South African
Computer
Journal
Number 12
November 1994

Suid-Afrikaanse
Rekenaar-
tydskrif
Nommer 12
November 1994

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

The South African Computer Journal
An official publication of the Computer Society of South Africa and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif
'n Amptelike publikasie van die Rekenaarverenging van Suid-Afrika en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

Editor
Professor Derrick G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083
Email: dkourie@dos-lan.cs.up.ac.za

Subeditor: Information Systems
Prof Lucas Introna
Department of Informatics
University of Pretoria
Hatfield 0083
Email: lintrona@econ.up.ac.za

Production Editor
Dr Riël Smit
Mosaic Software (Pty) Ltd
P.O.Box 23906
Claremont 7735
Email: gds@mosaic.co.za

Subscriptions

	Annual	Single copy
Southern Africa:	R50,00	R25,00
Elsewhere:	$30,00	$15,00

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Phone: +27 (11) 315-1319 Fax: +27 (11) 315-2276

CALL FOR CONTRIBUTIONS

South African Computer Journal (SACJ): Special Issue

Information Technology and Development

South Africa, as a developing country, needs to find ways to harness it's enormous potential in a rapid and sustainable way. It is argued by some that the developments in information technology are increasingly creating opportunities for socio-economic development to be enabled by the diffusion of IT. If this is true, then these opportunities should surely be explored. Consequently, the editorial board of SACJ has decided to devote a special issue to this theme.

Some of the questions that the special issue might address are:

- To what degree can information technology enable (or accelerate) socio-economic development?
- How can the diffusion of information technology be achieved in order to maximize its contribution towards socio-economic development?
- What are the conditions required for such technology enabled development?
- Are there high leverage areas where quick returns can be achieved?
- Should the state play an active role and in what way?
- What are the experiences of other developing countries in information technology enabled socio-economic development?
- What are the moral and ethical issues involved in information technology enabled socio-economics development?

We invite *all* researchers in the field of information technology and other disciplines to contribute to this special issue. All contributions will be reviewed by three independent reviewers. Contributions should be sent in four copies to:

Prof Lucas D. Introna
South African Computer Journal (SACJ)
Special Issue: IT and Development
Dept. of Informatics
University of Pretoria
Pretoria, 0002
SOUTH AFRICA

Contributions should be received by June 5, 1995. Contributions sent by facsimile or e-mail will not be accepted. For any information regarding the special issue contact Prof Lucas Introna at +2712 4203376 (office), +2712 434501 (fax) or Internet: lintrona@econ.up.ac.za.

Important Dates

Contribution deadline:	June 5, 1995
Notification to author(s):	September 1, 1995
Final versions due:	November 1, 1995

LEXICA

an integrated environment for machine translation

LEXICA's capabilities include :

- **Translator's Workbench**
- **Powerful word-based Post Editor**
- **Automatic translator**
- **Integrated on-line dictionaries**
- **Support for most Windows-based wordprocessors**

LEXICA is a language independent platform

New language pairs can be developed on request

LEXICA provides unique support for languages spoken on the African continent, including :

- **Swahili**
- **Tswana**
- **Zulu**
- **Afrikaans**
- **Portuguese**

as well as :

- **German and French**

Eenheid vir Programmatuur-Ingenieurswese
Unit for Software Engineering

in association with
Department of Computer Science
University of Pretoria, Pretoria 0002 South Africa

Telephone: (012) 420-2504
Fax: 43-6454
E-mail: lexica@cs.up.ac.za

Guest Contribution

About the author

Fred Lochovsky, is a long-standing and eminent member of SACJ's editorial board. He is a professor in the Department of Computer Science at the University of Science and Technology, Hong Kong. He received his B.A.Sc., M.Sc. and Ph.D. from the University of Toronto. Before joining HKUST in January 1991, he was a professor of Computer Science and Management at the University of Toronto. He was also, concurrently, associate director of the Computer Systems Research Institute (1988–1990). In 1983 he was a visiting scientist at IBM Research Laboratory (San Jose, California, USA). He was associate dean of Engineering at UST from Sept. 1992 to Dec. 1993. Prof Lochovsky serves or has served on the editorial boards of Information Systems (Pergamon Press), ACM Transactions on Information Systems, IEEE Database Engineering, The South African Computer Journal, and as Editor-in-Chief of IEEE Office Knowledge Enginering. His current research interests are in information systems design, database design, data and knowledge base integration, organizational support systems, and human-computer interaction.

Organizational Computing Technologies for Supporting Organizational Activities

Frederick H. Lochovsky
University of Science and Technology, Hong Kong

Abstract

Technologies for supporting organizational activities are becoming increasingly important as a means for organizations to achieve their business goals. Such technologies allow organizations to model, execute, and control work processes that typically involve multiple people. While much progress has been made in introducing standalone technologies into organizations, we are still a long way from providing integrated support for organizational activities. Many approaches have been proposed for supporting organizational activities ranging from simple form-based systems to intelligent, knowledge-based information systems. We review the major approaches proposed to date and outline some research directions which may further improve support for organizational activities.

1 Introduction

Organizational computing is concerned with the application of computer technology in organizations. As such, it encompasses support for organizational activities which, broadly defined, are any functions that help to achieve some business goal of the organization. These activities can range from those of a mostly routine and repetitive nature, such as order processing, to those of a more complex and one-time nature, such as making a decision on a merger. Technologies for supporting organizational activities are becoming increasingly important as a means for organizations to improve their productivity and competitiveness. By automating and re-engineering business functions, such systems offer organizations the ability to model, execute, and control organizational activities that typically involve multiple people in collaboration.

Support for organizational activities has been the subject of much research, particularly in the last ten years or so and is known by a variety of terms such as office automation, CSCW (computer-supported cooperative work), groupware, computer-supported coordination technology, among others [38]. In line with its many names, the research has covered a wide spectrum from complete automation to providing intelligent support. This research is driven both by technological advances and by a perception that organizations are not as productive as they could be and that computer-based support can help in increasing productivity [28, 40, 39]. Whether increased productivity can be achieved or not, there is no doubt that computers have affected and will continue to affect how activities in organizations are carried out.

In this paper we will survey some major approaches that have been proposed for supporting organizational activities. Our focus is on understanding the underlying technologies and how they support organizational activities and not on the human processes involved in such activities. In order to guide our understanding of the various approaches and how they are related to each other, we first outline a modelling framework within which various research ef-

forts and approaches can be described. Then we discuss in more detail one of the components of the framework (activity modelling) and some of the major technologies that have been proposed to support this component. Next we present an approach for integrating some of the technologies proposed for supporting organizational activities. Finally, we conclude the paper by outlining some areas for further research.

2 An Organizational Computing Modelling Framework

Research in organizational computing has grown rapidly over the last few years with researchers from many related, and relevant, disciplines taking part. As a consequence, there are many different perspectives on what constitutes organizational computing. In this section we identify and discuss four of these perspectives.

