An ontology for publishing and scheduling events and the lessons learned in developing it.

Craig Sayers
Hewlett Packard Labs, Palo Alto, California
craig_sayers@hp.com

Reed Letsinger
Hewlett Packard Labs, Palo Alto, California
reed_letsinger@hp.com

ABSTRACT

This paper provides a brief overview of an ontology for publishing and scheduling events. It summarizes our experiences using the FIPA SL0 language in a Jade environment, and aims to capture some practical lessons for future ontology development.

The described ontology supports the preferences and uncertainty which are a natural consequence of holding events with real people, in a real world. It represents the relationships between event times, places, and people; handles events that are physically or temporally distributed; and serves to describe events during all stages of scheduling, starting from an initial imprecise specification, and ending with an exact event description suitable for publication.

For future ontology development we argue for placing the burden on content producers, rather than consumers; for maintaining a single consistent representation; and for pushing localization and presentation issues to the edges of the network. In addition, we recommend exploiting hierarchies in the information (particularly with recursion) and allowing for the representation of incomplete knowledge.

1. INTRODUCTION

During the winter of 2001, we began work on an agent-based meeting management system. The system, which we called CoolAgent [6], is organized as a distributed collection of software agents and services implemented within the JADE [1] development environment. The agents negotiate with each other to determine times and places best suited to the needs of both the meeting organizer and the meeting participants. For this negotiation to work, the software agents must be able to exchange messages containing information about meetings.

A simple description of a meeting would collect together answers to questions like: Where will the meeting be held? What time does it start? Who is invited? What is the purpose? This is the approach taken by conventional calendaring systems. See for example: iCalendar [2], iTip [9] and two early Internet drafts: draft-dawson-ical-xml-dtd-01 [3] and draft-reddy-xml-ical-00 [7].

For our application, simply collecting a set of facts into a flat structure is not sufficient. It does not allow us to to capture the nuances of real-world interactions. For example, a quite reasonable meeting request of a human would be:

> "I'd like to arrange a meeting with you in Cupertino on Thursday morning, or in Palo Alto on Friday afternoon"

This requires handling the dependencies between times and places. For example, an acceptable solution to that event request would not be a meeting in Cupertino on Friday afternoon. Replies to meeting requests can be equally difficult to represent. For example, consider a reply:

> "I could meet at any time on Thursday, but would prefer sometime between 10 and 11".

Handling that in a software agent environment requires representing preferences and uncertainty, which are a natural consequence of holding events with real people, in a real world. To satisfy those requirements, we needed to represent the relationships between event times, places, and people; we needed to represent events that are physically or temporally distributed; and we needed to represent events during different stages of scheduling, starting from an initial imprecise description, and ending with an exact event suitable for publication.

In the remainder of the paper we'll describe the representation of actions (section 2.1) and events (section 2.2) before presenting some general lessons for future development (section 3).

2. MESSAGES

In our system, agents communicate by sending messages expressed in the Foundation for Intelligent Physical Agents (FIPA) Agent Communication Language (ACL) [4]. Each ACL message is part of a conversation. ACL messages include the names of the sending and receiving agents, an identifier of the conversation the message belongs to, a performative, the message content, and the names of the language and ontology used to encode that content. Performatives are used to specify what role the message is playing in a conversation, e.g. a question, a request for action, or a response to a request. The FIPA standard specifies the set of possible performatives, which are all domain-independent.

The domain-dependent semantics of the message are encoded using a content language and a domain-specific vocabulary. We chose the FIPA-defined content language SL0 [5], and then defined the CoolAgent ontology to provide the vocabulary that can appear in SL0 sentences. SL0 is a very simple language that provides the means to describe particular objects and actions, and to express ground, atomic facts. No logical connectives, such as “and”, “or”, “not”, “for all”, come as part of the language. If there is need to express some level of logical complexity, the ontology must take on the responsibility.

In general, a CoolAgent conversation consists of a series of request/inform pairs. One agent requests another to perform an action on an event, and the agent later responds, informing the requester of the result of that action. The Coolagent ontology [8] defines those actions and events.

2.1 Actions

In SL0, an action specification is a verb describing some domain-specific activity. In CoolAgent, the most common actions performed on an event are: publish, inquire, schedule and cancel. The interpretation of these verbs is similar to the pre-existing iCalendar specification.

