North-Holland
Microprocessing and Microprogramming 27 (1989) 455–462

An Automatic Distributed Calendar and Appointment System

Friedemann Mattern and Peter Sturm

INCAS Project,*) University of Kaiserslautern, Computer Science Department, P.O. Box 3049,
D-6750 Kaiserslautern, Federal Republic of Germany
E-mail: mattern@uklirb.uucp and sturm@uklirb.uucp

We give a short overview of a prototype version of a fully distributed calendar and appointment system implemented on a network of workstations. We describe the software architecture, the appointment scheduling strategies and heuristics, the distributed implementation language, and the graphical user interface. The motivations for the realization of the system within a distributed systems research project are also discussed.

1. Introduction

Calendars are important tools for office workers, and several *electronic calendars* with features such as automatic alarm and reminder facilities have been realized and incorporated in office computer systems. However, as Kincaid et al. have observed [KIN85], these electronic calendars do not yet offer the power and flexibility of traditional paper calendars. Furthermore, most of them are either stand-alone systems or do not contain facilities for automatically *scheduling appointments involving multiple participants*, although this additional capability would be a clear advantage of automatic calendars over traditional paper calendars. Two notable exceptions are the Eden Shared Calendar Systems [HOL85] and the Amoeba Diary [JOH88]. The latter, however, is not a fully distributed system since it contains a so-called "global module" which acts as a centralized scheduling manager.

The design and implementation of a useful and practical automatic distributed calendar and appointment system represents a challenge in many respects. To be widely usable, appointment systems have to fulfill several requirements: They should behave in a natural manner in order to gain high user acceptance, they should have a comprehensive set of functional capabilities, and the user interface should be simple and preferably graphical, similar to a classical paper calendar. Because users of an appointment system are typically geographically distributed, its implementation as a distributed program seems to be quite natural. Centralized solutions quickly lead to bottlenecks with a growing number of participants. However, in order to behave sensibly, a fully distributed appointment system—which consists of a set of cooperating autonomous calendars without a central management process or global database—has to solve a complex and difficult task, comparable to a group of people trying to fix a common schedule by sending letters to each other.

Our appointment system has been implemented within the INCAS ("Incremental Architecture for Distributed Systems") distributed systems research project which investigates the benefits and problems of distributed and parallel systems [NEH87, STU89]. The project covers a wide scope ranging from hardware development over distributed operating systems up to high-level distributed applications and algorithms [MAT87] with special focus on development tools [BUH89], graphics based design methodologies [STU89b], programming languages [WYB89], and debug and monitoring support for distributed systems [WYB88]. The calendar and appointment system is one of several distributed applications which have been realized within the project to evaluate the methodologies and development tools. The main purpose of the prototype system was to identify and solve interesting and fundamental problems related to decentralization of data and control. Therefore, the primary goal was not the realization of a perfect and ready to use appointment system, but to gain experience in the specification and programming of a non-trivial distributed application.

The distributed calendar system, which has been implemented using an experimental distributed programming language described in Section 5, operates *without global knowledge* and in an *open environment*, therefore new participants can be incorporated easily at any time. Elaborate *heuristics* are used to guarantee a natural behavior of the scheduling mechanism and to allow the system to react in an adaptive way to specific user requirements. Furthermore, besides being a practical test for distributed language constructs and programming paradigms, the system serves as a prototype application to examine various methods for distributed problem solving and to gain experience with several application independent distributed control algorithms such as message-based distributed mutual exclusion, agreement, and election protocols.

*) funded by the Deutsche Forschungsgemeinschaft as part of the Research Institute SFB 124 "VLSI-Design and Parallel Architectures", Kaiserslautern - Saarbrücken, Federal Republic of Germany

2. The Architecture of the Appointment System

The architecture of the realized distributed calendar and appointment system is shown in Figure 1. Each user has access to the facilities offered by the system through a dedicated dialogue manager and an associated autonomous calendar agent.