Organization modelling

Organization modelling is concerned with describing the *structural* aspects of an organization. It describes the different parts of an organization, how these parts are related to each other and their properties. Important entities that are used in such a description include:

- **(organization) units**
  These are subsets of the organization which can be either permanent (e.g., marketing division, sales department) or temporary (e.g., projects, committees, etc.). Organizational units can be composed of other organizational units in a variety of ways (e.g., hierarchy, matrix, etc.).
- **positions**
  These are the staffing units within an organization. They are "abstract employees" in that positions are usually "filled" by people, but they can exist independent of any people to fill them [42]. (Positions could also be filled by appropriate software entities—see below.) Positions have responsibility and authority associated with them. For example, they are the "agents" that carry out activities.

Resource modelling

Resource modelling describes the *working materials*, *work objects* and *tools* that are available to an organization to achieve its goals and mission [42]. Resource entities include:

- **physical facilities**
  For organizational computing, the physical facilities of interest are the hardware and software resources. Modelling such resources can be useful in determining if the required resources exist for accomplishing some organizational goal or for matching appropriate physical resources with organizational activities.
- **agents**
  The resources within an organization that are capable of carrying out "independent" actions (i.e., people, some forms of software) are modelled as agents. (Software agents have also been known by other terms such as "actors" [1, 13, 5].) Agents can be a combination of human and software entities. Agents have certain capabilities associated with them and either hold (human agents) or are associated with (software agents) positions in an organization.

Data/knowledge modelling

Data/knowledge modelling describes the *data* and *knowledge* that are created by and used within an organization. From an organizational computing perspective, the data/knowledge entities include:

- **messages**
  In organizations, data appear as messages of various kinds. These messages can be characterized as structured (records, forms, etc.) or unstructured (letters, reports, etc.). They may be created and/or used by various resources available to the organization.
- **regulations**
  In organizations, knowledge often appears as regulations of various kinds. These govern the behaviour of the organization and determine some of the constraints under which it operates [14]. Regulations can be in the form of internal policies or external laws that must be adhered to. Regulations can usually be expressed as constraints (rules) on data and activities.

Activity modelling

Activity modelling captures the *dynamic aspects* of organizations in that activities usually effect changes on the other modelling entities (e.g., messages, units, etc.). While there is certainly a great deal of activity that happens in an organization that is based on individual effort, when we consider the activities of an organization as whole, we soon realize that these activities are mainly cooperative or collaborative in nature. That is, many people are involved, usually in a coordinated manner, in accomplishing a given activity.

One way to model a cooperative activity is to decompose it in terms of the different agents that are involved in the activity and the work that they contribute to it. For example, a purchasing activity could involve the actual end-user of the item to be purchased (who initiates the activity), a clerk who prepares the purchase order, the user's manager who may need to authorize the purchase, a clerk in the finance office who checks availability of funds, a purchasing agent who places the order with an outside supplier, etc. Each of these agents play a certain role, have certain responsibilities, and perform certain well-defined subactivities (perhaps more than one) in the overall purchasing activity. This view of organizational activities leads us to model them by a distributed environment made up of various agents through which messages flow and are processed.

In this view of organizational activities there are two aspects of an activity that need to be modelled:

- **tasks**
  Tasks define the processing (operations) done by agents involved in an activity. They can be performed by a single agent (e.g., filling out a form, sending a letter,

etc.) or by a collection of agents exhibiting *dependency* relationships such that a "dependent" task cannot be worked on until the "precedent" task has been completed. Tasks provide a context through which the interactions between data/knowledge and resources are defined. Tasks can be nested to multiple levels (sub-tasks), and may execute sequentially or concurrently [3, 29].

- **workflows**
  Workflows specify the flow among the agents of the messages being processed. Workflows represent a complementary aspect of tasks in that they define the order of task invocation, task synchronization, and data-flow. In a distributed environment, workflows also define the paths or routes that messages take as they move among the various agents of the organization [9].

These two aspects of activity modelling, while not independent of each other, can be considered independently for purposes of modelling and specification[1].

All four perspectives of organizational computing are important, and necessary, for supporting organizational activities [17, 20, 24, 26, 34, 35, 43, 50]. Nevertheless, by far the most research effort has been devoted to the last perspective, at least in the Computer Science community. Accordingly, in the following two sections we elaborate further on the two aspects of this perspective.

3 Tasks

As discussed in the previous section, organizational activities are composed of a number of tasks performed by one or more agents. Tasks in organizations have often been characterized as consisting of what has been called Type I and Type II tasks [41]. Type I tasks (also known as structured or procedural tasks) are of a routine nature for which a prescribed step-by-step solution is known. Theoretically, to support Type I tasks, we only need to encode the step-by-step solution in a computer program. Participation by agents in a Type I task, if necessary, is generally limited to supplying well-defined inputs. Type II tasks (also known as unstructured or problem-solving tasks), on the other hand, are novel and non-repetitive, and must be solved with creativity, initiative and originality [10, 41]. To support Type II tasks, precise goals and strategies to be used must be available. In addition, participation by human agents in Type II tasks is essential because it is not possible to capture all the required goals and strategies at once or, perhaps to ever capture all of them due to the open-ended nature of organizations [6, 13].

This distinction between Type I and Type II tasks is, obviously, an ideal characterization. Most tasks fall on a spectrum between Type I and Type II tasks, with the Type II end of the spectrum being open-ended. That is, an organizational activity may consist of either Type I, Type II or a mixture of both Type I and Type II tasks. In addition, tasks may be further decomposed into a number of sub-tasks. This leads to the view of an organizational activity as being composed of a hierarchy of related tasks (and sub-tasks). While the entire activity may be some combination of Type I and Type II tasks, each atomic (sub-)task is either Type I (structured) or Type II (unstructured). For example, in the purchasing activity outlined earlier, the sub-task of checking whether sufficient funds are available is a Type I task. The sub-task of deciding on which item to purchase (i.e., evaluating different possible alternatives) is, on the other hand, generally a Type II task. Overall, the purchasing activity is a combination of Type I and Type II tasks.

This Type I, Type II characterization of tasks is useful because it allows them to be placed on a task spectrum and thus to determine to what degree and perhaps how the related organizational activities can be supported by computerization. In general, organizations try to force tasks towards the Type I end of the task spectrum. In this way, their outcome is more predictable and it is possible to perform the task more efficiently (since Type I tasks eliminate, to a large extent, unnecessary searches for the appropriate actions to perform).

It is possible that a Type I task enters a situation where there is no *a priori* set of procedures to handle the task. In this case, the task has to be handled as a Type II task. On the other hand, people learn from experience and as they know more about some particular Type II task, it may evolve to be very routine and structured. In this case, the task should be handled as a Type I task. Thus, in general, support for organizational activities should allow the associated tasks to move on the task spectrum between the Type I and Type II end as required.

