) lookup performance. By aggressively caching the object all along the lookup path, Beehive achieves O(1) lookup performance for that object. Since, Beehive associates different replication levels for different application objects, an average lookup performance of O(1) is achieved. CoDoNS builds a DNS based on a self-organizing P2P overlay formed across organizations (if each organization can provide a server for CoDoNS). CoDoNS associates a domain name with the node having the closest matching id as the domain name's hashed id. If the home node fails, the node with the next best matching id takes over as the home node for that particular domain. Performance studies over PlanetLab testbed show that CoDoNS achieves lower lookup latencies, can handle slashdot effects and can quickly disseminate updates. However, the use of DHTs as the basis leaves CoDoNS vulnerable to network partitions. For example, if an organization is partitioned from the outside world, while conventional DNS would ensure that local lookups worked correctly, with CoDoNS even local lookups may fail (DHT lookup may go outside the local network even for local lookups—stretch property of DHTs). This suggests that SkipNets [35] may be a better choice for realizing DNS than DHTs. This is because data in SkipNets is organized by using string names which guarantees routing locality. This is in addition to the normal numeric identifier based organization used in DHTs.

5.2.4. Large Scale Deployment. OpenDHT [68] is a public large scale DHT deployment that allows clients to use DHTs without having to deploy them. It provides a shared storage space abstraction using the *get* and *put* primitives. The main motivation for OpenDHT is that it is hard to deploy long running distributed system services, especially in the public domain. OpenDHT is deployed on PlanetLab (http://www.planet-lab.org/), a global testbed for deploying planetary scale services. OpenDHT is deployed on *infrastructure* nodes which alone participate in DHT routing and storage. Clients only *use* the storage space through the *get* and *put* interface on gateway (infrastructure) nodes. OpenDHT allows different mutually untrusting applications to share the DHT. It ensures that clients get a fair share of storage resources without imposing arbitrary quotas—a trade-off between fairness and flexibility. This is achieved by associating a Time-to-Live (TTL) with application objects and letting them expire if clients do not renew them. OpenDHT provides *storage* abstraction of DHTs in contrast to the *lookup* abstraction of Chord or the *routing* abstraction of Pastry.

It is realized over Bamboo DHT(bamboo-dht.org), that is similar to Pastry but has differences in handling node dynamics. OpenDHT is not a shared object space. The level of abstraction provided to programmer is different. For instance, the programmer has to take care of object serialization, RTTI (runtime type inferencing) etc. to realize an object storage on top of the byte storage that OpenDHT provides. OpenDHT provides limited consistency for the shared byte space. Conflict resolution (for concurrent writes) is left to the application, similar to the Bayou system that ensures "eventual consistency", a very loose form of consistency. But conflict resolution is a non-trivial task for the application programmer. The performance of OpenDHT (especially worst case response time) suffers due to the presence of stragglers or slow nodes. This has been improved by using delay aware and iterative routing in [71].

5.2.5. State of the Art P2P DDBMS. Atlas P2P Architecture (APPA) [86] is the current state of the art data management solution for large scale P2P systems. It uses a three layered architecture, with the P2P network forming the lowest layer. This layer could be realized using unstructured or structured or super-peer based P2P concepts. Above this layer, the basic P2P services layer is built. This provides P2P data sharing and retrieving (key based) in the P2P network, support for peer communication, support for peer dynamics (join and leave) and group membership management. Over the basic services layer advanced P2P data management services such as schema management, replication, query processing and security are built. The shared data is in XML format and queries expressed in X-Queries in order to make use of web services. It is realized over JXTA. It provides replica management by extending traditional centralized log based reconciliation techniques for P2P systems. It assumes the existence of a shared storage space for distributed reconciliation by peers. This requires consensus protocols for realization and may be expensive. It has not been evaluated in large scale systems.

A recent effort has been made to provide a middleware based data replication scheme in [94] by using *Snapshot Isolation* (SI) as the isolation level. In SI based DBMS, read operations of a transaction T are handled from a snapshot of the database (set of committed transactions when T started). This implies read operations never conflict with write operations and only write-write conflicts can occur, resulting in more concurrency and consequently better performance. It has been proposed at the cluster level and may not be applicable for P2P systems due to its strong assumption of a totally ordered multicast.

5.3. Software Availability and Usage Summary. Juxmem and Segank are research prototypes. Deployment information on Structella is not available. Vishwa and Virat are research prototypes that are available as open binaries. OpenDHT has been deployed on the Planetlab testbed and is also available as an open source software. APPA is a research prototype.

6. Conclusions. We have presented a scalability taxonomy of data management solutions in distributed systems. We group data management work done in DSMs and shared object spaces in the Centralized/Naively Distributed (CND) data management category. The Sophisticated/Intermediate Data (SID) management techniques include data management in grid computing systems and data grids as well as Content Distribution Networks (CDNs) and data management in distributed mobile systems. These solutions scale better than CND techniques by using distributed data management, instead of centralized approaches. They however, assume an inner set of reliable servers which take care of consistency and reliability issues. However, in order to take the data management services to the edges of the Internet, Large Scale Data (LSD) management techniques make use of P2P concepts. They consequently provide better scalability and fault-tolerance, but at the cost of relaxing consistency (most approaches provide probabilistic guarantees or eventual consistency).

The taxonomy is depicted in figure 6.1. The figure shows the state of the art efforts in orange color and the possible future directions also in blue. The future directions are detailed below.

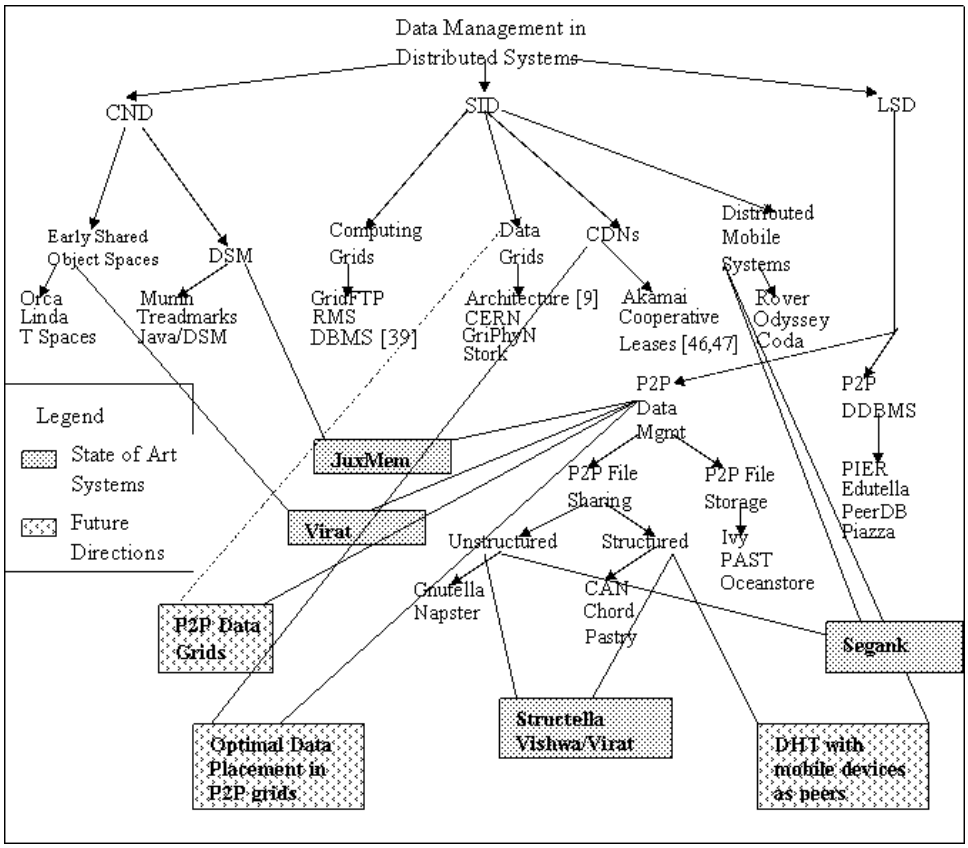


FIG. 6.1. *Pictorial Representation of Scalability Taxonomy*

It can be observed that LSD techniques such as Virat [8] handle large number of small data objects. The case of handling large number of large data objects arises when existing data grids become purely P2P, instead of using SID techniques. The existing LSD techniques may not work in this case, as the size of data objects calls for special mechanisms to handle some operations including updates. Incremental updates or function shipping in combination with LSD data management techniques may have to be explored.

Another interesting avenue for exploration is the use of LSD techniques combined with node mobility. The solutions which have been proposed for handling data management in distributed mobile systems do not use P2P concepts, but assume the presence of reliable servers that handle mobile client requests. When mobile nodes form the P2P overlay, *churn* could be very high due to node mobility. This, coupled with the device constraints, may open up a wealth of research questions.