In constructing conversations it simplified the ontology to overload the actions by making their interpretation dependent on the identity of the receiver, rather than just the message content. So, for example, when we ask a personal agent to “schedule” it is interpreted to mean “schedule a person to attend this meeting”; while when the same message is sent to a room broker agent, it is interpreted to mean “schedule a conference room for this meeting”. This framework was particularly helpful in allowing code reuse among the agents.

2.2 Events

The ontology for events is built on a single event object. Within each event object there are expressions for different instances of that event; where each instance is a particular combination of time, place and people. An event description contains three main components:

- Event - the highest level in the hierarchy, capturing details that are relevant for the whole event. It includes a unique id number, a summary, the name of the organizer, and an expression for the invitees. It also includes an expression for the different instances of this event.
- Instance - describes a particular combination of time, place, and people. It includes expressions for the start-time, duration, location, and attendees.
- Place - describes a particular location (for example, a conference room). It includes the address, phone number and a list of available features (for example, a whiteboard, or an overhead projector).

In the simplest case, the event contains only a single instance, and the location of that instance is only a single place. In practice, we need to represent events which are much more complex. That requires a representation for expressions.

For example, to represent an event that takes place in several times/places, we use an all-of relationship, so the single instance is replaced by an expression:

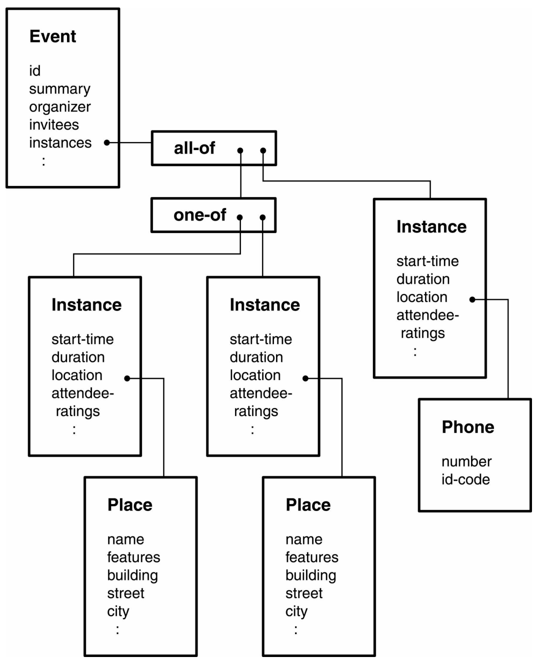

Figure 1: An example of an event described using the ontology. In this case, the event will take place in one of two physical locations and will also have a call-in number.

all-of(instance1, instance2, . . . instanceN)

This has the force of a conjunction: the event has instance1, and the event has instance2, and so on.

To represent the choices during the planning stages of an event, we have the additional expressions:

one-of(instance1, instance2, . . . instanceN)
any-of(instance, instance2, . . . instanceN)

A one-of expression captures the force of an exclusive-or, while an any-of expression has the semantic effect of an inclusive-or. All three types of expressions may be combined and nested to arbitrary levels.

An example of an event using expressions is shown in Figure 1. This example is typical of an event during scheduling. Similar expressions were used to present alternatives for people and places. An example message is shown in Figure 2

This hierarchical structure proved helpful in separating event-specific from instance-specific information. The higher-level event object captures information which is never open to negotiation, and which is defined only once for each event, while the lower-level objects capture information specific to a particular instance which may be under consideration. Not all meetings may be represented in a pure hierarchy. In particular, we can not directly encode the knowledge that two different instances must occur simultaneously. However, since we know that is usually the case, that knowledge can be hard-coded into the message interpreter.

```
(REQUEST
 :sender ( agent-identifier:name gateway)
 :receiver (set( agent-identifier:name griss))
 :content
  ((action
     (agent-identifier :name griss)
     (schedule-event
        (CoolEvent
         :type meeting
         :summary "Coolagent discussion"
         :invitees
            (any-of
               (set
                  (agent-identifier :name griss)
                  (agent-identifier :name letsinge)
                  (agent-identifier :name wymore)))
         :instances
             (CoolInstance
               :start
                   (range-of
                     (CoolDate "20010504T150000000Z")
                     (CoolDate "20010505T000000000Z"))
               :duration 3600
               :location(CoolPlace)
               :expected-attendee-count 5
              :attendee-ratings(set)
              :attendee-probabilities(set))))))
 :reply-with  gateway.Thread-42.988050547328
 :language  FIPA-SL0
 :ontology  CoolAgent
)
```