In the rest of this section, we outline some of the approaches that have been proposed for dealing with the processing of tasks. We begin with approaches for handling Type I tasks and move toward the Type II end of the spectrum.

Tasks as form-based processing

In form-based task processing, the messages that are processed by each agent are highly structured and represented as forms. Forms consist of a set of attributes and attribute values and can usually be stored and managed by a database system [11, 24, 30, 46, 52]. While individual forms can circulate among agents, they are usually collected into folders (possibly along with other types of messages) for circulation (Figure 1) [8, 20].

The forms themselves may be *passive* or *active*. A passive form is one that is just deposited with an agent and waits for the agent to use it in some task. Active forms, on the other hand, cause some task to be triggered *automatically* on their arrival at an agent [8, 16].

The tasks that process forms can be specified using a programming language. However, this makes the task specification unavailable to most people. One approach that has been proposed to address this problem is to use the forms themselves to specify tasks [16]. In this way, the same interface through which an agent enters and retrieves

[1] In fact, technologies used to support the one can also be used to support the other.

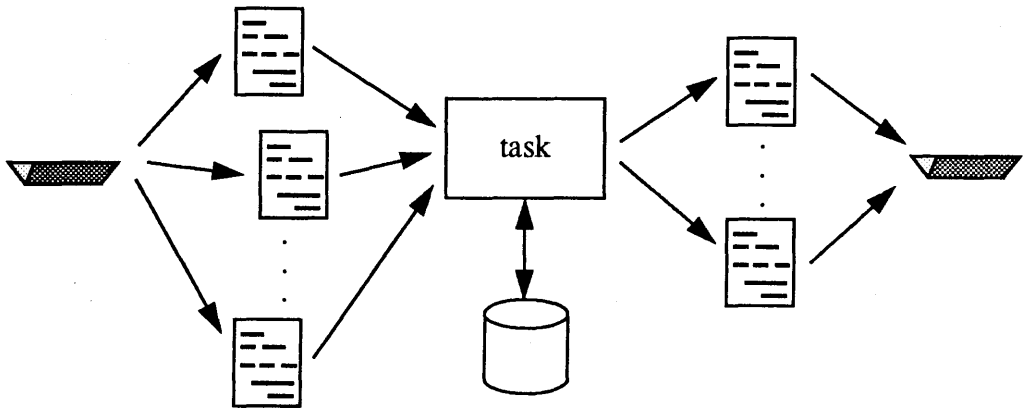

Figure 1. Form-based task processing.

information (a form) is used to specify the tasks that process the forms. However, this approach may limit the type of processing that can be specified.

Tasks as rule-based processing

Form-based task processing can be made more flexible in two ways. First, the messages can carry unstructured data (i.e., arbitrary text) in addition to the structured attributes of forms. Such messages have been called semi-structured [26, 32]. The messages can be further arranged in an inheritance lattice and classified according to their *purpose* (i.e., speech act—see Section 4). This allows the system to suggest a possible course of action to the user for the type of message received.

Second, the processing can be specified by means of production rules rather than algorithmic procedures. Such an approach is less structured than form-based processing since rules can be added to tasks in an incremental fashion. Rules can be grouped into tasks by the messages to which they are applicable and multiple levels of reasoning are also possible using rules. In the Information Lens, semi-structured messages and rules are used to filter and to classify received messages and to help users find messages that are not addressed to them [32].

Tasks as tool-based processing

At the most flexible end of Type I task processing (and where it starts to encroach on Type II task processing) is tool-based task processing [3, 21, 36, 37]. Many tasks in organizations are already tool-based (i.e., supported by software tools such as email, calendars, editors, etc.). However, it is the user that has the knowledge of what tools to use in a given situation. To provide greater support, one can provide a high-level description of a task consisting of the sub-tasks, the goals for each sub-task and, possibly, a tool to implement the sub-task. Such a high-level task description provides tool independence and links use of different tools to accomplish a task (Figure 2). Thus, the tools can change, but the high-level descriptions remain invariant (like data independence in database management systems); only the mapping need change. Note that the tools that are used can perform form-based, rule-based or any other kind of structured task processing.

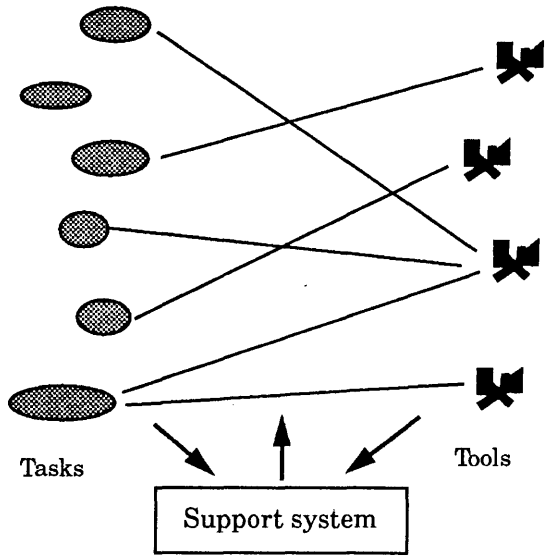

Figure 2. Tool-based task processing.

Such a tool-based approach, in effect, provides to the user an intelligent assistant who can recognize what task currently needs to be done and can suggest or invoke tools to assist in doing the task [21]. This combines the efficiency of Type I task processing with some of the flexibility of Type II task processing. However, since there is no general problem-solving mechanism, this approach is not classified as Type II task processing. Human agents do all the problem-solving. The system merely tracks the progress of a task invoking tools to assist/accomplish a sub-task where appropriate and, otherwise, reminding the user that a sub-task needs to be done before a task can proceed [3].

Tasks as problem-solving processing

In problem-solving task processing, goals and actions are used as an alternative to the procedural Type I task processing approach [1, 2, 4, 27, 31, 47]. However, traditional AI problem-solving (i.e., the state-operator paradigm) is generally not applicable to most organizational problem-solving situations as it is often not clear what the initial or goal states are [1]. The state-operator paradigm is more suited to a fact-centered deduction system. Object-

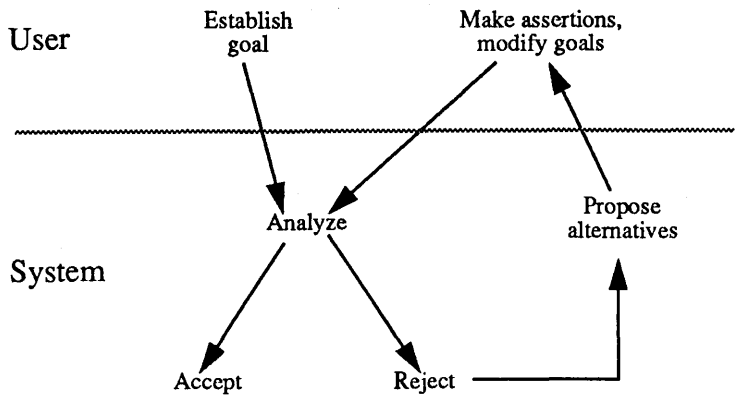

Figure 3. Problem-solving in the organizational context

centered systems seem to be more natural for organizational problem-solving. In supporting organizational problem-solving, we usually leave difficult problem-solving tasks to the user and simple problem-solving and algorithmic tasks to the computer. Problem solving is thus a cooperative effort between the user and the computer (i.e., the computer does not do it all by itself—see Figure 3).