Optimal data placement techniques which have been proposed for CDNs [92] can be used in P2P grids. Existing data management techniques in grids (or even P2P grids such as P-Grid [46]) do not address optimal replica placement issues. The work [8] provides heuristics for replica placement in P2P grids. But placement of replicas may not be exactly optimal. Thus, we see that techniques for data management in one category can be applied to others to open up research in large scale data management.

REFERENCES

[1] A VIJAY SRINIVAS, M VENKATESHWARA REDDY, AND D JANAKIRAM, *Designing a Replication Service for Large Peer-to-Peer Data Grids*, IEEE Distributed Systems Online, 7 (2006).

[2] GABRIEL ANTONIU, LUC BOUGÉ, AND MATHIEU JAN, *JUXMEM: An Adaptive Supportive Platform for Data Sharing on the Grid*, Scalable Computing: Practice and Experience, 6 (2005), pp. 45–55.

[3] MAARTEN VAN STEEN AND PHILIP HOMBURG AND ANDREW S. TANENBAUM, *Globe: A Wide-Area Distributed System*, IEEE Concurrency, 7 (1999), pp. 70–78.

[4] P WYCKOFF, S W MCLAUGHRY, T J LEHMAN, AND D A FORD, *T Spaces*, IBM Systems Journal, 37 (1998), pp. 454–474.

[5] A. CHAZAPIS, A. ZISSIMOS, AND N. KOZIRIS, *A Peer-to-Peer Replica Management Service for High-Throughput Grids*, in Proceedings of the International Conference on Parallel Processing (ICPP), Washington, DC, USA, June 2005, IEEE Computer Society, pp. 443–451.

[6] A Vijay Srinivas and D Janakiram, *A Model for Characterizing the Scalability of Distributed Systems*, ACM SIGOPS Operating Systems Review, 39 (2005), pp. 64–72.

[7] A Vijay Srinivas and D Janakiram, *A Peer-to-Peer Framework for Collaborative Data Sharing Over the Internet*, Tech. Report IITM-CSE-DOS-2005-28, accepted for publication in IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollobarateCom 2006), IEEE Computer Society Press.

[8] A Vijay Srinivas and D Janakiram, *Node Capability Aware Replica Management for Peer-to-Peer Grids*, Technical Report IITM-CSE-DOS-2006-04, Distributed & Object Systems Lab, Indian Institute of Technology, Communicated to IEEE Transactions on Software Engineering.

[9] S. Abiteboul, A. Bonifati, G. Cobéna, I. Manolescu, and T. Milo, *Dynamic XML documents with distribution and replication*, in SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, New York, NY, USA, 2003, ACM Press, pp. 527–538.

[10] C. Amza, A. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel, *TreadMarks: Shared Memory Computing on Networks of Workstations*, IEEE Computer, 29 (1996), pp. 18–28.

[11] Anoop George Ninan, Purushottam Kulkarni, Prashant Shenoy, Krithi Ramamritham, and Renu Tewari, *Scalable Consistency Maintenance in Content Distribution Networks Using Cooperative Leases*, IEEE Transactions on Knowledge and Data Engineering, 15 (2003), pp. 813–828.

[12] Anoop Ninan, Purushottam Kulkarni, Prashant Shenoy, Krithi Ramamritham, and Renu Tewari, *Cooperative Leases: Scalable Consistency Maintenance in Content Distribution Networks*, in WWW '02: Proceedings of the 11th international conference on World Wide Web, New York, NY, USA, 2002, ACM Press, pp. 1–12.

[13] Anthony D Joseph, Joshua A Tauber, and M Frans Kaashoek, *Mobile Computing with the Rover Toolkit*, IEEE Transactions on Computers, 46 (1997), pp. 337–352.

[14] G. Antoniu, J.-F. Deverge, and S. Monnet, *How to Bring Together Fault Tolerance and Data Consistency to Enable Grid Data Sharing*, Concurrency and Computation: Practice and Experience, 17 (2006). To appear.

[15] Antony Rowstron and Peter Druschel, *Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility*, in SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles, New York, NY, USA, 2001, ACM Press, pp. 188–201.

[16] Artur Andrzejak and Zhichen Xu, *Scalable, Efficient Range Queries for Grid Information Services*, in P2P '02: Proceedings of the Second International Conference on Peer-to-Peer Computing, Washington, DC, USA, 2002, IEEE Computer Society, pp. 33–40.

[17] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, *Tapestry: A Resilient Global-Scale Overlay for Service Deployment*, IEEE Journal on Selected Areas in Communications, 22 (2004), pp. 41–53.

[18] D. Bauer, P. Hurley, R. Pletka, and M. Waldvogel, *Bringing efficient advanced queries to distributed hash tables*, in LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN'04), Washington, DC, USA, 2004, IEEE Computer Society, pp. 6–14.

[19] Bill Allcock, Joe Bester, John Bresnahan, Ann L. Chervenak, Ian Foster, Carl Kesselman, Sam Meder, Veronika Nefedova, Darcy Quesnel, and Steven Tuecke, *Data Management and Transfer in High-Performance Computational Grid Environments*, Parallel Computing, 28 (2002), pp. 749–771.

[20] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, *An xor-based erasure-resilient coding scheme*, 1995.

[21] Brian D Noble, M Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker, *Agile Application-Aware Adaptation for Mobility*, in SOSP '97: Proceedings of the sixteenth ACM symposium on Operating Systems Principles, New York, NY, USA, 1997, ACM Press, pp. 276–287.

[22] C Gray and D Cheriton, *Leases: an Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency*, in SOSP '89: Proceedings of the twelfth ACM symposium on Operating systems principles, New York, NY, USA, 1989, ACM Press, pp. 202–210.

[23] C Greg Plaxton, Rajmohan Rajaraman, and Andrea W Richa, *Accessing Nearby Copies of Replicated Objects in a Distributed Environment*, in SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel Algorithms and Architectures, New York, NY, USA, 1997, ACM Press, pp. 311–320.

[24] N. Carriero and D. Gelenter, *Linda in Context*, Communications of the ACM, 4 (1989), pp. 444–458.

[25] J. B. Carter, *Design of the Munin Distributed Shared Memory System*, Journal of Parallel and Distributed Computing, 29 (1995), pp. 219–227.

[26] T. D. Chandra and S. Toueg, *Unreliable Failure Detectors for Reliable Distributed Systems*, Journal of the ACM, 43 (1996), pp. 225–267.

[27] Chervenak, A, Foster, I, Kesselman, C, Salisbury, C, and Tuecke, S, *The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets*, Journal of Network and Computer Applications, 23 (2001), pp. 187–200.

[28] U. Dayal, K. Ramamritham, and T. M. Vijayaraman, eds., *Proceedings of the 19th International Conference on Data Engineering, March 5-8, 2003, Bangalore, India*, IEEE Computer Society, 2003.

[29] Dirk DÃijllmann and Ben Segal, *Models for Replica Synchronisation and Consistency in a Data Grid*, in HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), Washington, DC, USA, 2001, IEEE Computer Society, p. 67.

[30] Eric A Brewer, Randy H Katz, Elan Amir1, Hari Balakrishnan, Yatin Chawathe, Armando Fox, Steven D Gribble, Todd Hodes, Giao Nguyen, Venkata N Padmanabhan, Mark Stemm, Srinivasan Seshan, Tom Henderson, Joshua A Tauber, and M Frans Kaashoek, *A Network Architecture for Heterogeneous Mobile Computing*, IEEE Personal Communications, 5 (1998), pp. 8–24.

[31] Ewa Deelman, Carl Kesselman, Gaurang Mehta, Leila Meshkat, Laura Pearlman, Kent Blackburn, Phil Ehrens, Albert Lazzarini, Roy Williams, and Scott Koranda, *GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists*, in Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 20002 (HPDC'02), Washington, DC, USA, 2002, IEEE Computer Society, p. 225.

