APROCO
A Programmable Coordination Medium
First Results

Daniel Kühni*

*Software Composition Group, Institute for Computer Science and Applied Mathematics, University of Berne, Neubrückstrasse 10, CH-3012 Bern, dkuehni@iam.unibe.ch http://www.iam.unibe.ch/~dkuehni/aproco.html

Overview of the Talk

1. Problem description
2. Goals
3. Architecture and parts of APROCO
4. Dynamics
5. Results
6. Further Work

Problem Description

Inter-agent coordination is a complex problem in open distributed systems:

1. agents can enter and leave the configuration at any time
2. coordination problems affect more than one agent

Separation of concerns:

1. separation of coordination and computation and
2. factoring out the coordination aspects into components

increases the potential for reuse.

Coordination is often hard-coded into the participants.

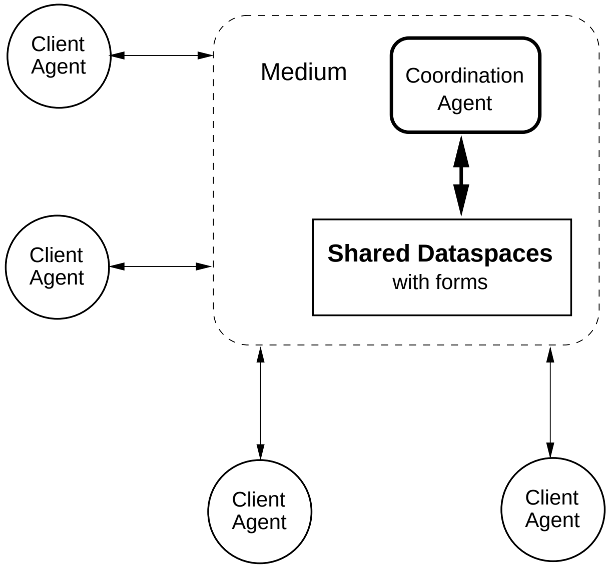
Standard object-oriented techniques are not well suited to cope with multi-object coordination (Aksit & Bergmans, 1992).

Goals

- Encapsulate flexible coordination abstractions into reusable entities.
- Find a useful set of such abstractions.
- Getting the coordination out of the participants as much as possible.
- Find the limits of the model of agents coordinating through the exchange of forms using real-world application requirements.

Architecture of APROCO

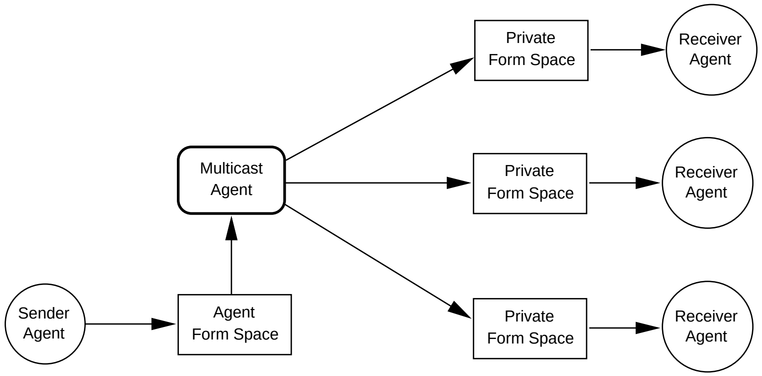
We propose an architecture to solve coordination problems based on coordination agents and shared dataspaces with forms.



- Multiple Linda-like shared dataspaces (generative communication style)
- Coordination agents provide coordination services
- Forms are the basic communication containers

Simple Example: Multicast

We take a very simple example to show the different parts of the coordination medium.