At the heart of the problem-solving approach is a way for generating and executing plans. Plans are generated from descriptions of (sub)tasks which specify how a task is carried out, which things are affected by the task, and the goal of the task. These descriptions are used by a planner to generate a sequence of actions that should achieve the overall goal of the task.

The basic execution cycle of the Polymer planner is typical of the approach used [4]:

1. The user (through an application) posts a goal.
2. The planner generates a plan to achieve the goal using some planning technique (hierarchical planning in the case of Polymer). The planning result is a specification of the sequence of actions required to achieve the goal. Plans usually depend heavily on constraints provided by user actions (e.g., form-filling provides values for attributes). Plans thus, in general, will be partial. (The minimal requirement for the Polymer planner is to generate a set of expectations for the next step in the plan.)
3. The execution monitor selects an action for execution. This action may be executed directly by the system or may require input from the user. Assumptions behind planning decisions usually are recorded to allow for backtracking.
4. The execution monitor compares the actual action taken (and its result) with the expected action. If the action is O.K., then the execution monitor goes to step 2; else, it records the occurrence of an exception. Exceptions are caused by the open-ended nature of organizations (e.g., due to an inaccurate or incomplete plan description) and generally cause the system to replan the execution taking into account the changed circumstances.

4 Workflows

In the same way in which we characterized tasks as Type I and Type II, we can also characterize workflows as Type I (structured) and Type II (unstructured). A Type I workflow is one in which the agents to which the messages will go are known *a priori* and can thus be specified. A Type II workflow is one for which the agents to which a message will go are not known beforehand, but are determined dynamically as the message is processed. In the most open-ended case, a workflow can be completely impromptu as is the case where email messages, sent among agents, result in the accomplishment of perhaps a new activity for the organization.

Again, as for tasks, workflows fall on a spectrum from Type I to Type II, with the Type II end of the spectrum being open-ended. In the following sections, we discuss different approaches to specifying workflows starting with Type I and moving to Type II. It should be noted that, in general, research on workflows is less developed than that for tasks.

Workflow as message routing

For many organizational activities, the paths or routes that messages take through the organization often can be specified *a priori* as they are fixed and usually do not change [19, 20, 34, 43]. Such routing is usually specified for a class of messages so that the same routing applies to all instances of the class. However, just because the routing is specified *a priori* for a class does not mean that all instances of that class will follow exactly the same path through the organization. This is because the routing is not necessarily deterministic. For instance, the routing can depend on such factors as values in a message, load on an agent, etc. The routing also could be time dependent. That is, if a message has been with an agent more than a given amount of time it could be taken away from that agent and sent to another agent.

To allow for exceptional or unforeseen circumstances, there is usually a capability to "step outside" the pre-specified routing and to send messages to agents either not part of the original routing or in a different order than that specified in the routing. This introduces some degree of

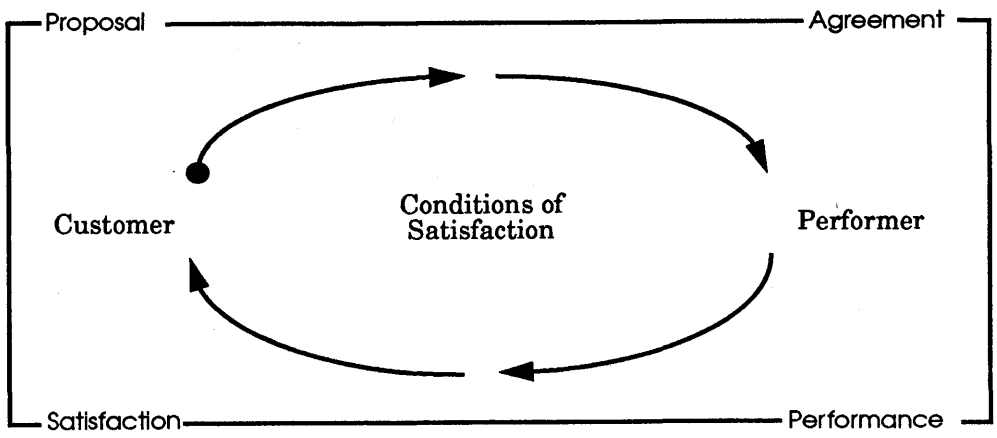

Figure 4. Language/Action workflow loop

flexibility into the routing specification. However, undoing a message routing is a difficult problem and may involve the use of compensating tasks [19].

Changing and distributing the routing for messages can pose a challenging problem. If the routing information travels with the messages, then there is no problem as each instance carries its own routing specification. However, this may be very wasteful of network bandwidth and of storage. On the other hand, keeping the routing information with each agent, while saving bandwidth and possibly storage, requires a mechanism for distributing the routing information to all relevant agents and applying it only to appropriate messages (i.e., those created after the routing was updated).

Workflow as language/action

While pre-specified message routing is obviously the most efficient, it leaves much to be desired in terms of flexibility and the handling of *ad hoc* situations. In addition, there are many messages that flow through an organization that are not structured or for which the routing is determined only as the messages are processed by each agent (e.g., email messages or other informal communication). A major problem faced by organizations is the tracking of the tasks that are initiated by such messages [7, 35].

The language/action approach to workflow attempts to address this problem. In this approach, speech acts are used to define and control the routing of messages. Message routing is not determined *a priori*, but is determined dynamically as the activity proceeds. The activity does not even have to be defined beforehand, but can also be determined dynamically as messages are sent and received. However, at any point in the activity, the possible next messages and/or routings of a message are partially or completely predetermined by the speech act of the received message.

The basic premise of the language/action approach is that speaking (communicating) and acting are synonymous. That is, when we say something, we also conduct a speech act which either requires the hearer to carry out some action or commits the speaker to perform some action. Thus, speech acts specify the "meaning" of what we say. They can be aggregated to form a *conversation* which is a sustained stretch of related speech acts. Conversations exhibit systematic regularities that can be used to control both the messages that can be formed and their routing.

The useful speech acts for forming conversations for workflows are [7, 48]:

- directives – which get the hearer to do something (e.g., request, suggest, beg, invite, warn, advise);
- commissives – which commit the speaker to some future course of action (e.g., promise, offer, permit, refuse).