[32] A. Finkelstein, C. Gryce, and J. Lewis-Bowen, *Relating Requirements and Architectures: A Study of Data-Grids*,

Journal of Grid Computing, 2 (2004), pp. 207–222.
[33] Gnutella, *The Gnutella protocol specification v0.4.* http://www9.limewire.com/developer/gnutella protocol 0.4.pdf 2000.
[34] L. Gong, *JXTA: A Network Programming Environment*, IEEE Internet Computing, 5 (2001), pp. 88–95.
[35] Harvey, Nicholas J. A., Jones, Michael B., Saroiu, Stefan, Theimer, Marvin, and Wolman, Alec, *Skipnet: A scalable overlay network with practical locality properties*, in Proceedings of the Fourth USENIX Symposium on Internet Technologies and Systems (USITS '03), Seattle, United States, March 2003, USENIX Association.
[36] Henri E Bal, M Frans Kaashoek, and Andrew S Tanenbaum, *Orca: A Language for Parallel Programming of Distributed Systems*, IEEE Transactions on Software Engineering, 18 (1992), pp. 190–205.
[37] Henri E Bal, Raoul Bhoedjang, Rutger Hofman, Ceriel Jacobs, Koen Langendoen, Tim Ruhl, and M Frans Kaashoek, *Performance evaluation of the orca shared-object system*, ACM Transactions on Computer Systems, 16 (1998), pp. 1–40.
[38] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica, *Querying the Internet with PIER.*, in VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany, J. C. Freytag, P. C. Lockemann, S. Abiteboul, M. J. Carey, P. G. Selinger, and A. Heuer, eds., Morgan Kaufmann, 2003, pp. 321–332.
[39] I. Foster and C. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit*, Intl Journal of Supercomputer Applications, 11 (1997), pp. 115–128.
[40] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, *Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications*, IEEE/ACM Transactions on Networking, 11 (2003), pp. 17–32.
[41] L. Iftode, J. P. Singh, and K. Li, *Scope Consistency: a Bridge Between Release Consistency and Entry Consistency*, in SPAA '96: Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures, New York, NY, USA, 1996, ACM Press, pp. 277–287.
[42] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, *Condor-G: A Computation Management Agent for Multi-Institutional Grids*, in HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), Washington, DC, USA, 2001, IEEE Computer Society, p. 55.
[43] John Dilley, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, and Bill Weihl, *Globally Distributed Content Delivery*, IEEE Internet Computing, 06 (2002), pp. 50–58.
[44] K. L. Johnson, J. F. Carr, M. S. Day, and M. F. Kaashoek, *The measured performance of content distribution networks*, Computer Communications, 24 (2001), pp. 202–206.
[45] K Gummadi, R Gummadi, S Gribble, S Ratnasamy, S Shenker, and I. Stoica, *The Impact of DHT Routing Geometry on Resilience and Proximity*, in SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, 2003, ACM Press, pp. 381–394.
[46] Karl Aberer, Philippe Cudre-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt, *P-Grid: a Self-Organizing Structured P2P System*, ACM SIGMOD Record, 32 (2003), pp. 29–33.
[47] P. J. Keleher, B. Bhattacharjee, and B. D. Silaghi, *Are Virtualized Overlay Networks Too Much of a Good Thing?*, in IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, London, UK, 2002, Springer-Verlag, pp. 225–231.
[48] Kourosh Gharachorloo, Daniel Lenoski, James Laudon, Phillip Gibbons, Anoop Gupta, and John Hennessy, *Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors*, in ISCA '90: Proceedings of the 17th annual international symposium on Computer Architecture, New York, NY, USA, 1990, ACM Press, pp. 15–26.
[49] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao, *OceanStore: an Architecture for Global-Scale Persistent Storage*, SIGARCH Computer Architecture News, 28 (2000), pp. 190–201.
[50] L G Alex Sung, Nabeel Ahmed, R. andHerman Li, Mohamed Ali Soliman, and David Hadaller, *A Survey of Data Management in Peer-to-Peer Systems.* CS856 Web Data Management, 2005. School of Computer Science, University of Waterloo.
[51] L Guy, P Kunszt, E Laure, H Stockinger, and K Stockinger, *Replica Management in Data Grids.* Technical Report, GGF Working Draft, 2002.
[52] M. Ahamad and R. Kordale, *Scalable Consistency Protocols for Distributed Services*, IEEE Transactions on Parallel and Distributed Systems, 10 (1999), pp. 888–903.
[53] M. V. Reddy, A. V. Srinivas, T. Gopinath, and D. Janakiram, *Vishwa: A Reconfigurable Peer-to-Peer Middleware for Grid Computing*, in 35th International Conference on Parallel Processing, IEEE Computer Society Press, 2006, pp. 381–390.
[54] Mahadev Satyanarayanan, *Accessing Information on Demand at any Location. Mobile Information Access*, IEEE Personal Communications, 3 (1996), pp. 26–33.
[55] Miguel Castro, Manuel Costa, and Antony Rowstron, *Debunking Some Myths About Structured and Unstructured Overlays*, in Proceedings of the 2nd Usenix Symposium on Networked System Design and Implementation, Boston,MA, May 2005.
[56] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen, *Ivy: a Read/Write Peer-to-Peer File System*, SIGOPS Operating Systems Review, 36 (2002), pp. 31–44.
[57] NAPSTER, *Napster media sharing system.* http://www.napster.com
[58] W. Nejdl, W. Siberski, and M. Sintek, *Design issues and challenges for RDF- and schema-based peer-to-peer systems*, SIGMOD Record, 32 (2003), pp. 41–46.
[59] W. S. Ng, B. C. Ooi, and K.-L. Tan, *BestPeer: A Self-Configurable Peer-to-Peer System.*, in Proceedings of the 18th International Conference on Data Engineering, 26 February - 1 March 2002, San Jose, CA, IEEE Computer Society, 2002, p. 272.
[60] W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou, *PeerDB: A P2P-based System for Distributed Data Sharing.*, in Dayal

et al. [28], pp. 633–644.

[61] Object Management Group, *The Common Object Request Broker: Architecture and Specification*. 2. 3. 1, October 1999.

[62] Oppenheimer, D., Albrecht, J., Patterson, D., and Vahdat, A., *Design and Implementation Tradeoffs for Wide-area Resource Discovery*, in Proceedings. 14th IEEE International Symposium on High Performance Distributed Computing, 2005. HPDC-14, Washington, DC, USA, July 2005, IEEE Computer Society, pp. 113–124.

[63] Petar Maymounkov and David Mazires, *Kademlia: A Peer-to-Peer Information System Based on the XOR Metric*, in IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, London, UK, 2002, Springer-Verlag, pp. 53–65.

[64] Peter J Keleher, *The Relative Importance of Concurrent Writers and Weak Consistency Models*, in ICDCS '96: Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS '96), Washington, DC, USA, 1996, IEEE Computer Society, p. 91.

[65] Prasanna Ganesan, Beverly Yang, and Hector Garcia-Molina, *One Torus to Rule Them All: Multi-Dimensional Queries in P2P Systems*, in WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases, New York, NY, USA, 2004, ACM Press, pp. 19–24.

[66] M. Raynal, G. Rhia-kime, and M. Ahamad, *Serializable to Causal Transactions for Collaborative Applications*, in Proceedings of the 23rd Euromicro Conference, Budapest, Hungary, September 1997.

[67] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz, *Pond: The OceanStore Prototype*, in Proceedings of the Conference on File and Storage Technologies, USENIX Association, 2003.

[68] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, *OpenDHT: a public DHT service and its uses*, in SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, 2005, ACM Press, pp. 73–84.

[69] A. Rowstron and P. Druschel, *Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems*, in Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Midleware 2001), Heidelberg, Germany, November 2001, pp. 329–350.

[70] Russ Cox, Athicha Muthitacharoen, and Robert Morris, *Serving DNS Using a Peer-to-Peer Lookup Service*, in IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, London, UK, 2002, Springer-Verlag, pp. 155–165.

[71] S. Rhea, B. G. Chun, J. Kubiatowicz, and S. Shenker, *Fixing the Embarrassing Slowness of OpenDHT on PlanetLab*, in Proceedings of USENIX WORLDS 2005, USENIX Association, 2005.

[72] Sai Susarla and John Carter, *Flexible Consistency for Wide area Peer Replication*, in Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS), Washington, DC, USA, 2005, IEEE Computer Society.

[73] K.-U. Sattler, P. Rösch, E. Buchmann, and K. Böhm, *A Physical Query Algebra for DHT-based P2P Systems*, in Proceedings of the 6th Workshop on Distributed Data and Structures, Lausanne, Switzerland, July 2004.

[74] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, *Coda: A Highly Available File System for a Distributed Workstation Environment*, IEEE Transactions on Computers, 39 (1990), pp. 447–459.

[75] T. Seidmann, *Replicated Distributed Shared Memory For The .NET Framework*, in Proceedings of 1st Int.Workshop on C# and .NET Technologies on Algorithms, Computer Graphics, Visualization, Computer Vision and Distributed Computing, Plzen, Czech Republic, February 2003.

[76] Stefan Saroiu, Krishna P Gummadi, Richard J Dunn, Steven D Gribble, and Henry M. Levy, *An Analysis of Internet Content Delivery Systems*, SIGOPS Operating Systems Review, 36 (2002), pp. 315–327.

[77] Stephanos Androutsellis-Theotokis and Diomidis Spinellis, *A Survey of Peer-to-Peer Content Distribution Technologies*, ACM Computing Surveys, 36 (2004), pp. 335–371.

[78] H. Stockinger, *Distributed Database Management Systems and the Data Grid*, in MSS '01: Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems and Technologies, Washington, DC, USA, 2001, IEEE Computer Society, p. 1.

[79] H. Stockinger, A. Samar, K. Holtman, W. E. Allcock, I. Foster, and B. Tierney, *File and Object Replication in Data Grids.*, Cluster Computing, 5 (2002), pp. 305–314.