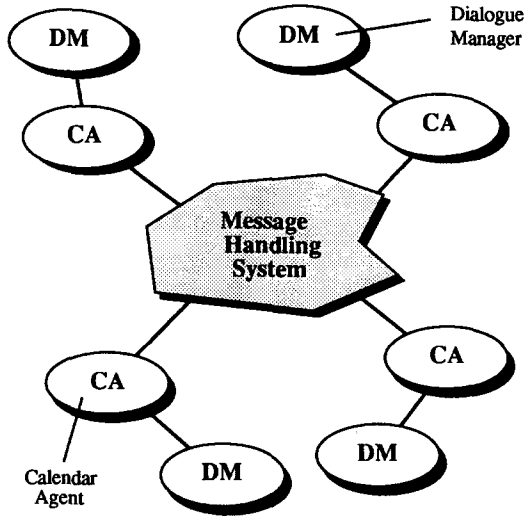

Fig. 1: Architecture of the Appointment System

The *dialogue manager* provides a graphical user interface and resides on a workstation with a bitmap display. It enables the user to inspect and manage his or her private calendar, offers facilities to schedule new appointments, and notifies the user when new appointments are scheduled, confirmations are requested, or already fixed appointments are changed. The dialogue manager communicates solely with its associated calendar agent.

A *calendar agent* manages the user's private calendar. It provides basic management functions, among other things the addition of new private appointments (involving no other participants), the modification and deletion of already fixed appointments, and the querying of the calendar's contents. The calendar agents have no direct access to the calendars of other participants. However, by sending messages to each other, they *cooperate* to schedule an appointment which involves multiple participants. Viewed in this way, the calendar agents act as a *society of communicating problem-solving experts*; the negotiation protocol can therefore be seen as a heuristics driven anthropomorphic *distributed problem solving process.*

Communication between calendar agents is established by a *message handling system* which provides various high-level communication features (such as virtual broadcast facilities) appropriate for the calendar and appointment system. The message handling system is part of an application-independent *support layer* which facilitates the development of distributed programs. Besides establishing reliable communication support, this layer comprises distributed control algorithms such as mutual exclusion and global snapshot computation which can be called by any distributed application implemented on top of the support layer.

3. Scheduling Strategies and Heuristics

Besides the basic management functions concerning the private calendar of each user, the major task of a calendar agent consists in the negotiation of appointments involving *multiple participants*. Most systems are only capable to schedule an appointment when at least one proposal made by the initiator does not collide with the calendar contents of all participants. The probability of a successful negotiation therefore decreases rapidly the more participants are involved.

To be truly useful, more sophisticated negotiation strategies should be incorporated into an appointment system. Two extensions are conceivable and are realized in our system. First, the system should be capable of *rescheduling* less important appointments if no proposal made by the initiator is acceptable to all participants. Second, it should be possible to automatically remove less important participants if their calendar contents disable all proposals of the initiator and if the colliding appointments cannot be rescheduled. Often, appointments can only be scheduled successfully when both strategies are applied.

To realize these strategies, additional information must be available to the calendar and appointment system. For example, a priority number should be assigned to each user which reflects his or her importance within the organization. This priority value serves as a decision base for appointment rescheduling. Only scheduling commands initiated by more important users are allowed to reschedule appointments of initiators with a smaller priority value. Already fixed appointments also have a priority value. This appointment priority determines the easiness of automatic adjournment. As part of the appointment information, a third type of priority is assigned to every participant which is needed for the decision to remove less important participants.

The automatic rescheduling of appointments enforces the system to be aware of user preferences. This knowledge has been incorporated into the appointment system by the use of so-called *user profiles* which can be easily specified in a graphical manner (cf. Figure 2). By a user profile, every participant tells the system for which time periods he or she is willing to accept appointments. The "degree of reluctance" can be specified by using a value between 1 and 11; the value 11 signifies that an appointment is definitely not possible. The user profile reflects general user attitudes and pref-

erences on a weekly basis which are independent of specific appointments.

