A Meeting Scheduling System Based on Open Constraint Programming

Kenny Qili Zhu and Andrew E. Santosa

Department of Computer Science, National University of Singapore
S16, 3 Science Drive 2, Singapore 117543, Republic of Singapore
{kzhu,andrews}@comp.nus.edu.sg

Abstract. In this paper, we introduce a meeting scheduling system based on open constraint programming (OCP) paradigm. OCP is an extension to constraint logic programming (CLP), where a server capable of executing constraint logic programs acts as a mediator of communicating reactive agents. A meeting among several users can be scheduled automatically by a constraint logic program in the server based on the meeting participants' preferences. Flexible user preferences can be programmed using the OCP reactors language. CLP is suitable to be used in meeting scheduling which is a combinatorial problem. Its declarative programmability is more amenable to changes.

1 Introduction

Commercial calendars such as Outlook are not flexible enough to adapt to different working environments. On the other hand, building custom software is often costly and does not cater for frequent changes in corporate management.

In this paper, we present a meeting scheduling system based on *Open Constraint Programming* (*OCP*) paradigm [3]. OCP extends CLP with *reactivity* and *openness*. OCP is a concurrent programming framework which allows special queries that react to internal and external events, hence the reactivity. The queries can be submitted by concurrently running, distributed client programs that can be written in any language, hence the openness. These queries, a.k.a. *reactors*, are written using a simple language with synchronization and timeout capabilities. Our system stands out from the rest due to this *declarative programmability* of scheduling constraints.

The system consist of a constraint store in the form of a constraint logic program, a CLP($\mathcal{R}$) solver, an OCP server, and a web-based CGI user interface. The store contains global and user data, integrity and user constraints and other rules that govern the behavior of the system. The users may send a query by clicking buttons on the user interface. These are translated into OCP reactors and sent to the server. Some of the reactors can be delayed in the server and triggered later when some condition is satisfied.

In the next section we will explain the knowledge base which is a part of the constraint logic program. We will thus elaborate on the triggering mechanism in Section 3. We will describe some related works in Section 4 before concluding our paper in Section 5.

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 792–796, 2002.

2 Knowledge Base

We assume an organization consisting of research groups, as in universities. A research group itself consists of a principal researcher, and co-researchers. A principal researcher often needs to call for a meeting by announcing it to the potential participants. The participants thus reply to the announcement by stating their preferences on the timing. These concepts are captured by the following components of the constraint store:

- **Users.** There are three *roles* in the system, namely *normal users* (or *participants*), *hosts*, and *administrator*. A user assumes at least one of the role, and can have multiple roles at the same time.
- **User preferences.** All the preferences are real numbers between 0 and 1, where 0 represent "blackout" for that slot and 1 shows a free slot. For example:
  `pref(kenny, pingpongmatch, [1.0, 1.0, 0.5, ..., 0.3, 1.0]).`
- **Public resources.** Resources required for the meetings like space and equipment.
- **Public meetings.** Each is defined as having a *title*, *proposed participants*, *required resources*, *deadline of submission*, and *schedule time window*.
- **Bundles.** Bundles are collections of time slots with practical meaning to a user or a group of users, e.g. "all mornings", "every Tuesday afternoon", "lunch time", etc.
- **Optimization rules and integrity constraints.** The optimization rules aim at maximizing the combined weighted preferences of all participants of a meeting, and the integrity constraints ensure that no conflicts such as one resource being used in two locations will occur.

The CLP($\mathcal{R}$) solver is used to solve the *static event scheduling problem:* Given a set of variable constraints on participants, resources and meeting time window, produce a schedule of events that achieve collective satisfaction among all the participants before pre-defined deadlines. When there are a number of events to be scheduled together, this becomes a combinatorial optimization problem. The constraint solving based on CLP($\mathcal{R}$) allows a good, if not optimal, solution.

User-defined constraints and their preferences can be added, removed or changed dynamically. For example, as a user adds more meetings to attend to his calendar, the addition will be automatically translated into changed preferences. Each such change will trigger a constraint solving procedure.

3 User Reactors

OCP is a framework which generalizes constraint stores and their interactions [3]. The essential model consists of a shared constraint store, concurrent programmable agents, which are independent of the store, and a notion of reactor, which provides synchronization and atomicity of operations against the store. A reactor follows this syntax:

wait $\langle cond_1 \rangle \Rightarrow \langle action \rangle$
unless $\langle cond_2 \rangle$

The reactor would be suspended in the OCP server until the *sustain condition* $\langle cond_1 \rangle$ becomes true and remains true during the execution of $\langle action \rangle$. At any point, any query or suspended query will be aborted when the *timeout condition* $\langle cond_2 \rangle$ is satisfied.

Next we introduce the workings of the system by using sample reactors.