[80] Sumeet Sobti, Nitin Garg, Fengzhou Zheng, Junwen Lai, Yilei Shao, Chi Zhang, Elisha Ziskind, Arvind Krishnamurthy, and Randolph Y. Wang, *Segank: A Distributed Mobile Storage System*, in FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, 2004, USENIX Association, pp. 239–252.

[81] Sun Microsystems, *JS—JavaSpaces Service Specification*. `http://java.sun.com/products/jini/2.0/doc/specs/html/js-spec.html` 2001.

[82] Sushant Goel, Hema Sharda, and David Taniar, *Atomic Commitment and Resilience in Grid Database Systems*, International Journal of Grid and Utility Computing, 1 (2005), pp. 46–60.

[83] I. Tatarinov, Z. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, X. Dong, Y. Ka diyska, G. Miklau, and P. Mork, *The Piazza Peer Data Management Project*, SIGMOD Record, 32 (2003).

[84] D. B. Terry, K. Petersen, M. Spreitzer, and M. Theimer, *The Case for Non-transparent Replication: Examples from Bayou.*, IEEE Data Engineering Bulletin, 21 (1998), pp. 12–20.

[85] Tevfik Kosar and Miron Livny, *Stork: Making Data Placement a First Class Citizen in the Grid*, in ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04), Washington, DC, USA, 2004, IEEE Computer Society, pp. 342–349.

[86] P. Valduriez and E. Pacitti, *Data Management in Large-Scale P2P Systems.*, in VECPAR, M. J. Daydé, J. Dongarra, V. Hernández, and J. M. L. M. Palma, eds., vol. 3402 of Lecture Notes in Computer Science, Springer, 2004, pp. 104–118.

[87] S. Venugopal, R. Buyya, and K. Ramamohanarao, *A Taxonomy of Data Grids for Distributed Data Sharing, Management and Processing*, ACM Computing Surveys, (2006). To appear.

[88] Venugopalan Ramasubramanian and Emin G Sirer, *Exploiting Power Law Query Distributions for O(1) Lookup Performance in Peer to Peer Overlays*, in Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI), USENIX Association, 2004.

[89] ———, *The Design and Implementation of a Next Generation Name Service for the Internet*, in SIGCOMM '04: Proceedings

of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, 2004, ACM Press, pp. 331–342.

[90] Weijian Fang, Cho-Li Wang, and Francis C M Lau, *On the Design of Global Object Space for Efficient Multi-threading Java Computing on Clusters*, Parallel Computing, 29 (2003), pp. 1563–1587.

[91] Weimin Yu and Alan Cox, *Java/DSM: A Platform for Heterogeneous Computing*, in ACM 1997 Workshop on Java for Science and Engineering Computation, June 1997.

[92] Xueyan Tang and Jianliang Xu, *QoS-Aware Replica Placement for Content Distribution*, IEEE Transactions on Parallel and Distributed Systems, 16 (2005), pp. 921–932.

[93] B. Yang and H. Garcia-Molina, *Designing a super-peer network.*, in Dayal et al. [28], pp. 49–62.

[94] Yi Lin, Bettina Kemme, Marta Patino-Martinez, and Ricardo Jimenez-Peris, *Middleware Based Data Replication Providing Snapshot Isolation*, in SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, New York, NY, USA, 2005, ACM Press, pp. 419–430.

[95] L. Zhenyun Zhuang and Member-Yunhao Liu, *Dynamic Layer Management in Superpeer Architectures*, IEEE Transactions Parallel and Distributed Systems, 16 (2005), pp. 1078–1091.

Edited by: Thomas Ludwig
Received: May 25, 2006
Accepted: October 11, 2006

Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 131–140. http://www.scpe.org
ISSN 1895-1767

THE COMPARISON OF *J2EE* AND *.NET* FOR ENTERPRISE INFORMATION SYSTEMS

JONGWOOK WOO*

Abstract. e-Business and Enterprise Information Systems have held the spotlight since Internet and World-Wide-Web came out to the world. The e-Business applications have been evolved from legacy client-server architecture into n-tier architecture, lately even into Enterprise Information Systems. There are two famous approaches to build the e-Business applications, which are *J2EE* and *.NET*. In this paper, e-Business and n-tier architecture are illustrated. Besides, n-tier architecture for Enterprise Information Systems is introduced, which provide the access to the disparate data sources. In addition, *J2EE* and *.NET* are compared for e-Business applications based on many criteria including the methodologies to implement Enterprise Information Systems.

Key words. Integrated Information Systems, Enterprise Information Systems, e-Business, *J2EE*, *.NET*, n-Tier architecture

1. Introduction. e-Business systems have been popular in the world since Internet and World-Wide-Web came out. IBM defines e-Business as the leveraging of network capabilities and technologies in order to achieve and maintain the huge advantages for customers, suppliers, partners, and employees [9]. e-Business activities can be classified into three categories based on end-users of transactions, normally on the Internet: Intra-business, Business-to-consumer, and Business-to-business. Intra-business activity is to share company information and computing resources among employees on the intranet such as knowledge management. Business-to-consumer, the most common activity, is to provide services to consumers who is out of organizations such as customer resource management, e-Commerce, and web auctions etc. Business-to-business activity is to improve inter-organizational partnerships and relationships such as supply chain integration [8].

The needs of the legacy e-Business systems were simple to maintain functionality and stability on the corporate computing environment. However, the legacy e-Business systems are not sufficient for the current high volume e-Business transactions. People need systems that handle high workloads and changing requirements by applying and adapting applications quickly. Businesses have to improve efficiency by integrating data and applications across the enterprise. Besides, the highest levels of performance and availability must be maintained for the critical businesses. Thus, n-tier architecture for e-Business system has been presented. It partitions systems and software to more flexible blocks that have different roles in order to enable high performance, scalability, and availability to businesses [2]. Section 1 of this paper introduces n-tier architecture in detail.

Either Java—especially, *J2EE* (Java 2 Enterprise Edition)—or ASP (Active Server Pages) has been exclusively used to build server site web systems for e-Business. *J2EE* is the one of editions in Java that is a platform independent and object-oriented language—Java is the product of Sun Microsystems. Thus, *J2EE* fits well to build e-Business systems at both a development and a server site in both Unix (Linux) and Windows operating systems. Besides, the applications of *J2EE* are normally built in Windows operating system and published into servers in any operating systems. Microsoft Corporation provides ASP for e-Business systems. ASP applications are integrated with the codes in *Visual Basic* or *C++*, etc. given by Microsoft Corporation as the products. Therefore, ASP applications are developed and published only in Windows operating systems.

The Unix operating systems have dominated the server market of the large organizations such as banking and entertainment industries because Unix OS have been more stable than Windows so that it was chosen prior to Windows. Thus, e-Business systems of the server market have been mainly developed in *J2EE* instead of in ASP. Microsoft Corporation might want to compete with Unix systems for the e-Business markets so that it introduced the concept of *.NET* on June 2000. And, *.NET* has been presented to the market in 2002. *.NET* is not only platform independent—even it is limited for research—but also programming language independent. *.NET* has been popular for several years in the e-Business world and competed with *J2EE*—probably has dominated the small businesses more than *J2EE*.

In this paper, *.NET* and *J2EE*, the most popular e-Business development approaches, are compared in terms of programming language, platform independency, component model, application server, market proof, openness, and Database connectivity including the connectivity to disparare data sources. Since they are the standards to build e-Business systems nowadays, this paper will be useful for people who want to see the de facto distributed computing environment for e-Business systems and who want to select one of approaches. In the paper, Section 2 introduces the e-Business architectures. Section 3 describes the frameworks of *J2EE* and *.NET* in detail. Section 4 compares *J2EE* and *.NET* in terms of several factors including the approach for

*Computer Information Systems Department, California State University, Los Angeles 90032-8530, Los Angeles, CA, USA (jwoo5@calstatela.edu).

information integration. Section 5 illustrates the summary of the comparison based on the analysis in Section 4. Section 6 is the conclusion and culmination of the comparison for integrated information systems.

2. e-Business Architecture.

2.1. n-tier Architecture. The traditional Client-Server architecture has a mainframe that includes core applications and data. The mainframe is accessed from thick clients that are big applications that contain presentation and business logics. We can call it 2-tier architecture as shown in Figure. 2.1. The 2-tier architecture has many loads between client and server because of their tight interoperations for its presentation logic, business logic, and data access logic. As shown in Figure. 2.1, client has not only the operations of presentation logic but also the part or the full of business and data access logics. This tight interoperation has generated many issues in the current high volume business systems. It is not scalable because it should replace the entire system when its capacity is exceeded. And, it is not flexible because its presentation logic, business logic, and data access logic are tightly coupled. If the developer wants to modify its business logic, he or she should modify the entire logics. Besides, the developer must adapt or modify the business logic when it is integrated with the World-Wide-Web or other applications [2].