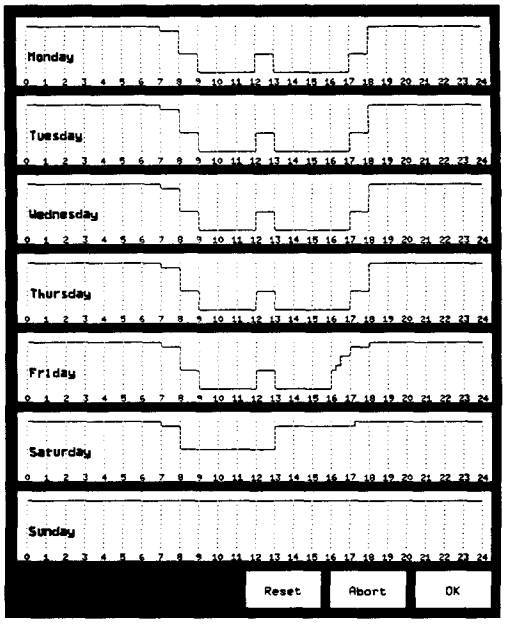

Fig. 2: Graphical Specification of a User Profile

A user who wants to schedule an appointment defines an appropriate *appointment profile.* This profile consists of a copy of the initiator's user profile (possibly updated by additional and specific time constraints), the list of participants, and an indication of the expected appointment duration. Optional attributes such as

- time range,
- dependence on other appointments,
- importance of every participant of this appointment,
- a value indicating the possibility of automatic adjournment,
- explicit confirmation request

complete the profile. Also, additional scheduling hints such as "schedule as early as possible" can be specified. All attributes are initialized with appropriate default values

After the receipt of a scheduling command, the initiator's calendar agent determines several proposals within the indicated time range. This *make proposals strategy* (cf. Figure 3) is assisted by heuristics which, among other things, decide whether the proposals should be chosen at random, or selected partially from the beginning, middle, and end of the time interval, or whether the proposals should be all located at the beginning (if the "schedule as early as possible" hint is given). If not enough proposals can be found, the "make proposals" strategy tries to reschedule appointments in the initiator's calendar supported by other heuristics. Currently, the number of proposals is fixed, but a future extension might be that the heuristics decide to increase this value with the number of participants and with the duration of the appointment. This would counteract the increasing probability of conflicting situations.

The appointment proposals are propagated to the calendar agents of all participants. As part of the *assessment strategy* (cf. Figure 3) every proposal is evaluated by the participant's calendar agent using the local knowledge of its calendar content and user profile. If no appointment in the calendar collides with the proposal, an assessment value is assigned to the proposal which is equal to the maximum value in the corresponding time interval of the user profile. Otherwise, approximate costs are assigned to the proposal, reflecting the rescheduling costs of the colliding appointments. These rescheduling costs are estimations based on the local information of the participant and cannot be exact, because dependencies on other appointments are in most cases unknown to the participant's calendar agent. The evaluation of costs in complex situations is supported by specific heuristics. In a future version of the appointment system the extension to user-definable assessment heuristics will be considered. However, the user's influence must be restricted in some way to fit into the overall scheduling strategies.

The assessment information of all participants is collected by the initiator. If all assessment values of one proposal are less than a given limit—defined by the initiator—and no colliding appointment exists, the appointment negotiation can be finished successfully. A positive notification accompanied by the complete appointment information is sent to the initiator's dialogue manager and to all participants. Often, however, no satisfying solution can be found directly. Then, a three stage *conflict resolution strategy* (cf. Figure 3) is applied. First, further time period proposals are made—if possible—, and control is given back to the "make proposals" strategy.

If this does not succeed, the calendar agent of the initiator tries to reschedule less important appointments in a second stage. For that, "conflict resolution" heuristics look for the most promising time periods making use of the assessment information previously obtained. The probability to be a promising candidate increases if only few appointments with small predicted rescheduling costs collide and the assessment values of the remaining conflict-free participants are small. An overall assessment value of the proposal is not allowed to exceed a certain threshold. This threshold increases slowly if no agreement is obtained until a user defined value is reached. Additional informations such as user