These speech acts are useful for workflows because they result in the formation of *commitments* which bind the future behaviour of all parties and pledge them to certain activities/expectations [22].

The basic workflow loop (conversation) of proposing, agreeing, performing and satisfactorily completing an activity is shown in Figure 4 [35]. The conversation here is between two agents (customer and performer). For example, a customer (manager) might send a purchase order to a performer (purchasing agent). This results in a proposal (request) being issued by the customer to which there can be two possible replies on the part of the performer: agreement to the request (i.e., a promise to carry it out) which leads to its performance and, possibly, satisfactory completion or declining of the request (i.e., a refusal to do it). Agreement to carry out the request can lead to further requests being made by the performer (now playing the role of customer) resulting in the formation of further workflow loops for the purchase order. These (sub-)conversations can be tracked, their status queried, etc. [7]. It should be noted that the language/action approach can also be used to specify the processing that should be done by each agent [17, 18].

Workflow as intelligent messages

Our view of workflow so far has been that of passive messages circulating among the agents in an organization. However, there is no need for messages to be passive. They can be active in the sense that they can encode tasks and thus can do some processing themselves based on input re-

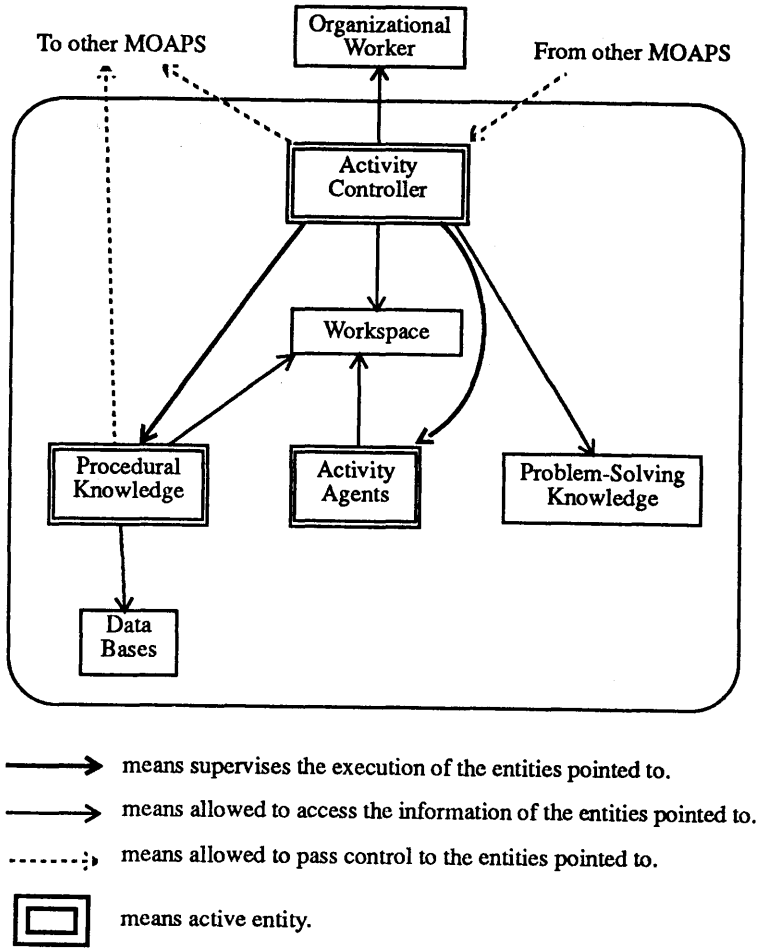

Figure 5. A micro-organization activity processor (MOAP).

ceived from an agent or on the nature of their environment. They can thus dynamically determine their routing and even possibly change their processing capabilities (tasks) as they travel around an organization [15, 45, 49]. This approach to workflows, while intriguing, has not been extensively explored to date.

5 Technology Integration

Up to this point we have discussed various technologies that have been used to support tasks and workflows. Each of these technologies, while useful in its own right, supports only part of the range of activities that occur in organizations. In this section, we describe an architecture that integrates some of the task and workflow technologies discussed so far.

Since our goal is to support the activities that occur within an organization, it seems reasonable to assume that the structure of a system that supports organizational activities would parallel that of the organization in which it is embedded [5]. To this end, it has often been observed that organizations behave much like a society (i.e., they are composed of many, perhaps specialized, (sub)units which each operate semi-autonomously, but often cooperatively, to accomplish some goal) [14, 23]. Therefore, in the integrating architecture, organizations are viewed as a composition of many units. Each one of these units is called a *micro-organization* because it behaves much like an organization except that it only contains a small domain of organizational knowledge. Similar to a society, organizational activities result from what happens inside micro-organizations as well as the interactions between them.

Assuming that we are given the structure of an organization in terms of micro-organizations[2], we can embed an activity support system within the organization by assigning an *activity processor* to perform all or part of the functions of all or some of the micro-organizations. An activity processor, in addition to coordinating the execution of the entities within itself, can communicate with other micro-organizations when it is necessary to do so. Figure 5 shows the system architecture of a micro-organization activity processor (MOAP). Although the architecture is the same for all the MOAPs, the knowledge between any two of them can be different and/or inconsistent.

As shown in Figure 5, there are six different types of entities in a MOAP: data repository, procedural knowledge, problem-solving knowledge, workspace, activity agent, and activity controller. Each one of them plays a distinct role in supporting organizational activities. Details on each

[2]In terms of our modeling framework, micro-organizations correspond to units and positions.

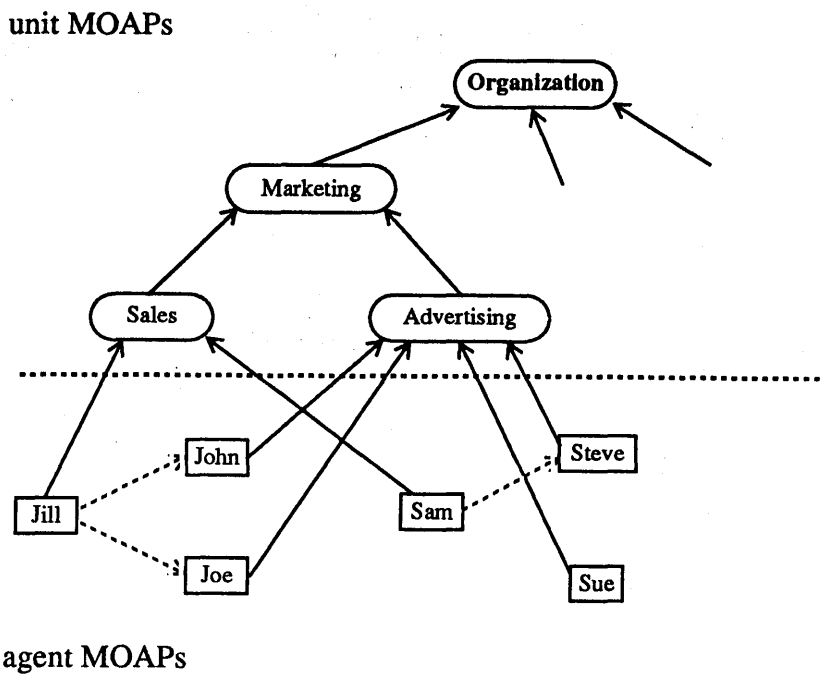

Figure 6. An example knowledge inheritance hierarchy of agent and unit MOAPs

of these types of entities can be found in [51, 50]; here we only describe them briefly.