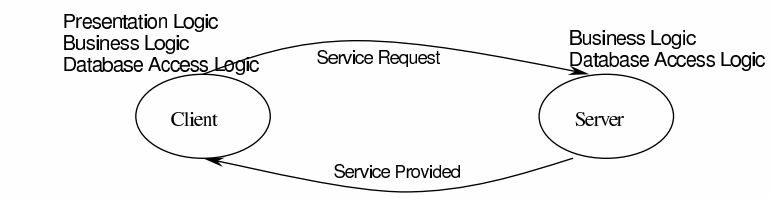


FIG. 2.1. *2-tier Architecture.*

The n-tier architecture has addressed the issues of the 2-tier architecture and become the solution of the current e-Business systems on Internet and World-Wide-Web. It partitions application functionalities into n independent layers, mainly three layers as in Figure. 2.2. Thus, it becomes easier to integrate with the existing business systems. The layer 1 is the presentation logic that is typically hosted on Web server with web browser. The presentation logic is to send the request of client and receive its response from business logic. The response is normally dynamic or static web pages formatted to present to the client. The layer 2 is hosted on mid-tier (middleware) server as business logic. It includes the business functions that are the main of the e-Business applications on the n-tier architecture. It produces the response of the request from the client and provides the response to the client. If the request is related to data access, it will pass the data access request to the back-end database server. The layer 3 is hosted on the back-end database, XML, or other data sources as data access logic. It is to handle the request of data source from the business logic. It has the functions to access data sources such as database, XML, file systems, or EIS (Enterprise Information Systems) etc. Since business logic is separated from presentation logic and database access logic physically, each layer can be scalable and upgradeable independently. And, even if a layer is modified or replaced, the application of other layers do not need to be recreated. Besides, each layer can be implemented with clustered servers for its logic. The clustering enables high-performance computing, availability, and scalability [2]. Therefore, n-tier architecture has been the way to implement the e-Business systems lately.

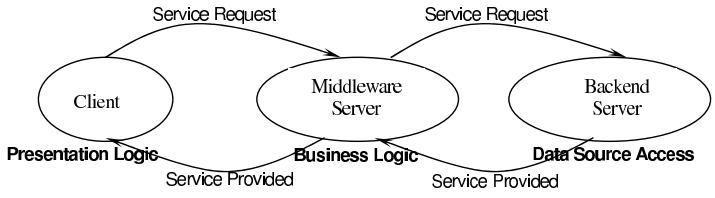


FIG. 2.2. *n-tier Architecture.*

2.2. n-tier Architecture for Enterprise Information Systems. Most of the organizations and companies already have adopted n-tier Architecture for their e-Businesses. Simply, they have the different data sources and their data access methods are different. Thus, each organization's individual solution has made more difficult to share the information among the departments within an organization and among the organizations. However, there has been great need to provide integrated information these days in order to support cooperative works among staffs in agencies and to support their employees and customers. If the different organizations or the different departments of an organization have the integrated information, the integrated information systems will benefit the public.

Integrated Information System can be defined as the system that merges information from the disparate (or heterogeneous) data sources despite differing conceptual, contextual, and typographical representation even in distributed applications. Figure. 2.3 shows the n-tier Architecture with the layer of Information Integration logic that resides on middleware server between Business and Data Source Access logic.

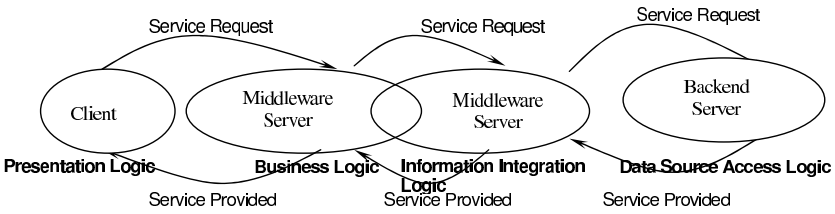


FIG. 2.3. *Information Integration n-tier Architecture.*

3. The *J2EE* and *.NET*. *J2EE* and *.NET* are most popular programming language and framework in order to implement n-tier architecure. This section illustrates the fundamental concepts and frames of *J2EE* and *.NET*.

3.1. *J2EE*. Java platform is composed of APIs (Java Application Programming Interfaces) and JVM (Java Virtual Machine) as shown in Figure. 3.1. Java programs—*J2SE* (Java 2 platform Standard Edition)—are compiled to Java byte codes that are executable on JVM. JVM interprets the byte codes for native operating system of the computer system. In other words, the byte codes are translated to target languages—machine codes—in order to run on the computer system. Thus, Java byte codes can be executable on any operating system if its JVM is installed. That is, Java is a platform independent language that reduces the cost to adapt the existing Java applications to new platform.

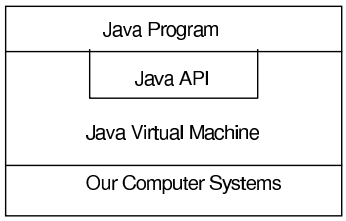


FIG. 3.1. *The Java Platform.*

Java APIs are a set of built-in libraries as byte codes. *J2EE* (Java 2 platform Enterprise Edition) defines the standard APIs for n-tier architecture [10]. *J2EE* has been popular to implement e-Business applications because it is platform independent and has higher performance comparing to the legacy CGI systems with Perl, PHP and C++ etc. Microsoft Corporation's ASP is another competitor to build e-Business applications but it is only for Microsoft Windows system with the exclusive IIS web server that is the product of Microsoft. Thus, *J2EE* has been the popular method to build e-Business systems in the large scaled market such as bank and entertainment.

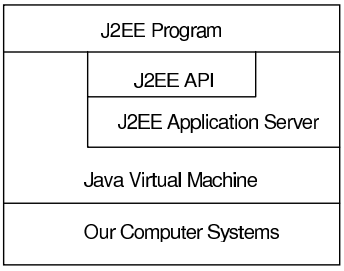


FIG. 3.2. *Application Server for J2EE.*

J2EE is the extended APIs from *J2SE*. It is based on the *J2EE* components for modularization and to simplify the development cycle by providing the details of application behaviors. Thus, it enhances a developer to focus on the business logic without implementing the expensive applications such as transaction, security, database management, and naming service, etc. *J2EE* includes the features of *J2SE* such as platform independency and object-oriented language. Besides, *J2EE* supports APIs for enterprise systems: JDBC for database access, EJB (Enterprise JavaBeans), Java Servlets, JSP (JavaServer Pages), XML, Java Mail, and Java Messaging etc. As are *J2SE* codes, *J2EE* source codes are compiled to Java byte codes and run on JVM that converts Java byte codes to the machine codes. Most operating systems support JVM so that a code runs on an operating system should be executable on other operating systems, which meets the policy of *write-once-run-anywhere* from Sun Microsystems. In order to execute *J2EE* codes, a *J2EE* application server is needed as well as JVM as shown in Figure. 3.2. There are many application servers in the market such as BEA WebLogic, IBM WebSphere, ATG Dynamo, RedHat JBoss, Apache TomCat, and Sun One Application server, etc. And, in order to connect databases, JDBC driver is needed for each database. Normally, each database vendor provides its JDBC driver. Sun Microsystems provides the *J2EE* specification for *J2EE* application servers in oder to maintain *write-once-run-anywhere*.

In Nov 2006, Sun Microsystems announced to be open sourcing all of its Java Source Implementations under GPL (General Public License) version 2 licensing used by GNU/Linux Operating System [17]. The platform implementations include Java SE (JDK), Java ME (Mobile & Imbedded), and Java EE. Before that, there are open Java software projects such as GNU Java [18] and Apache Harmony [19]. Since Sun opens Java implementations, the open Java platform can address the new markets for all Java devices more dramatically.

3.2. *.NET.* Microsoft Corporation is the most famous for Windows operating systems in the personal computer market. Microsoft's ASP (Active Server Page) and languages in Visual Studio have been used to build e-Business applications on Internet and World-Wide-Web. However, the applications mainly depend on Windows operating system so that Microsoft has lost the major portions of server market against Unix server systems. It means that Microsoft may lose the huge market of e-Business system against *J2EE*. Therefore, Microsoft has presented *.NET* solution in June 2000. With *.NET* framework, Microsoft can compete with and hopefully may win over *J2EE* for e-Business applications in large-scaled markets.