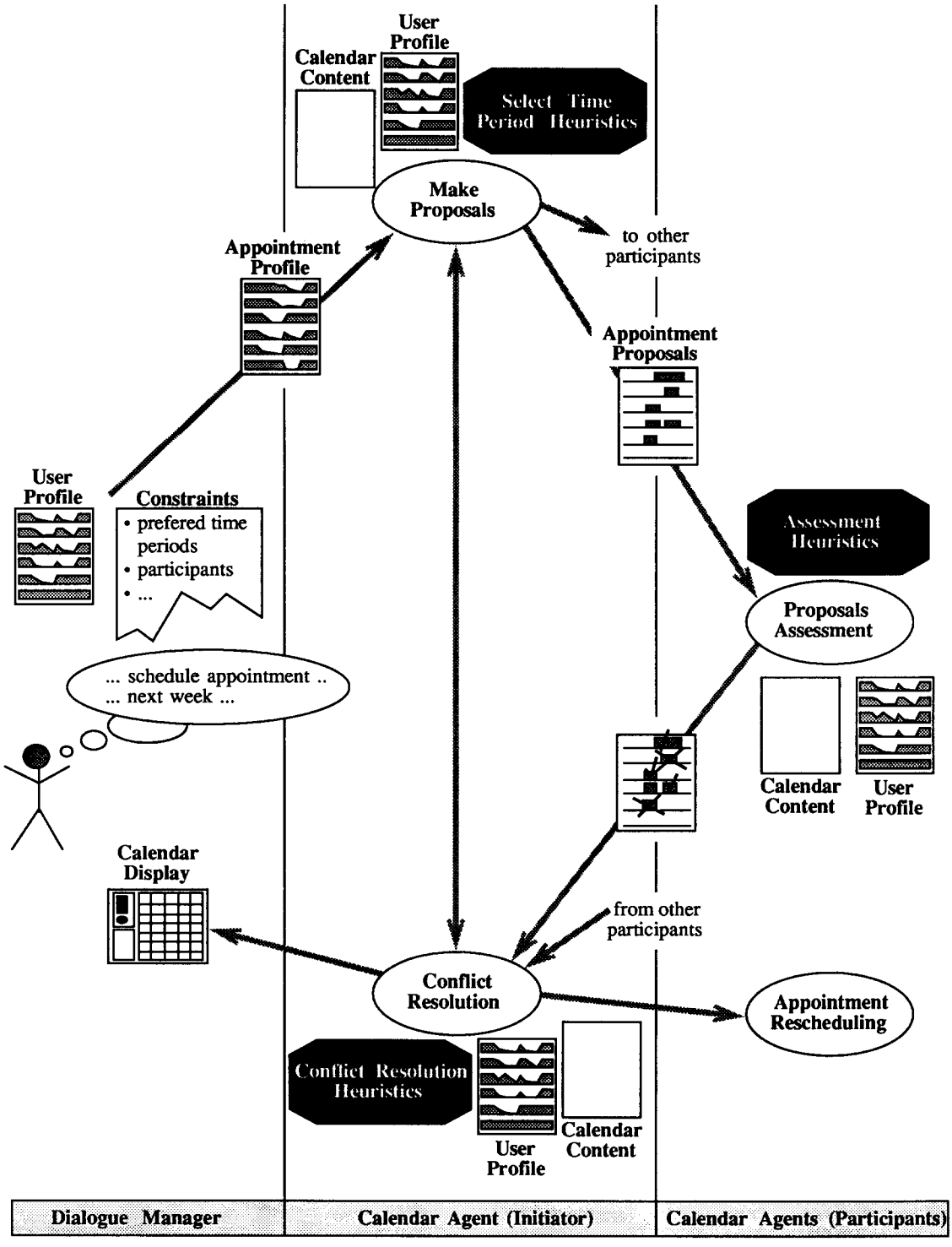

Fig. 3: Scheduling Strategies and Heuristics

preferences, scheduling hints, and appointment density in the initiator's calendar do also affect the heuristics. For every colliding appointment of a chosen time proposal, a rescheduling command is sent to the participant. Then, the calendar agent waits for the receipt of positive or negative acknowledgments. On positive acknowledgments, the appointment negotiation is completed. If at least one negative acknowledgment arrives, the heuristics (driven by user specific directives) decide how long to continue with stage two (using less promising candidates) or when to begin with stage three if no satisfying solution can be found.

In stage three it is tried to remove less important participants with colliding appointments or high assessment values. Additional rules are used by the conflict resolution heuristics if the set of conflicting participants and the set of less important participants are disjoint. Then for example, the user may be asked to increase the time range in order to enable the strategies to find additional time slots. If none of these measures lead to success, the calendar agent notifies the user about the unsuccessful result by sending an appropriate message to the dialogue manager.