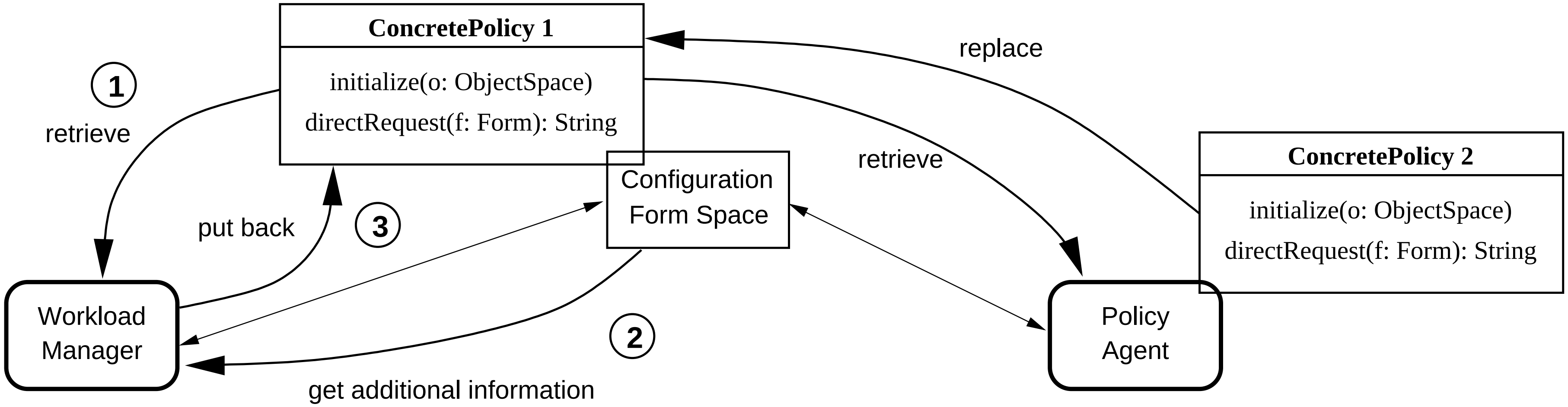


The example shows a situation where one agent e.g. has to notify other agents about changes in its state (similar to an observer situation).

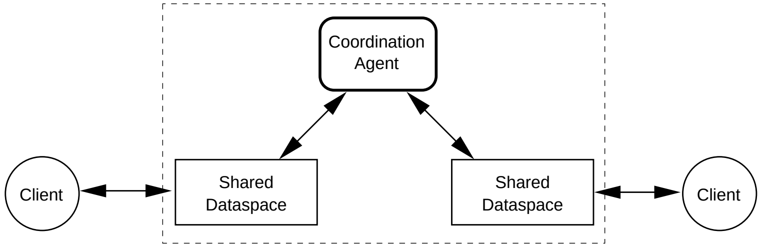
Coordination Agents

- Pro-active encapsulations of coordination abstractions.
- Provide coordination "services" transparently to the clients.

Example of pro-active behavior: dynamic exchange of policy



Multiple Shared Dataspaces



Uniform data access for client agents and for coordination agents while enabling the coordination agents to control traffic.

Homogeneous component model for large systems: nested hierarchies of shared dataspaces.

But: degradation of anonymity of communication!

Private Dataspaces

Private dataspaces are shared dataspaces that are only accessible for one client and the special agents but not for other clients.

- Liveness: all interested client agents can eventually read a specific dataspace item.
- Atomicity: no interference between clients retrieving multiple dataspace items at the same time.[1]
- Efficiency: blocking wait instead of polling for changed items.
- Independence: every client can proceed in its own pace.

[1] There is no way to atomically retrieve more than one dataspace item in standard Linda

Forms

Forms are based on ideas found in incrementally “grown” systems like electronic mail or HTML where new features are added using new tags.

This supports **extensibility** because new functionality can be added without affecting the existent behavior.

Example: server with different generations of clients.
Using forms the server can treat all requests identically, because the older clients just ignore the extra result tags without breaking.

Configuration Dataspaces and Access Rights

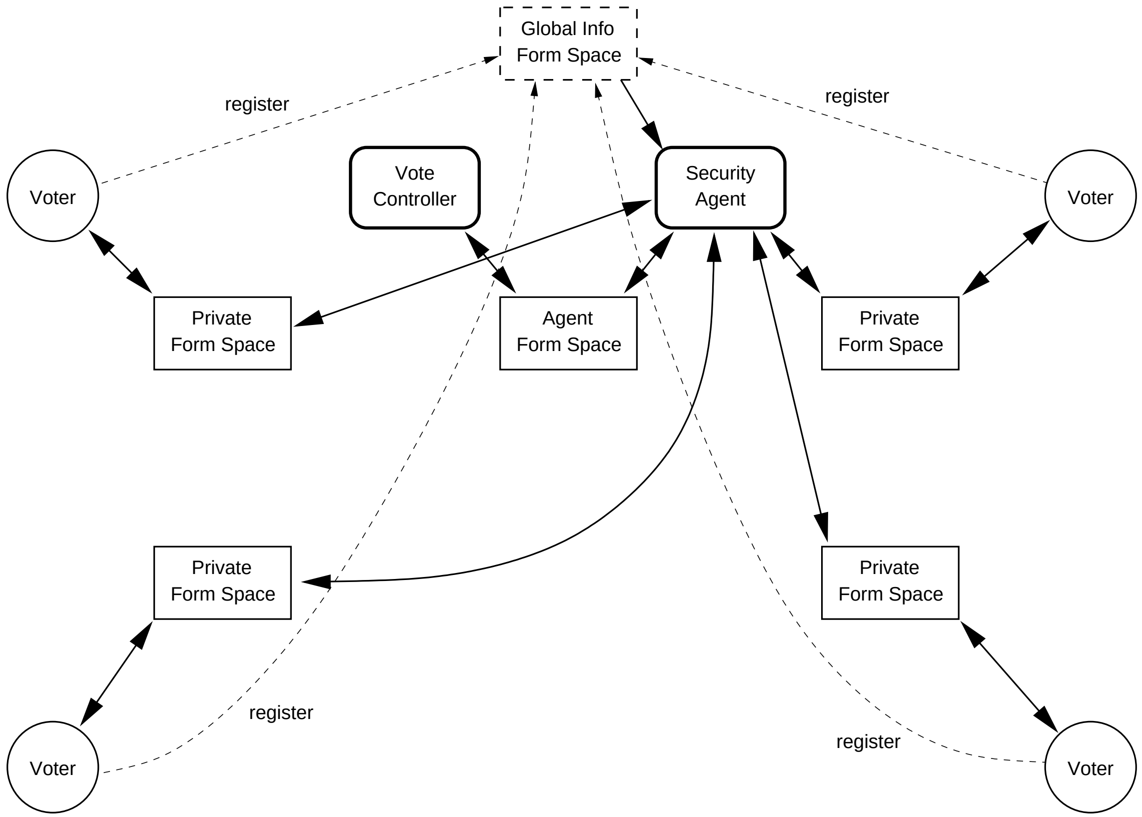
At least one globally accessible configuration dataspace needed, e.g. to