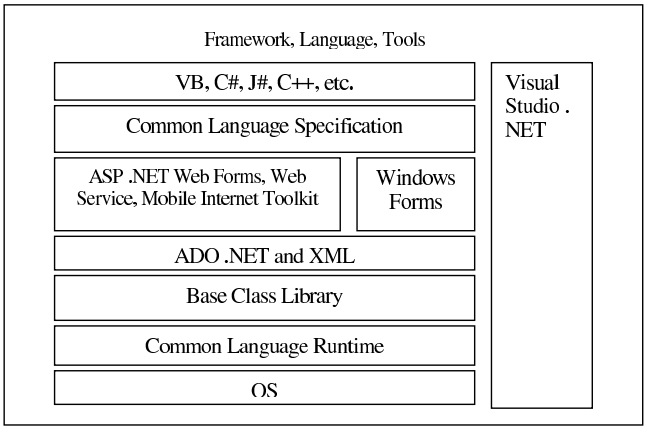


FIG. 3.3. *.NET Framework [Micro].*

Microsoft has focused on its components such as COM (Component Object Model). Component is similar to object and it is the independent unit that provides a function to a client with an interface of operation, property, and event. If a component is implemented, a developer can sell the component and modularize a code with the number of components. Besides, the components modularized can be used in the distributed computing environment. The component model has been extended in *.NET* framework. Microsoft has produced Windows products integrated with *.NET* framework such as Windows XP and 2003 server etc. *.NET* framework supports multi-language environment. At this moment, *.NET* framework supports Visual Basic, C++, C♯, and J♯ languages. Any code written in one of these languages is compiled to a MSIL (Microsoft Intermediate Language) code. Then, CRL (Common Runtime Language) of *.NET* framework interoperates the MSIL codes so that MSIL codes in any language can communicate each other. CRL is to translate the MSIL codes to the machine codes as JVM does in Java. Besides, *.NET* framework may accomplish the platform independency as Java does. Even though it only runs on Microsoft Windows system at this moment, Microsoft provides SSCLI (Shared Source Common Language Implementation) to provides platform independency. Even though it is not clear if the platform independency is the target of *.NET*, Microsoft has studied the possible platform independency to build *.NET* framework executable on FreeBSD and Mac OS X 10.2 operating systems [6].

Mono project is originally an open development initiative sponsored by Novell in order to support *.NET* development to *Unix OS*. *Mono* platform provides the necessary software such as compilers and libraries to develop and run *.NET* client and server on any platform. Mono project provides both programming language and platform independency. The platforms to run *Mono* are *Linux*, *BSD*, *Solaris*, *MacOSX*, *Windows*, and *Unix* etc. Multiple languages can be used with *Mono* platform, which are *C#.NET*, *Java*, *VB.NET*, *ASP.NET*, *Python*, *PHP*, and *JavaScript* etc [20].

4. *J2EE* and *.NET* comparison. This section compares *J2EE* and *.NET* in terms of programming language, platform independecy, component model, database connectivity, market, openness, and application server. Besides, they are compared for information integration that receives the most spotlight in the world these days.

4.1. Programming Language. *J2EE* is the enterprise edition of Java. *J2EE* technology and its component model is the extension of *J2SE*. *J2EE* provides simple enterprise development and deployment with the enterprise APIs such as JDBC, JNDI, Servlet, JSP, RMI, EJB, and JMS. The JDBC—we may regard it as Java Database Connectivity—APIs are used to connect a Java code to a data source, that is, database that provides its JDBC driver. The JNDI (Java Naming and Directory Interface) APIs are to register distributed objects and access one of them. The Servlet APIs are to handle HTTP requests and responses between clients and servers such as application and database servers. The JSP is to create dynamic pages as an extended format of Servlet by integrating presentation logic with *HTML* documents. The RMI (Remote Method Invocation) APIs are to execute the methods of the remote objects on networks. The EJB APIs are to build components that simplify the implementation of server site applications such as session controls with Session Bean, data access and mapping logic with Entity Bean, and asynchronous messaging with Message Bean. EJB also can modulate the applications as component. The JMS (Java Messaging Service) APIs are to provide synchronous communications between objects. Besides, since Java is an object-oriented language, the codes written in *J2EE* are easy to extend and to maintain. Therefore, *J2EE* has been a wll-known solution for e-Business systems more than 10 years.

.NET is the product of Microsoft corporation. It is language independent so that the existing *.NET* programming languages such as *C++.NET*, *VisualBasic.NET*, *ASP.NET*, *C♯.NET*, and *J♯.NET* can interoperate each other on Common Runtime Library (CRL) of *.NET* framework. Microsoft's *VisualStudio.NET* supports these languages with each compiler of the languages that supports CRL [3-5]. Therefore, we can simply extend the existing enterprise systems built in one of these languages by using any of those programming languages. Besides, *.NET* languages are object-oriented languages that have the same benefits as *J2EE*. Thus, *.NET* framework is more extensible—in particular, on Windows—than *J2EE* as it is programming language independent and object-oriented.

4.2. Platform Independency. Java is the platform independent language with JVM provided by Sun Microsystems. Java codes in *J2EE* are compiled to Java byte codes as in J2SE. The Java byte codes can run on any platform such as Unix (Linux) or Windows environment, in which the platform has its JVM installed. JVM converts the byte codes to machine codes of the platform. Almost all platforms have their JVMs to make Java byte codes executable on them. *.NET* framework may have a goal to achieve platform independency. However, it only works on Windows environment at this moment. There is the source code named SSCLI (Shared Source

Common Language Implementation). It is the working implementation to provide a Platform Adaption Layer (PAL) for academics and researchers. SSCLI is under a noncommercial shared-source license and it will run on Microsoft Windows XP, the FreeBSD OS, and Mac OS X 10.2 [5]. If SSCLI is successful, codes on *.NET* framework will be run on FreeBSD OS and Mac OS X 10.2 as well as Windows OS. Therefore, *.NET* framework may achieve the platform independency even though it does not run on most UNIX OSs.

4.3. Component Model. Component in software can be defined as an independent unit to provide an operation with the interfaces such as operation, property, and event. If a component model is built for a certain function, the component can be salable and integrated with other products. In addition, many components can be developed in modules and run on distributed computing environment. Each component should be registered in a naming server for distributed computing environment. *J2EE* provides component model named EJB. It runs on an EJB application server. The basic idea is to use the built-in applications of EJB application server such as expensive security, transaction, and database integration functions. If a developer purchase an EJB application server, the developer can only focuses on implementing his or her business logic with EJB instead of spending on building those expensive functions. It will save time and cost to develop a product of the organization. EJB application server normally includes JNDI (Java Naming and Directory Interface) server. EJBs are registered to the JNDI server so that an EJB objects registered can be found in the JNDI server whenever they are called in a code.

Microsoft Corporation has developed a component model such as COM (Component Object Model). It is a Microsoft specification for component interoperability. It has been extended to DCOM (Distributed Component Object Model) in 1990s. About 1997, COM+ plan was announced by Microsoft, which is an extension of COM. COM+ builds on COM's integrated services and features. It also makes it easier for developers to create and use software components in any language [4]. Microsoft Corporation has applied the existing component concept to *.NET* framework. *.NET* framework is an integral Windows component for building and running the software applications and Web services. However, *.NET* components are only registered in the Windows registry. Thus, it cannot be seperated from technology and support of Microsoft products.

4.4. Database connection. JDBC technology is an API to access virtually any tabular data source from Java codes. If a data source such as database is linked to JDBC driver, Java codes can access the database. Normally, each database vendor provides its JDBC driver as the database product. When a Java code is built for database access application, it needs to refer to classes of JDBC API of the JDBC driver that is accessible from the code. In addition to JDBC, an entity bean of EJB has database connection interfaces. A developer can easily implement an entity bean that connects a database without building JDBC connection logic. Thus, the developer can only focus on implementing business logic so that it will save the cost of his or her product.

OLE (Object Linking and Embedding) DB is a standard interface of Microsoft with which a developer can refer to any data source. It is built in as a part of the *.NET* framework. *ADO (ActiveX Data Object).NET* is on top of OLE DB as another layer. *ADO.NET* is a database object model that is composed of many standard classes to refer to data from any database. The integrated developing environment (IDE) such as Visual Studio *.NET* normally supports the OLE DB database provider of each database. Since the provider uses certain *ADO.NET* classes to connect a database, the developer can easily establish the database connection application in *.NET*.

4.5. Application Server. Java codes run on JVM. However, *J2EE* codes are not executable on JVM alone. It needs an application server that makes the codes executable. *J2EE* codes on an application server are mainly for web applications—you may regard them as e-Business applications. The popular application servers in the market now are BEA WebLogic, IBM WebSphere, ATG Dynamo, and Oracle application server etc. Besides, there are free application servers such as Apache TomCat and RedHat JBoss. Since there are many vendors that implement application servers, some *J2EE* codes runnable on an application server are not executable on other servers. It violates the motive of Java language. Thus, Sun Microsystems provides J2EE specification to keep the *write-once-run-anywhere* motto. Thus, any *J2EE* application will run on the application server if the vendor follows the direction of the specification when implementing the application server. The server that meets the specification is called the Sun certified J2EE application server

To run *.NET* applications on the legacy Windows OS, *.NET* framework is needed that can be downloaded from the Microsoft Corporation web site [3]. Otherwise, we can purchase and install Windows server 2003 to run *.NET* applications. For web applications, normally, *ASP.NET* is used for a client site—web browser—to access the dynamic functions built in other *.NET* languages at a server site. *ASP.NET* only runs on Microsoft IIS web server. It means that Microsoft Corporation exclusively dominates the *ASP.NET* market with IIS server.