Since in a fully distributed and symmetric solution all calendar agents are identical and no one has a complete and consistent view of the global state, the realization of control tasks such as mutual exclusion, deadlock avoidance, and concurrency control is a non-trivial problem that has to be solved by the appointment negotiation strategies. Because message delays cannot be ignored and any calendar agent can initiate the negotiation protocol at any time—independently of other agents—possible collisions and conflicts must be detected or avoided. This necessitates the use of sophisticated distributed control algorithms and protocols which are part of the already mentioned application-independent support layer.

Although most of these strategies and heuristics have been implemented, they have not yet been fully assessed. The heuristics can—up to a certain degree—be dynamically adapted to specific user requirements by using user and appointment profiles as well as additional scheduling hints. These mechanisms allow to largely influence the behavior of the appointment system. However, further work has to be put into the development of more sensible and adaptable heuristics and scheduling strategies in order to get a more flexible appointment system.

Currently, the overall strategy of the appointment system is sometimes unable to find a solution, although in principle a solution satisfying the constraints defined by the appointment initiator and the participants would exist. The reason is that proposals, which have once been refused based on the incomplete knowledge by the heuristics, are not taken into consideration again. Here, the incorporation of back-tracking strategies which roll-back to earlier abandoned decisions would provide more possibilities. However, the complexity of the negotiation strategies would be greatly increased by such mechanisms.

In the next version of the appointment system the interactive participation of the user in the distributed negotiation process will be considered. The user may evaluate several proposals of the system. By the interaction with users the appointment system should become aware of user preferences. Users should be able to end the interaction with the negotiation process at any time if the strategy of making appointments seems to be suitable. Participating in the decision process should lead to further adaption to specific user needs and should lead to a more realistic behavior of the system.

4. The Graphical User Interface

Much effort has also been spent on the design of an easy to use graphical user interface (cf. Figure 4). In many respects it resembles a traditional paper based calendar. The interface has been written in C and uses multiple window facilities offered by the X Window System. The user can easily move through the calendar and switch between different display formats. User profiles and appointment profiles are displayed graphically and can be changed and updated very quickly (cf. Figure 2). Default values and repetition facilities on a daily, weekly, and monthly basis further simplify the specification of profiles.

In contrast to paper based calendars, computer based calendar and appointment systems introduce some interesting advantages. Besides automatic rescheduling facilities, the notification of events such as positive or negative acknowledgments, new scheduled appointments, requested confirmation, and changes in already fixed appointment can be animated graphically. In the first prototype of the graphical user interface, days with new and yet unread notification messages are flagged by an exclamation mark. The mark is removed when the information is presented on screen. It became quickly apparent that this mechanism was not sufficient. For example, a deleted appointment cannot be reconstructed by the user; the exclamation mark is only capable to signal that something happened. Currently, a *notification animator* as part of the user interface is being developed. The animator collects every incoming notification message. After user request, it presents the collected informations in a menu and enables the user to replay selected items in a graphical manner. For example, changes in the beginning time of an appointment are animated by an arrow moving from the old to the new beginning time. More sophisticated mechanisms are imaginable and will possibly be included in a future version.

The adequate graphical representation of the interactive negotiation process described in the previous section is another interesting subject. New concepts and mechanisms must be defined to present the large

amount of information in a high-level and user-oriented manner. In our opinion, the employment of graphical mechanisms and their application to the user interface of the calendar and appointment system is promising and may bridge the gap between traditional paper calendars and current computer based calendars.