- keep track of the participants
- store application dependent information for coordination agents

Access to a shared dataspace either statically at compile-time or dynamically through the acquisition of a *dataspace handle* at run-time.

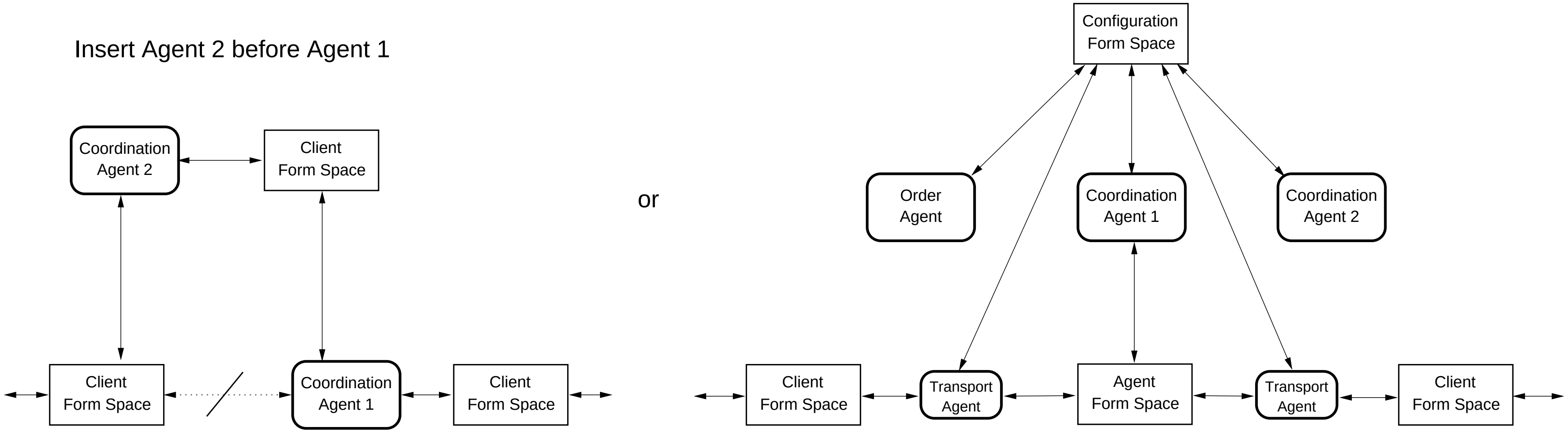
Access rights are a central property of the approach. Without different access rights it is not possible to ensure security.

Example: Electronic Vote



Dynamics

The medium is configurable: new coordination agents can be introduced at any time.[2] This requires means for defining the relative order of the coordination agents in the information processing.



[2]There is no special treatment of ongoing activities at the moment.

Results

List of useful coordination agents so far:

- Transport
- Multicast
- State Proxy
- Security
- Vote Controller
- Fault Tolerance
- Policy
- Workload Manager
- Client Information

Meta-level coordination agent: Order Agent

Multiple dataspaces with access rights needed.

Participants protocols easier and clearer than with standard Linda.

Open, dynamic, flexible, extensible architecture proposed.

Further Work and Plans

- Form Algebra: formal groundwork for forms and their relation to tuples.
- Justify the "programmable" in the title: create an easy way to change the configuration of the medium (configuration language).
- Case study: take the conference management example to show the applicability of the approach for bigger systems (building composites).
- Describe the kind of problems that can't be solved with this architecture.