- The **data repository** stores persistent information. The major criteria for data to qualify for storage in the data repository is that it be reliable, factual data that is amenable to structured storage and retrieval operations and that must be kept for some period of time.
- The **procedural knowledge** contains the well-thought-out, well-organized, well-tested, and well-adapted organizational knowledge. In the MOAP architecture, procedural knowledge is described in terms of tasks which can be broken down into a hierarchy of subtasks. Some of these subtasks can reside within other MOAPs in the organization (shown using the dotted arrows in Figure 5).
- The **problem-solving knowledge** describes strategies and serves as advice to the activity controller (i.e., it provides assistance to the activity controller when the procedural knowledge is insufficient to perform a task).
- The **workspace** serves as the communication medium for the active entities (i.e., procedural knowledge, activity agents, and the activity controller) in that information put into the workspace can be picked up by other active entities at a later time. In addition, activity agents from other MOAPs can access the workspace (through the activity controller).
- The **activity agent** is a representative of a MOAP which specializes in communicating with and collecting information from other MOAPs (i.e., external communication).
- The **activity controller** controls the execution of tasks and activity agents, and the use of problem-solving knowledge within its micro-organization jurisdiction. It also responds to inquiries from other MOAPs (either directly or via activity agents). In general, the bias of the activity controller is to try to use procedural knowledge whenever possible since it is generally more efficient to do so. The activity controller also provides facilities for interacting with the user.

As mentioned above, one MOAP can be assigned to each of the micro-organizations in an organization. Because each MOAP contains some organizational knowledge, the MOAP architecture explicitly partitions and distributes organizational knowledge among different MOAPs. Thus, the MOAP architecture can support private and, possibly, conflicting knowledge [13] within each MOAP. In addition, it can share (common) knowledge among the entities in an organization by means of a *knowledge inheritance hierarchy*. This hierarchy categorizes the knowledge contained in MOAPS into agent MOAPs and unit MOAPs [29] and relates these two types of knowledge in an inheritance hierarchy according to the structure of the organization (Figure 6).

Conceptually, an agent MOAP inherits all the procedural knowledge in the unit MOAPs with which it is associated. However, the owner of the agent MOAP can modify this procedural knowledge provided that the constraints set by the unit MOAPs are satisfied. In general, the knowledge between any two MOAPs can be inconsistent. However, knowledge among a unit MOAP and its associated unit/agent MOAPs is *always* consistent. This is because of the knowledge inheritance hierarchy. Thus, every MOAP is always *internally consistent*. However, MOAPs may be *externally inconsistent*. That is, if two MOAPs are not on the same path from the root in the knowledge hierarchy, then they may contain (some) inconsistent knowledge.

Supporting tasks

In the MOAP architecture, if a task is very structured and well understood (i.e., a Type I task), then it is encoded as procedural knowledge in which case it can be executed by the activity controller without the aid of problem-solving knowledge. For tasks for which no procedural knowledge exists, the activity controller first attempts to use the problem-solving knowledge available to it. If this is insufficient, it attempts to use activity agents to either acquire the needed knowledge or to request another MOAP to perform the task. If all these fail, then the activity controller consults the human agent to which the MOAP is assigned for assistance. Note that there is a smooth transition between Type I tasks and Type II tasks in this approach.

It is possible that a task that is initially a Type II task becomes quite structured and routine. In this case, it can migrate from problem-solving knowledge to procedural knowledge [29]. Conversely, if a Type I task encounters difficulty, then the activity controller can attempt to use problem-solving knowledge, activity agents, and ultimately a human agent to handle the task.

Supporting workflows

The MOAP architecture supports both pre-specified and *ad hoc* workflows. Pre-specified workflows are encoded as part of the procedural knowledge. For example, a MOAP can "make a call" to another MOAP, of which it is aware, to ask it to carry out some task on its behalf (dotted arrow from procedural knowledge to other MOAPs in Figure 5 and dotted arrows in Figure 6). These types of calls are equivalent to the workflows discussed in Section 4.

For *ad hoc* workflow, the activity controller can send out activity agents to interact with other MOAPs. Activity agents can either be sent to specific MOAPs or they can be sent to an arbitrary MOAP and thereafter determine their own routing through the organization. The interaction can be guided by speech acts [49].

6 Summary

In this paper we reviewed several approaches and technologies for supporting organizational activities. These were discussed within a modelling framework that views the activities in an organization as being composed of tasks that are initiated by agents in an organization and the (work)flow of messages among the agents. As well, an architecture that integrates task and workflow technologies was outlined.

While many of the technologies that have been proposed for supporting organizational activities are proving to be useful, lack of integration among them will prove to be a major barrier to improved support in the future. Thus, further research on integrating models and architectures is needed.

Additional technologies and issues that require further investigation include:

- facilities for migrating tasks between Type I and Type II. Most research has addressed the support of either Type I or Type II tasks without much regard to the fact that a Type I task can require Type II support or that a Type II task can evolve into a Type I task.
- development of appropriate building blocks for constructing activity support systems. To date, research efforts have used whatever implementation technologies were at hand. Malone has shown that a few constructs may be sufficient for building a wide variety of activity support systems [33].
- tools which allow the agents to see the context within which their tasks take place (i.e., to be able to see what tasks have already taken place and what tasks are likely to take place subsequently). Such tools would be a first step towards allowing tasks to be executed in a "what if" manner to determine their effect before actually executing them [44].
- facilities (user interfaces) for knowledge acquisition. Since organizational activities occur in an open-ended environment, knowledge about tasks and workflows is likely to be constantly changing. These changes need to be captured and incorporated into activities as they occur.
- support for negotiation activities. Organizational activities do not always occur in a cooperative environment. Often there is conflict (e.g., when determining budgets of organizational units) and a need for negotiation to arrive at a solution acceptable to all [25].
- facilities to help support the redesign of business processes. The introduction of new technology should be used as an opportunity to examine the business processes that will be supported to determine whether they should be redesigned to achieve the full benefits of the new technology [12].