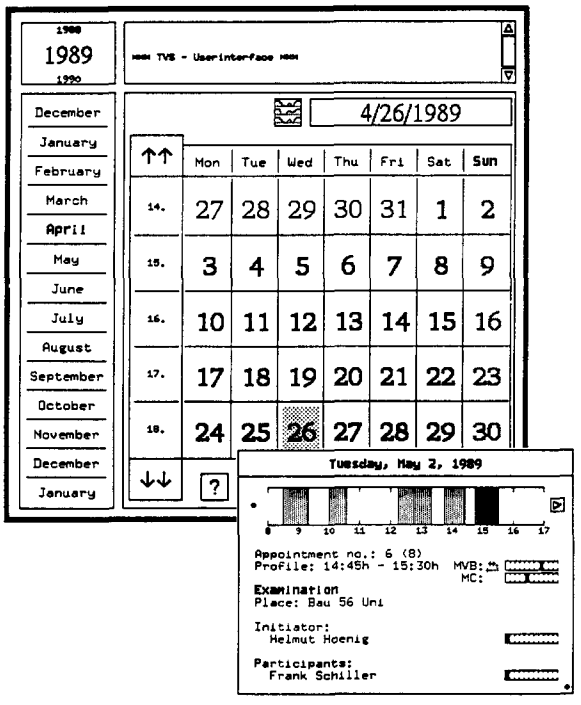

Fig. 4: The Graphical User Interface

5. The Implementation Language CSSA

The calendar agents, the message handling system, and the support layer have been implemented in the distributed programming language CSSA. CSSA (*C*omputing *S*ystem for *S*ocieties of *A*gents) is an experimental high-level programming language for expressing message-driven distributed algorithms which involve many loosely coupled cooperating tasks. The underlying model of distributed computations is based on the notion of *actors* originally developed by Hewitt [HEW77]. Its object-oriented philosophy provides clean mechanisms for exploiting parallelism and is especially well-suited for distributed programming.

The sequential structures and data types of CSSA are similar to those of Pascal, but concepts of modularization and data abstraction have been combined in a homogeneous way to allow a structured implementation of distributed applications. Data values of any type, including those of recursively defined complex types (i.e. arrays, records, sets) and structures built up by dynamic records and pointers, can be transmitted by messages.

Computations are performed by *agents*, which can be created dynamically and which are active objects that communicate with other agents solely by message passing. An agent is an autonomous entity consisting of several clusters of *operations*. An operation can be activated by sending a message to the agent. The basic communication construct is the asynchronous *send statement*

send <operation-name> (<message>)
to <target-agent>

which does not cause the sender to wait. Each agent processes only one message at a time without interruption; messages arriving at an agent while it is executing an operation are collected in a private *mailbox*. Execution of an operation may result in any number of messages being concurrently transmitted to other agents and many agents may be sending or receiving messages at the same time. *Acquaintances* with other agents may be transmitted in messages. Therefore, the *agent net*, which illustrates the potential flow of information, may dynamically change during a computation.

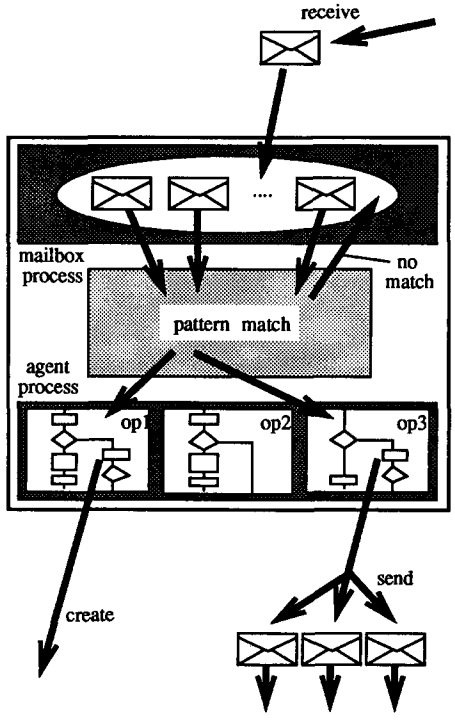

Fig. 5: A CSSA Agent

A cluster of operations comprising local variable decla-

rations is called a *facet*. At any given time, in every agent only one facet—the current facet—is active. Dynamic replacement of the current facet by another facet may change the behavior of an agent. Facets can also be set up recursively, the return to a previous facet then restores the old state

An agent is *passive* whenever it is not executing an operation. Then the mailbox is scanned implicitly for a message which names an operation of the current facet. An operation describes the message it is ready to accept by a pattern and an assertion. Among other things, the *pattern* is used to break up composite data structures to extract pieces of the message and bind them to local variables, while the *assertion* allows the use of an arbitrary predicate on the values of the message and the variables of the agent. If the pattern-match succeeds and the assertion evaluates to true, the operation is executed with the actual variable-bindings, similar to the execution of a procedure. Otherwise, the message remains in the mailbox without any side effects and its match is retried at a later time. This *implicit message receipt* is the normal case for agents acting as servers. *Explicit message receipt* within an operation can be programmed using a blocking or non-blocking *receive statement*.