Figure 2: An example message requesting that an event be scheduled. This is an incoming message to the personal agent "griss" asking it to schedule a meeting with the specific people represented by the other invitee agents. The meeting requires a physical location and it is estimated that 5 people will attend (even though only three were personally invited). This example also includes an additional "range-of" expression which serves to replace what would otherwise be a huge "one-of" expression.

As the event description is refined during scheduling, there will often be a choice of time or location. To decide among those, we take into account the preferences of each invitee, allowing them to tag each instance with both a probability of attendance and a rating.

The rating allows the expression of subtle preferences which are typically ignored when scheduling meetings by hand. For example, if you were to manually schedule a meeting with six other people, the burden of personally coordinating the schedules of six people and finding a conference room is so high that you would likely be unwilling to accept any additional constraints – especially if they were small personal preferences.

In cases where there are a range of options, the recipient can break a single instance into multiple instances in order to rate each differently. For example, if the start time was specified as a range from 9:00-12:00, a recipient could break that into two instances: 9:00-10:00 and 10:00-12:00 and rate each separately.

3. LESSONS FOR FUTURE WORK

While the CoolAgent ontology was focused on event scheduling, many of the lessons are applicable to other ontology development.

Choose an appropriate language. When choosing a language there is a tradeoff between expressiveness and complexity. With a simple language, the ontology must assume responsibility for adding expressiveness. These custom expressions require effort to implement, but they can simplify the work for agents interpreting messages since the expressiveness is limited to exactly the places it is needed. A more complex language allows for a simpler ontology, but may require more work in each agent which must interpret messages.

In our case, we found the SL0 language a pragmatic and convenient choice. At the time, it was the only language supported by Jade, and we could add just enough logical complexity to serve our needs without needing to unduly burden each agent. However, if our project had been much larger, we would likely have run into problems. SL0 has no formal language for describing constraints on the ontology, so one must write custom code to verify that the message content is valid. In our case that proved not to be a large burden (due partly to some built-in ontology assistance in Jade, and partly to code re-use among many of the agents).

Use existing standards and not existing formats. This might seem obvious, but it is very tempting to simply choose an ontology representation which matches the existing data format, regardless of how obscure that might be. The short-term advantages of such an approach are rarely worthwhile.

Place the burden on content producers. For most fragments of content there will be one producer and many consumers. So, given a choice, push work onto the single content producer to simplify the work for all the content consumers. In particular, that means we should force producers to put data into a standard format.

Maintain a single consistent representation. It is particularly beneficial to store any one piece of information using only one representation. This simplifies the ontology, and makes less work for code receiving messages in that ontology. Interestingly, there are a number of standards that don't enforce this. That shows an unfortunate lack of fortitude by their designers. At design time, in cases where one is not sure, or where a committee can not agree, its always easiest to simply allow several variants. But that is costly later, since now every consumer needs to accept every possible variant. We would strongly urge a move to only supporting a single canonical representation.

In the CoolAgent project we found it particularly helpful to store event information in a single representation through all stages of scheduling and publishing. This meant we could re-use code for processing events in many of the agents. In many cases, agents could delegate tasks by simply forwarding the content of the message they had received. It also had the more subtle benefit of simplifying debugging by allowing a developer who understood any one agent to have at least some understanding of the messages passed between other agents.

Localize at display time. It is impossible to predict all the potential consumers. So always store information in a consistent globally acceptable form. For example, store measurements using SI units and store dates/times relative to UTC, rather than in some local timezone. Localization must be the responsibility of the information consumer, since only it knows the preferences of the human to which the information is being displayed.

In our implementation we ran into at least half-a-dozen different date/time representations as we interfaced with different legacy systems. We also ran into a number of bugs (especially when we started to schedule teleconferencing calls across timezones during a switch to daylight-savings-time). The choice of UTC format for the dates/times within messages provided valuable insulation between the different pieces of the system.