User's calendar consists of time slots. Priority states how much the user prefers to attend that meeting in the time slot. The reverse of this priority which we call *preference* is defined as $preference(time, user) = 1 - priority(time, user)/10$. The following is the reactor triggered if a user is invited to a meeting:

wait `invited`$(meeting, participants), user \in participants$
$\Rightarrow$ `update_pref`$(user, preferences)$, `update_cons`$(user, constraints)$
unless `false`

A normal user may manually change their preferences in the calendar, and resubmit it. Some other meetings added to the system may also automatically change his or her preferences. When a tentative schedule has been computed, it is marked on the user's calendar. We have the reactors:

wait `true`
$\Rightarrow$ `set_pref`$(user, preferences)$, `set_cons`$(user, constraints)$
unless `false`

wait `tentative_schedule`$(meeting, time)$
$\Rightarrow$ `mark_calendar`$(meeting, time)$
unless `deadline`$(meeting, d), clock?d$

The host is a special user proposing a meeting:

wait `true` $\Rightarrow$ `propose`$(meeting, timewindow, participants, resources)$
unless `false`

The host has the right to favor certain participants by giving their preferences higher priority, so that their weighted preferences are higher.

After all participants have submitted their preferences and constraints, the solver will present to the host the average preferences of all the time slot within the time window, along with the best solutions, in the form of a preferences table. The host may thus choose one of the time slot for the meeting. The host also reserves the right to cancel the meeting if it is no longer needed, or if the schedule produced by the system is not satisfactory. However, the host may not tamper with other user's preferences to make a particular time slot more favorable. As user preferences change over time, the server computes new average preferences. The host thus may change the meeting's schedule accordingly. The host can also submit constraints to bias the solutions computed by the server since the system is very flexible to cope with changes in such requirements. By entering different constraints using the user interface, the behavior of the system can be altered easily.

The following reactor is used by the host to notify the participants of a new solution to meeting schedule:

wait `new_schedule`$(meetings, times), meeting \in meetings, \ time \in times$
$\Rightarrow$ `tentative_schedule`$(meeting, time)$
unless `false`

The administrator is in charge of scheduling for all meetings whenever any user preferences, constraints or required resources are updated. The following reactor sustains on such changes:

wait (`change_resource`(r), $r \in resources$;
`change_preferences`$(user,\ preferences)$, $user \in participants$);
`change_constraints`$(user,\ constraints)$, $user \in participants$)
$\Rightarrow$ `re_schedule`$(open\_meetings,\ times)$
unless `deadline`$(meeting, d)$, $meeting \in open\_meetings$, $clock?d$
The administrator also controls the availability of public resources:
wait `true` $\Rightarrow$ `update_resource`$(resource, newschedule)$
unless `false`

4 Related Work

In the paper [6], the authors stated that 88% of the users of calendar system said that they use calendars to schedule meetings. This justifies an integrated approach to calendar and meeting scheduling system. As in our system, in [1] the meeting scheduling system is integrated with users' calendar. User specifies the events that they want to attend, and the priority of the event. Users calendars are combined to find an appropriate time slot for a meeting. However, negotiation is done manually through exchanging of email messages. Our system can be seen as providing additional automated negotiation feature.

The paper [7] describes a formal framework for analyzing distributed meeting scheduling systems in terms of quantitative predictions. The aspects analyzed are announcement strategies, bidding strategies and commitment strategies. Interesting analyses are presented in the paper, including probability of time slots being available.

Groupware toolkits [2,5,4] provide APIs for building groupware applications. The APIs often provide user configurability as a limited fashion to cope with changes in user requirements. However, toolkits are intended solely to be used by the developers, but our system is intended to be programmable by the users as well as developers.

5 Conclusion

In this paper we have presented a meeting scheduling system. The system consists of a single open constraint programming (OCP) server and a number of client calendars UI. The OCP server provides synchronization, concurrency control, and a knowledge base. The constraint solving capability at the server aims at maximizing user satisfaction in terms of preferences. We have presented a set of reactors functioning to schedule meetings that can be submitted from the client UI. In future, more advanced topics such as scalability, fault tolerance and communication efficiency, as well as actual usability of the system will be addressed.

References

1. D. Beard, M. Palaniappan, A. Humm, D. Banks, A. Nair, and Y.-P. Shan. A visual calendar for scheduling group meetings. In *Proceedings of the CSCW '90 Conference on Computer-Supported Cooperative Work*, pages 279–290. ACM Press, 1990. 795
2. S. Greenberg and M. Roseman. Groupware toolkit for synchronous work. In M. Beaudoin-Lafon, editor, *Computer-Supported Cooperative Work*, volume 7 of *Trends in Software*, chapter 6, pages 135–168. John Wiley & Sons, Ltd, 1999. 795
3. J. Jaffar and R. H. C. Yap. Open constraint programming. In *Principles and Practice of Constraint Programming – CP'98, 4th International Conference, Pisa, Italy*, volume 1520 of *Lecture Notes in Computer Science*, page 1. Springer, 1998. 792, 793
4. Lotus Corp. *Lotus Notes Developer's Guide Version 4.0.* Cambridge, MA, USA, 1995. 795
5. I. Marsic. A framework for multimodal collaboration in heterogeneous environments. *ACM Computing Surveys*, (4), June 1999. 795
6. L. Palen. Social, individual, and technological issues for groupware calendar systems. In *Proceedings of the CHI '99 Conference on Human Factors in Computing Systems*, pages 17–24. ACM Press, 1999. 795
7. S. Sen and E. H. Durfee. A formal study of distributed meeting scheduling. *Group Decision and Negotiation*, 7:265–289, 1998. 795