The IIS server handles both static and dynamic web pages so that we can call it application server. Since there are some issues in IIS server, for example, security and open source needs, Microsoft provides Cassini that is source-available Web server platform and written entirely in C♯. Thus, a developer can modify the internal functions of Cassini for his or her need and implement the *.NET* compatible application server. Cassini supports *ASP.NET* and other basic functions such as directory browsing on HTTP 1.1. You can demonstrate Cassini on the *.NET* Framework [1].

4.6. Openness. There have been many approaches for Java Open Source. Sun has had an *OpenSolaris* project to develop *SolarisOS* by releasing most of the Solaris source code under the Common Development and Distribution License (*CDDL*) [21]. However, many open source communities critisize that *OpenSolaris* project does not have the true open source community processes. *Sun* provided Java open source for *OpenSolaris* project. And, *GNUJava* project has supported *Java* language with *Java* Compiler and *VM* etc [18]. *Apache* also launched *Harmony* project to support platform independent *JavaSE*5*JDK* under *Apache* license [19]. On Nov 2006, *Sun* announced to open Java source for *SE*, *ME*, and *EE* under *GNUGPL* license [17]. And, many open source communities believe that it can be useful for *Java* world amazingly.

Mono project is to provide open source software for *.NET* on *Unix* platform sponsored by *Novell*. It provides *.NET* compiler and libraries etc. Besides, it is actually both platform and language independent platform even though it needs more studies to be compatible to the platforms and langauges [20].

4.7. Information Integration. There has been great need to integrate and share the information among disparate data sources within the same or among many different organizations. We can define the disparate data sources as databases from differen vendors, file systems, and XML etc. The integrated information system has many demands to satisfy security and reliable requirements, as well as data privacy, quality, and ownership. In addition, it has the complexity of integrating disparate data. Enterprise Information Integration (EII) is one technical approach that addresses integration complexity. EII is the process of using data abstraction to tackle the data access challenges and complexity associated with the disparate data sources in e-Business.

In Java, there have been several approaches to resolve the problem to integrate the disparate data sources for n-tier architecture such as JDO, JAXB, EMF and SDO. With these approaches, Java developers can only focus on the business logic without wasting resources for data management applications. JDO stands for Java Data Objects standardized by JCP (Java Community Process). It provides an API to access data in data sources such as database and file systems etc. EMF (Eclipse Modeling Framework) generates a unifying metamodel based on a data model defined using Java interfaces, XML schema, and UML class diagrams. JAXB stands for Java API for XML Data Binding. It is release by JCP and used to generate Java objects in memory corresponding to XML data [11, 12].

SDO stands for Service Data Objects. It was originally developed as a joint collaboration between BEA and IBM and is now being developed by BEA, IBM, Oracle, SAP, Siebel, Sybase and XCalia etc. SDO is abstracts data in order to utilize multiple disparate data sources, which includes databases, entity EJB components, XML, Web Services, Java Connector Architecture, and JSP pages [11-13]. SDO provides SDO API as JDO. However, SDO is more general than JDO so that SDO can be used for between any tiers on n-tier architecture while JDO is for data access tier only. JDO can be even considered as a data source for SDO. Both SDO and EMF present data representation. SDO is created by EMF code generation and is a facade over EMF as part of EMF project. JAXB only focuses on Java-to-XML binding while SDO takes care of any data source. Thus, SDO has been received many lights as it provides only a single and simple interface to a variety of disparate data. And, it can be also applicable to SOA (Service Oriented Architecture) such as Web Services [12].

Microsoft introduced ActiveX Data Objects (ADO) on the release of VB 5. ADO was built to provide access to disparate data sources on distributed computing, that is, n-tier architecture. *ADO.NET* is the expansion of ADO by using XML. There are proprietaries as *ADO.NET* providers such as Simba Technologies, DataDirect Technologies, and OpenLink Software that present drivers and bridges to other data sources [15]. *ADO.NET* is the product of Microsoft. Java SDO API is JSR (Java Specification Request) 235 that is the request to be Java standard API.

5. Summary. Up to Section 4, we see the approach of *J2EE* and *.NET* to build e-Business applications. It is described how *J2EE* API and *.NET* products are used on n-tier architecture in Figure. 4.1. To build the presentation logic of e-Business application, JSP and servlet of *J2EE* API and *ASP.NET* of *.NET* framework can be used. For the business logic, EJB—especially Session Bean—and standard Java classes for *J2EE* and *C++.NET*, *C♯.NET*, and *VB.NET* etc. for *.NET* can be applicable to build the business functions. And, there is information integration logic between business and data access logics. In *J2EE*, JDO, EMF, and JAXB can

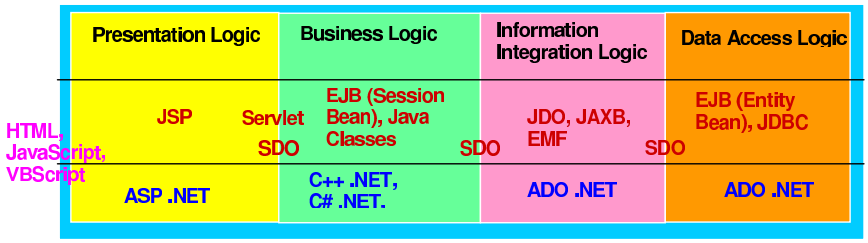


FIG. 4.1. *J2EE and .NET on Enterprise n-tier Architecture.*

be used as *ADO.NET* in *.NET*. For information integration of *J2EE*, SDO can resides on between any tiers. Finally, the developer can implement the database access logic with EJB—especially Entity Bean—and JDBC classes for *J2EE* and *ADO.NET* for *.NET*.

Figure. 5.1 summarizes the comparison between *J2EE* and *.NET* for the criteria of e-Business applications as analyzed in Section 4. The criteria are how to handle dynamic web contents, how to access database, platform independency, possible programming languages to build the applications, to see if there is a component model and if it is proven in the market, how much the cost to use them, how to integrate heterogeneous data sources, how is openness, and performance. In the market, *J2EE* has been proven for more than 10 years and *.NET* has been only for several years. However, *.NET* has been used by many companies and organizations so that it is already proven too. In terms of the cost to build and execute applications, *J2EE* can be less expensive since it is free and there are free application servers to make the *J2EE* codes run, for example, JBoss. But, in *.NET*, people need to buy a *VisualStudio.NET* IDE (Integrated Development Environment) and IIS Web server in order to build solid applications. As the alternative and cheap methodologies to develope *ASP.NET* applications, Cassini as application server and WebMatrix [16] as IDE are not good enough to implement the solid products.

J2EE is platform independent. *.NET* is the Microsoft language independent but not platform independent. However, there is Mono project to make *.NET* code executable on *Unix* platform. *J2EE* community has worked on integrated data source as *ADO.NET* so that *SDO* has come out to the world. In order to get the benefit of open source as *Linux* has done, *Sun* provided open source for *Java*.

For the performance, the Middleware Company presents the report insisting on that *.NET* has better performance on the Pet Store benchmark tuned for *.NET* than *J2EE* on the benchmark [7]. However, since the benchmark is optimized for *.NET* and executed on Windows OS while *J2EE* runs on JVM of Windows OS, the result should be a matter of course. For the better fairness, the performances of *.NET* and *J2EE* applications should be measured with the well optimized benchmark for both *.NET* and *J2EE* on the different platform such as Unix, which is almost impossible at this moment.

	J2EE	.NET
Dynamic Web Content	JSP	ASP.NET
DB Access	JDBC	ADO.NET
Platform Independency	Yes	Yes (Mono project)
Languages	Java	C++, C#, Visual Basic, J#
Component Model	Yes (EJB)	Yes
Market Proven	Yes	yes
Cost of product	Some freeware	No freeware
Integrate Disparate Data Sources	JDO, SDO	ADO.NET
Openness	Java Open Source	Mono
Performance	?	?

FIG. 5.1. *Summary: J2EE and .NET.*

6. Conclusion. As e-Business applications have been implemented, the importance of the information integration among the collaborative groups has been grown. In this paper, n-tier architecture of e-Business is described. Then, Enterprise Information Systems architecture is introduced. The most popular approaches are illustrated to build the applications on n-tier architecture: *J2EE* and *.NET*. *J2EE* is the specification provided by Sun Microsystems. *J2EE* is more flexible because *J2EE* API is free and anyone can implement *J2EE*

application server that meets the specification given by Sun. *.NET* of Microsoft Corporation is the product. Thus, it is only dedicated to Microsoft products. If considering the applications on Windows only, *.NET* is more flexible than *J2EE* because it is programming language independent. *J2EE* and *.NET* are compared in terms of dynamic web content, database connectivity, platform and language independency, component model, market, cost, openness, and heterogeneous data source integration methodologies. However, it is not easy to compare the performance of *J2EE* and *.NET* because *.NET* is not executable on the other platforms yet. The paper should be the useful reference to establish e-Business and Enterprise Information Systems for both profit and non-profit organizations which do not have the technical and architectural ideas for the systems.