The user is fully integrated into the CSSA system. By means of a specific agent connected to a bit-map terminal, the so-called *interface agent*, the user takes part in the distributed computation. The interface agent consists of a CSSA interpreter which can be programmed dynamically during execution. Similiar to all other agents, the interface agent can send and receive messages and create new agents. In addition, it comprises various features for mointoring distributed programs.

A first implementation of the CSSA-system for an experimental system of five MC68000 microcomputers was started at the beginning of the INCAS Project in 1983. In 1986, a new version of the compiler running under UNIX and generating code for a virtual stack machine was realized. CSSA is now available on a network of UNIX machines and Sun workstations connected by Ethernet using a TCP/IP-based protocol.

6. Conclusions

The realization of a first prototype version of the distributed calendar and appointment system was an interesting project and many experiences have been gained concerning specification, implementation, debugging, and testing large distributed programs. The lack of adequate tools in these areas was particularly noticeable. The mastery of the additional complexity of distributed programs caused by parallelism, non-determinism, decentralization of data and control, and temporary inconsistencies is still a major challenge for the designer and software engineer. Despite these difficulties, a working and stable prototype is now operational and will be used on a trial basis within our research group which consists of about ten persons. Besides small optimizations and fine tuning of the interface, we expect that the use of the system will yield a list of requirements for a next version.

Future work will concentrate on improved scheduling heuristics, the use of persistent objects for the storage of the calendar contents, and possibly also the use of electronic mail and directory standards for the underlying message handling system.

Acknowledgments

We would like to thank Helmut Hönig, Dagmar Mörscher-Krämer, Axel Schindler, Elke Schmidt, and Kai Wolf who assisted us in the design and implementation of the distributed calendar and appointment system.

References

[BUH89] **Buhler, P. and Wybranietz, D.**, Tools for Distributed Programming in the INCAS Project, this volume

[HEW77] **Hewitt, C.**, Viewing Control Structures as Patterns of Passing Messages, Artificial Intelligence, Vol. 8, pp. 323-364, 1977

[HOL85] **Holman, C. and Almes G.**, The Eden Shared Calendar System, Technical Report TR 85-05-02, University of Washington, Seattle, Washington.

[JOH88] **Johansen, D. and Anshus, O.J.**, A Distributed Diary Application, in: Cerveira, A.G. (ed.), Computer Communication Systems (Elsevier Science Publishers, Amsterdam, 1988), pp. 73-82.

[KIN85] **Kincaid, C.M., Dupont, P.B., and Kaye, A.R.**, Electronic Calendars in the Office, ACM Trans. on Office Information Systems, Vol. 3, No. 1, 1985.

[MAT87] **Mattern, F.**, Algorithms for Distributed Termination Detection, Distributed Computing, Vol. 2, No. 3, pp. 161-175, 1987

[NEH87] **Nehmer, J., Haban, D., Mattern, F., Wybranietz, D., and Rombach, H.D.**, Key Concepts of the INCAS Multicomputer Project, IEEE Trans. Software Eng., Vol. SE-13, No. 8, pp. 913-923, 1987

[STU89] **Sturm, P., Wybranietz, D., and Mattern, F.**, The INCAS Distributed Systems Project— Experiences and Current Topics, Proc. Workshop "Distribution and Objects", DEC Karlsruhe, pp. 97-114, 1989

[STU89b] **Sturm, P., Buhler, P., Mattern, F., and Wybranietz, D.** An Integrated Environ-

ment for Programming, Evaluating, and Visualizing Large Distributed Systems, Workshop on Parallel Computing in Practice, Jerusalem, Israel, 1989

[WYB88] **Wybranietz, D. and Haban, D.**, Monitoring and Performance Measuring Distributed Systems During Operation, Performance Evaluation Review, Vol. 16, No. 1, pp. 197-206, 1988

[WYB89] **Wybranietz, D. and Buhler, P.**, The LADY Programming Environment for Distributed Operating Systems, to appear in Proc. of the PARLE Conference, Springer-Verlag LNCS, 1989