References

1. G R Barber. 'Supporting organizational problem solving with a workstation'. *ACM Trans. Office Inf. Sys.*, 1(1):45–67, (1983).
2. N Bhandaru and W B Croft. 'An architecture for supporting goal-based cooperative work'. In S Gibbs and A A Verrijn-Stuart, eds., *Multi-User Interfaces and Applications*, pp. 337–354. Elsevier Science Publishers B.V., North-Holland, Amsterdam, (1990).
3. W B Croft and L S Lefkowitz. 'Task support in an office system'. *ACM Trans. Office Inf. Sys.*, 2(3):197–212, (1984).
4. W B Croft and L S Lefkowitz. 'Using a planner to support office work'. In *Proc. ACM Conf. Office Inf. Sys.*, pp. 55–62, (1988).
5. P de Jong. 'Structure and action in distributed organizations'. In *Proc. ACM Conf. Office Inf. Sys.*, pp. 1–10, (1990).
6. R E Fikes and D A Henderson, Jr. 'On supporting the use of procedures in office work'. In *Proc. 1st Annual Natl. Conf. Artificial Intelligence*, pp. 202–207, (1980).
7. F Flores, M Graves, B Hartfield, and T Winograd. 'Computer systems and the design of organizational

interaction'. *ACM Trans. Office Inf. Sys.*, **6**(2):153–172, (1988).

8. A C Fong. 'A model for automatic form-processing procedures'. In *Proc. 16th Hawaii Intl. Conf. Systems Sciences*, pp. 558–565, (1983).
9. D Georgakopoulos, M F Hornick, F Manola, S Heiler, F Nayeri, and B Hurwitz. 'An extended transaction environment for workflows in distributed object computing'. *IEEE Data Engineering*, **16**(2):24–27, (1993).
10. G A Gorry and M S S Morton. 'A framework for management information systems'. *Sloan Management Review*, **13**(1):55–70, (1971).
11. H Hammainen, E Eloranta, and J Alasuvanto. 'Distributed form management'. *ACM Trans. Inf. Sys.*, **8**(1):50–76, (1990).
12. M Hammer. 'Re-engineering work: Don't automate, obliterate'. *Harvard Business Review*, pp. 104–111, (July/August 1990).
13. C Hewitt. 'Offices are open systems'. *ACM Trans. Office Inf. Sys.*, **4**(3):271–287, (1986).
14. C Ho, Y Hong, and T Kuo. 'A society model for office information systems'. *ACM Trans. Office Inf. Sys.*, **4**(2):104–131, (1986).
15. J Hogg and S Gamvroulas. 'An active mail system'. In *Proc. ACM SIGMOD Conf.*, pp. 215–222, (1984).
16. J Hogg, O M Nierstrasz, and D C Tsichritzis. 'Office procedures'. In D Tsichritzis, ed., *Office Automation Concepts and Tools*, pp. 137–166. Springer-Verlag, Berlin, (1985).
17. S M Kaplan, A M Carroll, and K J MacGregor. 'Supporting collaborative processes with ConversationBuilder'. In *Proc. ACM Conf. Organizational Computing Sys.*, pp. 69–79, (1991).
18. S M Kaplan, W J Tolone, D P Bogia, and C Bignoli. 'Flexible, active support for collaborative work with ConversationBuilder'. In *Proc. Conf. Computer-Supported Cooperative Work*, pp. 378–385, (1992).
19. B Karbe and N Ramsperger. 'Influence of exception handling on the support of cooperative office work'. In S Gibbs and A A Verrijn-Stuart, eds., *Multi-User Interfaces and Applications*. Elsevier Science Publishers B.V., Amsterdam, (1990).
20. B Karbe, N Ramsperger, and P Weiss. 'Support of cooperative work by electronic circulation folders'. In *Proc. ACM Conf. Office Inf. Sys.*, pp. 109–117, (1990).
21. A R Kaye and G M Karam. 'Cooperating knowledge-based assistants for the office'. *ACM Trans. Office Inf. Sys.*, **5**(4):297–326, (1987).
22. C C Koo. 'A commitment-based communication model for distributed office environments'. In *Proc. ACM Conf. Office Inf. Sys.*, pp. 291–298, (1988).
23. W A Kornfeld and C E Hewitt. 'The scientific community metaphor'. *IEEE Trans. Systems, Man and Cybernetics*, **SMC-11**(1):24–33, (1981).
24. T Kreifelts, E Hinrichs, K H Klein, P Seuffert, and G Woetzel. 'Experience with the DOMINO office procedure system'. In *Proc. European Conf. Computer-Supported Cooperative Work*, pp. 117–130, (1991).
25. B Laasri, H Laasri, S Lander, and V Lesser. 'A generic model for intelligent negotiating agents'. *Intl J. Intelligent and Cooperative Inf. Sys.*, **1**(2):291–317 (1992).
26. K Y Lai, T W Malone, and K C Yu. 'Object Lens A 'spreadsheet' for cooperative work'. *ACM Trans Office Inf. Sys.*, **6**(4):332–353, (1988).
27. H Liu, I Dranffan, and F Poole. 'A goal oriented office form system'. In *Proc. ACM Conf. Organizationa Computing Sys.*, pp. 123–128, (1991).
28. F H Lochovsky. 'Improving office productivity: A technology perspective'. *Proc. IEEE*, **71**(4):512–518 (1983).
29. F H Lochovsky, C C Woo, and L J Williams. 'A micro-organizational model for supporting knowledge migration'. In *Proc. ACM Conf. Office Inf. Sys.*, pp 194–204, (1990).
30. V Y Lum, N C Shu, F Tung, and C L Chang. 'Automating business procedures with form processing' In N Naffah, ed., *Office Information Systems*, pp. 7-38. North-Holland, Amsterdam, (1982).
31. R Lutze. 'Customizing cooperative office procedures by planning'. In *Proc. ACM Conf. Office Inf. Sys.*, pp 63–77, (1988).
32. T W Malone, K R Grant, F A Turbak, S A Brobst, and M D Cohen. 'Intelligent information sharing systems' *Comm. ACM*, **30**(5):390–402, (1987).
33. T W Malone, K Y Lai, and C Fry. 'Experiments with Oval: A radically tailorable tool for cooperative work'. In *Proc. Conf. Computer-Supported Cooperative Work*, pp. 289–297, (1992).
34. M S Mazer and F H Lochovsky. 'Logical routing specification in office information systems'. *ACM Trans. Office Inf. Sys.*, **2**(4):303–330, (1984).
35. R Medina-Mora, T Winograd, R Flores, and F Flores. 'The action workflow approach to workflow management technology'. In *Proc. Conf. Computer-Supported Cooperative Work*, pp. 281–288, (1992).
36. S Nirenburg and V Lesser. 'Providing intelligent assistance in distributed office environments'. In *Proc. ACM Conf. Office Inf. Sys.*, pp. 104–112, (1986).
37. M Palaniappan, N Yankelovich, G Fitzmaurice, A Loomis, B Haan, J Coombs, and N Meyrowitz. 'The Envoy framework: An open architecture for agents'. *ACM Trans. Office Inf. Sys.*, **10**(3):233–264, (1992).
38. J D Palmer and N A Fields. 'Computer-supported cooperative work'. *IEEE Computer*, **27**(5):15–17, (1994).
39. R R Panko. 'Spending on office systems: A provisional estimate'. *Office: Technology and People*, **1**:177–194, (1982).
40. R R Panko. 'Productivity trends in certain office-intensive sectors of the U.S. federal government'. *ACM Trans. Office Inf. Sys.*, **3**(4):370–379, (1985).
41. R R Panko and R H Sprague. 'Toward a new framework for office support'. In *Proc. ACM SIGOA Conf. on Office Inf. Sys.*, pp. 82–92, (1982).
42. W Rupietta. 'An organization and resources model for adapting office systems to organizational structures'. In *Proc. DEXA Conf.*, pp. 346–350, (1994).