Note. Figure. 6.1 is the table for acronyms used in this paper.

Acronym	
ADO	ActiveX Data Object
APIs	Application Programming Interfaces
ASP	Active Server Pages
CDDL	Common Development and Distribution License
COM	Component Object Model
CRL	Common Runtime Language
DCOM	Distributed Component Object Model
EII	Enterprise Information Integration
EJB	Enterprise JavaBeans
EMF	Eclipse Modeling Framework
GPL	General Public License
IDE	Integrated Development Environment
JCP	Java Community Process
JMS	Java Messaging Service
JNDI	Java Naming and Directory Interface
JSP	JavaServer Pages
JSR	Java Specification Request
J2EE	Java 2 platform Enterprise Edition
J2SE	Java 2 platform Standard Edition
JVM	Java Virtual Machine
MSIL	Microsoft Intermediate Language
OLE	Object Linking and Embedding
PAL	Platform Adaption Layer
RMI	Remote Method Invocation
SOA	Service Oriented Architecture
SSCLI	Shared Source Common Language Implementation

FIG. 6.1. *Table for Acronyms*

REFERENCES

[1] *Cassini Sample Web Server*, http://www.asp.net/Default.aspx?tabindex=7&tabid=41 Microsoft Corporation, 2003.

[2] *Building a Better e-Business Infrastructure: n-tier Architecture Improves Scalability, Availability and Ease of Integration*, http://www.intel.com/eBusiness/pdf/busstrat/industry/wp012302.pdf Intel e-Business Center White Paper, 2001.

[3] *Overview of .NET Framework*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpovrintroductiontonetframeworksdk.asp *.NET* Framework Developer's Guide, Microsoft Corporation, 2003.

[4] *COM+*, http://www.microsoft.com/com/tech/COMPlus.asp Microsoft Corporation, 2002.

[5] *Got Dot NET (.NET Framework Website)*, http://www.gotdotnet.com/team/lang/ Microsoft Corporation, 2003.

[6] *C♯/JScript/CLI Implementations Shared Source Licensing Program*, http://www.microsoft.com/resources/sharedsource/Licensing/CSharp_JScript_CLI.mspx Microsoft Corporation, June 2003.

[7] *The Petstore Revisited: J2EE vs .NET Application Server Performance Benchmark*, http://www.middleware-company.com/j2eedotnetbench Middleware Company, Oct. 2002.

[8] INDRAN NAICK, LUCA AMATO, JASON K O'BRIEN, JIM NICOLSON, AND TSUTOMU OYA, *Design Considerations: From Client/Server Applications to e-business Applications*, http://www.redbooks.ibm.com/redbooks/SG245503.html Dec 1999.

[9] BRIAN R. SMITH, CHARLES ACKEIFI, THOMAS G. BRADFORD, PRABHAKAR GOPALAN, JENNIFER MAYNARD, AND ABDULAMIR MRYHIJ, *IBM e-business Technology, Solution, and Design Overview*, http://www.redbooks.ibm.com/redbooks/SG246248.html August 2003.

[10] *Java 2 Platform, Enterprise Edition (J2EE)*, http://java.sun.com/j2ee/ Sun Microsystems, Inc, 2003.

[11] C. M. SARACCO, JACQUES LABRIE, AND STEPHEN BRODSKY, *Using Service Data Objects with Enterprise Information Integration Technology*, IBM Developer Works, July 2004.

[12] BERTRAND PORTIER AND FRANK BUDINSKY, *Introduction to Service Data Objects*, IBM Developer Works, Sept 2004.

[13] *Service Data Objects*, Dev2Dev at BEA World, Jan 2006.

[14] Jongwook Woo, *The Comparison of J2EE and .NET for e-Business*, The 2005 International Conference on e-Business, Enterprise Information Systems, e-Government, and Outsourcing, EEE 2005, Las Vegas, Nevada, June 20-23, 2005.
[15] Kirk A. Evans, Ashwin Kamannna, and Joel Mueller, *XML and ASP .NET*, published by New Riders, First Edition, April 2002.
[16] *WebMatrix*, `http://www.asp.net/webmatrix/` 2006 Microsoft Corporation.
[17] *OpenJDK*, `http://www.sun.com/software/opensource/java` 2007 Sun Microsystems, Inc.
[18] *GNU Java*, `http://www.gnu.org/software/java/` GNU Free Software Foundations.
[19] *Harmony*, `http://harmony.apache.org/` 2006 The Apache Software Foundation.
[20] *Mono*, `http://www.mono-project.com` Mono Project.
[21] *OpenSolaris*, `http://opensolaris.org/os/` 2006 Sun Microsystems, Inc.

Edited by: Domenico Talia
Received: May 25, 2006
Accepted: January 20, 2007

Scalable Computing: Practice and Experience
Volume 8, Number 1, pp. 141–142. http://www.scpe.org
ISSN 1895-1767

BOOK REWIES

EDITED BY SHAHRAM RAHIMI

Algorithms Sequential & Parallel: A Unified Approach
Russ Miller and Laurence Boxer
Prentice Hall, Upper Saddle River,
New Jersey 07458
330 pages, $40.77
ISBN: 0-13-086373-4

This book covers some fundamental computer science algorithms and discusses the implementation issues for both sequential and parallel models. The book is aimed to be used for a senior-level undergraduate or an introductory-level graduate course in computer science in order to help the students to understand the application and analysis of algorithmic paradigms to both sequential and parallel models of computing. It can be viewed as a complementary course to "Data Structure & Algorithms" or "Introduction to Algorithms", which are offered in undergraduate level.

The book is organized in three parts. The first part is some background materials for the course and is comprised of three chapters. The first chapter introduces the concept of asymptotic analysis which is used frequently through out the book for analysis of algorithms. Chapter 2 briefly reviews fundamentals of induction and recursion and Chapter 3 introduces the Master method as a powerful system for evaluating recurrence equations that are used for analysis of many algorithms in the book.

The second part of the book gives an introduction to models of computation. Chapter 4 motivates the natural use of parallel models of computing by presenting the sorting networks and analyzing the Bitonic-sort algorithm using sequential and parallel models. Chapter 5 introduces basic models of computation. It starts out with RAM as a traditional sequential model of computation and then introduces PRAM as a shared memory model of parallel computing and gives several examples of algorithms that utilize this model. The chapter ends up by introducing the distributed memory machines and some interconnection networks including linear array, ring, mesh, tree, pyramid and hypercube. In addition, this chapter introduces terminology such as granularity, cost, speed up and efficiency.

The third part of the book is the main part of the book and consists of chapter 6 to 13. This part covers a variety of algorithms in several application domains and discusses the implementation and analysis of these algorithms using different computational models introduced in chapter 5. Chapter 6 considers the problem of matrix multiplication and Gaussian Elimination on RAM, PRAM and mesh models. Chapter 7 introduces the parallel prefix operation and discusses the implementation and analysis of its sample applications for a number of computation models such as RAM, PRAM, mesh and hypercube. Chapter 8 covers the pointer jumping techniques and shows how some list based algorithms can be efficiently implemented in parallel. Chapter 9 presents the divide-and-conquer and shows the application of divide-and-conquer to problems involving data movement including sort, concurrent read and write and so forth. These algorithms and their analysis are presented for a number of sequential and parallel models. Chapters 10, 11 focus on the implementation and analysis of basic algorithms in the area of computational geometry and image processing. Chapter 12 dwells on implementation of some fundamental graph algorithms such as graph traversal, labeling, minimum cost and shortest path on RAM, PRAM and mesh models. Finally chapter 13, which is an optional chapter, concerns with some basic numerical problems such as evaluating a polynomial, approximation by Taylor series, trapezoidal integration and so forth. The focus of chapter is on sequential algorithms for polynomial evaluation and approximation of definite integrals.

The book is well-organized and covers a wide variety of fundamental computer science algorithms in several application domains. It helps the students to understand the implementation of algorithms in different parallel models and compare them to their sequential counterparts. However, there are some drawbacks that could be noted. Although the analysis of parallel algorithms deals with communication models and performance measurement, the book does not cover these areas in sufficient details. The number of parallel models and interconnection networks mentioned in the book are limited and the book does not include the methodology of designing parallel algorithms. Conclusively, while the book is useful to give an introduction to parallel models

and algorithms and compare them to sequential ones, students would still need to utilize other sources on parallel computing to obtain more knowledge on this domain.

Elham S. Khorasani,
Department of Computer Science
Southern Illinois University
Carbondale, IL 62901, USA

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:

- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:

- programming environments,
- debugging tools,
- software libraries.

Performance:

- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:

- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:

- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in LaTeX 2_ε using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.