Exploit hierarchies. When developing ontologies, its tempting to adopt a flat structure, but that tends not to scale well later. In our case, the inter-dependency between times, places and attendees meant that a simple flat structure would have proved unmanageable. So, look for potential hierarchies early, and use them (even if initially, there is only one option at some levels). Be especially careful to look for potential recursive definitions as these invariably end up being more useful than their designers imagined. In our case, the use of expressions proved particularly helpful and made for elegantly simple implementations.

Consider hard-coding some information. While it is usually possible to construct an ontology to represent all details, that is not always necessary. In our case, we were able to make do with a simpler ontology by hard-coding some information in the agents (for example, the knowledge that given a choice of times, the different instances of each event should occur simultaneously). We further simplified the ontology by using the identity of the receiving agent when decoding each incoming message.

Allow for incomplete knowledge. It is tempting to assume that all facts represented in the ontology are completely specified. Such an assumption simplifies things greatly, but it also limits the usefulness for conversing about a subject to decide on those facts. For example, making an ontology to represent the list of people who attended a meeting is much easier than making one for use in conversations to decide who can/should attend that meeting.

More information is not always better. It is tempting to define messages by making an exhaustive list of everything which might one day be helpful to a recipient. However, this neglects the important matter of privacy. In the Coolagent ontology, where we were exchanging information about employee meeting preferences, it was beneficial to not disclose too much information. For example, when providing meeting availability information, we stored only a single floating-point number, representing the individual's probability of attending each instance. This was sufficient for scheduling, without giving away too much personal information about the reasons behind the user's preferences.

4. CONCLUSION

We have described an ontology for scheduling and publishing meetings and events. It uses a single consistent representation with a hierarchical structure and is able to capture some of the uncertainty and complexity which occur in real-world systems.

This work represents one approach to the trade-off between logical complexity in the ontology and the content language. It avoided the need to have sophisticated reasoning abilities in our agents, while providing sufficient expressive power for our needs. The ontology was simplified by hard-coding some information in the agents, and by overloading actions.

In developing a new ontology, it is always tempting to simply copy the semantics of the first piece of pre-existing data one needs to encode. That is usually a poor choice. It is equally tempting to allow more than one representation for the same piece of data. Unfortunately, the more latitude one gives content producers, the more effort is needed by the much larger group of content consumers. We urge developers to adopt a single canonical representation, pushing issues of presentation or localization out to the edges of the network.

5. ACKNOWLEDGMENTS

The Coolagent project was a collaborative effort spanning several departments. We are grateful to all those who contributed, and to our management for supporting this work. Thanks also to all the contributors to the JADE system.

6. REFERENCES

[1] F. Bellifemine, G. Caire, T. Trucco, and G. Rimassa. *JADE Programmers Guide*. CSELT S.p.A., http://jade.cselt.it, 2001.

[2] F. Dawson et al. *iCalendar Message-based Interoperability Protocol - iMIP*. RFC 2445, November 1998.

[3] F. Dawson et al. *iCalendar XML DTD, Internet draft draft-dawson-ical-xml-dtd-01*, December 1998.

[4] Foundation for Intelligent Physical Agents, http://www.fipa.org. *FIPA ACL Message Structure Specification, document XC00061D*, 2000.

[5] Foundation for Intelligent Physical Agents, http://www.fipa.org. *FIPA SL Content Language Specification, document XC00008D*, 2000.

[6] M. Griss et al. Coolagent: Intelligent digital assistants for mobile professionals - phase 1 retrospective. Technical Report HPL-2002-55 (Revision 1), Hewlett Packard Laboratories, Palo Alto, California, 2002.

[7] L. Lippert et al. *iCal in XML, Internet draft draft-reddy-xml-ical-00*, November 1998.

[8] C. Sayers and R. Letsinger. The coolagent ontology: a language for publishing and scheduling events. Technical Report HPL-2001-194, Hewlett Packard Laboratories, Palo Alto, California, 2001.

[9] S. Silverberg et al. *iCalendar Transport-independent Interoperability Protocol - iTIP*. RFC 2446, November 1998.