43. S K Sarin, K R Abbott, and D R McCarthy. 'A process model and system for supporting collaborative work'. In *Proc. ACM Conf. Organizational Computing Sys.*, pp. 213–224, (1991).
44. K D Swenson. 'Visual support for reengineering work processes'. In *Proc. ACM Conf. Organizational Computing Sys.*, pp. 130–141, (1993).
45. D Tsichritzis, E Fiume, S Gibbs, and O Nierstrasz. 'KNOs: KNowledge acquisition, dissemination and manipulation Objects'. *ACM Trans. Office Inf. Sys.*, **5**(1):96–112, (1987).
46. D C Tsichritzis. 'Form management'. *Comm. ACM*, **25**(7):453–478, (1982).
47. F von Martial. 'A conversation model for resolving conflicts among distributed office activities'. In *Proc. ACM Conf. Office Inf. Sys.*, pp. 99–107, (1990).
48. T Winograd. 'A language/action perspective on the design of cooperative work'. *Human-Computer Interaction*, **3**(1):3–30, (1987-88).
49. C C Woo. 'SACT: A tool for automating semi-structured organizational communication'. In *Proc. ACM Conf. Office Inf. Sys.*, pp. 89–98, (1990).
50. C C Woo and F H Lochovsky. 'Supporting distributed problem solving in organizations'. *ACM Trans. Office Inf. Sys.*, **4**(3):185–204, (1986).
51. C C Woo and F H Lochovsky. 'Knowledge communication in intelligent information systems'. *Intl. J. of Intelligent and Cooperative Inf. Sys.*, **1**(1):203–228, (1992).
52. S B Yao, A R Hevner, Z Shi, and D Luo. 'FORMANAGER: An office forms management system'. *ACM Trans. Office Inf. Sys.*, **2**(3):235–262, (1984).

Acknowledgements: An earlier version of this paper appeared in *ACM Japan Symposium on Computers as Our Better Partners*, 1994. Our research is supported by the UPGC Research Grants Council of Hong Kong.

Received: October 1994

Editor's Notes

As editor, I have increasingly felt the need for a sounding board to test ideas and to critique the way things are going. Although email communication with the editorial board and other members of the SACJ team meets this need to some extent, it is not quite the same as having someone in the immediate vicinity with whom one can discuss issues of the moment. For this reason, the editorial board has agreed that Prof Lucas Introna at Pretoria University should take over the task of the Information Systems subeditor of the journal. Lucas replaces Prof John Shochot who has ably handled the task for the past few years. I would like to thank John for his work to date, and to welcome Lucas onto the SACJ team. IS contributions to SACJ should be forwarded directly to the new IS subeditor whose address appears on the front inside cover.

Apart from the normal editions of SACJ, two further special editions are planned for the future. The first will be published early in 1995, and will present material covered at the WOFACS workshop in July this year. Although this material is somewhat specialised, it has been found to be particularly useful to those who teach and research in formal aspects of Computer Science. Furthermore, it comes with no extra cost to subscribers. The second special edition will deal with the very topical issue of IT and development. Several experts in the field are being invited to submit articles, and a call for contributions will be found in this edition. This initiative is the brainchild of the new IS subeditor, and will be co-ordinated by him.

Derrick Kourie
Editor

SACJ is produced with kind support from
Mosaic Software (Pty) Ltd.

Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research papers. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as Communications or Viewpoints. While English is the preferred language of the journal, papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted in triplicate to the editor.

Form of Manuscript

Manuscripts for *review* should be prepared according to the following guidelines.

- Use wide margins and $1\frac{1}{2}$ or double spacing.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be numbered and titled. Figures should be submitted as original line drawings/printouts, and not photocopies.
- References should be listed at the end of the text in alphabetic order of the (first) author's surname, and should be cited in the text in square brackets [1–3]. References should take the form shown at the end of these notes.

Manuscripts accepted for publication should comply with the above guidelines (except for the spacing requirements), and may be provided in one of the following formats (listed in order of preference):

1. As (a) LaTeX file(s), either on a diskette, or via e-mail/ftp – a LaTeX style file is available from the production editor;
2. As an ASCII file accompanied by a hard-copy showing formatting intentions:
   - Tables and figures should be on separate sheets of paper, clearly numbered on the back and ready for cutting and pasting. Figure titles should appear in the text where the figures are to be placed.
   - Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin, if they are not clear in the text.

   Further instructions on how to reduce page charges can be obtained from the production editor.
3. In camera-ready format – a detailed page specification is available from the production editor;
4. In a typed form, suitable for scanning.

Charges

Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect scanning, typesetting, reproduction and other costs. Currently, the minimum rate is R30-00 per final page for LaTeX or camera-ready contributions that require no further attention. The maximum is R120-00 per page for contributions in typed format (charges include VAT).

These charges may be waived upon request of the author and at the discretion of the editor.

Proofs

Proofs of accepted papers in categories 2 and 4 above may be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Camera-ready submissions will only be accepted if they are in strict accordance with the detailed guidelines. It is the responsibility of the authors to ensure that their submissions are error-free.

Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to less than about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement

Placement of advertisements at R1000-00 per full page per issue and R500-00 per half page per issue will be considered. These charges exclude specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.

References

1. E Ashcroft and Z Manna. 'The translation of 'goto' programs to 'while' programs'. In *Proceedings of IFIP Congress 71*, pp. 250–255, Amsterdam, (1972).
2. C Bohm and G Jacopini. 'Flow diagrams, turing machines and languages with only two formation rules'. *Communications of the ACM*, **9**:366–371, (1966).
3. S Ginsburg. *Mathematical theory of context free languages*. McGraw Hill, New York, 1966.

South African
Computer
Journal

Number 12, November 1994
ISSN 1015-7999

Suid-Afrikaanse
Rekenaar-
tydskrif

Nommer 12, November 1994
ISSN 1015-7999

Contents

UP Drukkers
UP rinters