Engineering Norm-Governed Systems

Part I: Introduction to Norm-Governed Systems

Alexander Artikis[1,2] and Jeremy Pitt[1]

[1]Imperial College London, UK
[2]National Centre for Scientific Research "Demokritos", Greece

Objectives and Outcomes

- Objectives
  - To motivate, explain and demonstrate the use of Action Languages from Artificial Intelligence for engineering norm-governed multi-agent systems.
- Outcomes – tutorial attendees will:
  - Understand the basic concepts in norm-governed systems: permission, obligation and (institutional) power;
  - Apply these concepts in the (executable) specification of protocols using Action Languages from Artificial Intelligence;
  - Manipulate these specifications for animation of, reasoning about, and proving properties of such systems;
  - Understand how adaptive specifications can be used for engineering self-organisation, and other self-* properties.

Tutorial Structure

- Part I: Introduction to Norm-Governed Systems
- Part II: Action Languages 1 – Event Calculus & Voting Protocol
- Part III: Action Languages 2 – Action Language $C+$ & Resource Sharing Protocol
- Part IV: Dynamic Specifications
- Part V: Run-Time Configurations
- Part VI: Open Issues

Engineering Norm-Governed Systems: Part I

- Motivation
- Norm-Governed Systems
- Organised Adaptation
- Action Languages
- Examples

Motivation

- Pick a network:
  - individual people, forming online communities or social networks via computer-mediated communication
  - computing devices, forming ad hoc networks, MANETs, VANETs, Sensor Networks, etc.
  - business processes, forming virtual enterprises/organizations, holonic manufacturing, computational economies, etc.
- May be classified as *open systems* (in the sense of Hewitt)
  - autonomous components of heterogeneous provenance
  - can assume that components can communicate (i.e. a common language)
  - can *not* assume a common objective or central controller

Features

- Common features of open systems:
  - Dynamic and 'volatile': the environment, network topology and constituent nodes can vary rapidly and unpredictably
  - 'Evolutionary': known nodes can come/go, but can also have new nodes and node 'death'
  - Co-dependence and internal competition: nodes need others to satisfy their own requirements, but may also behave to maximise individual (rather than collective) utility
  - Partial knowledge: no single source knowledge, union of knowledge may be inconsistent
  - Sub-ideal operation: the nodes may fail to comply according to the system specification, by accident, necessity, or design.
- Actuality (what is the case) and ideality (what ought to be the case) do not necessarily coincide

Addressing the Features

- Distributed functionality and co-dependence
  - requires collective and coordinated interaction
  - requires role-based 'co-operative work'
- 'Evolution'
  - requires resilience to (unexpected) change, and scope for improvement
- Decentralised control and partial knowledge
  - requires sub-group decision-making
- Dynamic topology and asynchronous communications
  - too fast, too frequent and too complex for operator intervention
- Unpredictable behaviour and sub-ideal operation
  - requires conflict resolution (restoration of compliant states)
- Everything requires agreed rules and well-defined procedures, and rules and procedures for *changing* rules and procedures

Proposal (1): Norm-governed Systems

- Encapsulated by the idea of *institution*
- Institutions (Ostrom)
  - Sets of working rules used to determine who is eligible to make decisions in what area, what actions are allowed or constrained, what aggregation rules are used, etc.
- As Normative Systems:
  - Agents
  - Social constraints/rules
    - Communication language
    - Role

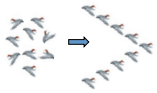
Social Constraints

- Three types of 'can'
  - Physical capability
  - Institutional power
    - The performance by a designated agent, occupying a specific role, of a certain action, which has conventional significance, in the context of an institution
    - A special kind of 'certain action' is the speech act
  - Permission (& obligation)
  - Can have (physical or institutional) power with/without permission
  - Sometimes power implies permission
- Sanctions and enforcement policies
- Right, duty, entitlement, and other more complex relations
- Social constraints can be adapted for intentional, run-time modification of the institution

Proposal (2): Organised Adaptation

- Organised Adaptation vs. Emergence
  - Emergent Adaptation:
    - the non-introspective application ...
    - of hard-wired local computations, ...
    - with respect to physical rules and/or the environment, ...
    - which achieve unintended or unknown global outcomes
  - ... as opposed to ...
  - Organised Adaptation:
    - the introspective application ...
    - of soft-wired local computations, ...
    - with respect to physical rules, the environment and conventional rules, ...
    - in order to achieve intended and coordinated global outcomes.

Examples of Organised Adaptation ('Real' Life)



Emergence

Coordination

Organizational Adaptation

Role Assignment
(White smoke from the Sistene Chapel of St. Peter's Basilica announces a new Pope)

Legal Proclamation

Achieving Intended Global Outcomes

- Organised Adaptation affects coordination, organization, institution and norms
  - coordination: the collective ability of heterogeneous and autonomous components to CHANGE THE ARRANGEMENT OR SYNCHRONISATION OF the performance of specified actions in sequential or temporal order
  - organization: THE ABILITY TO MODIFY a formal structure supporting or producing intentional forms of coordination
  - institution: ROLE-ASSIGNMENT in an organization where (inter alia) the performance of designated actions to produce conventional outcomes can by DIFFERENT empowered agents OVER TIME
  - norms: standards or patterns of behaviour in an institution CHANGED by decree, agreement etc.,

Engineering (Adaptable) Norm-Governed Systems: Action Languages

- Action languages:
  - representation of the agents' actions and their effects
  - exhibit a formal semantics
  - exhibit a declarative semantics
  - have direct routes to implementation
- Examples:
  - Situation Calculus
  - Event Calculus
  - E
  - $C+$
- We will use the Event Calculus and $C+$
- Other formalisms may be (have been) used for engineering norm-governed systems

Norm-Governed Systems: Examples

- Voting Protocol
  - There is a set of agents S, a subset of which occupy the role of *voters* who are entitled to vote, and a designated agent in S occupying the role of *returning officer*, who declares the result of a vote. The protocol stipulates that the officer calls for a ballot on a specific motion, the voters cast their votes (express their preference), the officer counts the votes and declares the result according to the standing rules.
- Resource Sharing Protocol
  - There is a set of agents S, a subset of which occupy the role of *subjects* who are entitled to access a resource, and a designated agent in S occupying the role of *chair*. The subjects are empowered to request access to the resource, the chair is empowered to grant or revoke access.The protocol stipulates that one or more subjects request access to the resource, the chair grants access to only one; that agent uses the resource until it is released it or access is revoked, and the cycle repeats.

Part I: Summary

- Motivated general requirement for adaptive institutions as approach to issues in open systems
- Proposed Norm-governed systems and Organised Adaptation as framework for formal approach
- Identified Action Languages from Artificial Intelligence as the basis for engineering norm-governed systems
  - representation of the agents' actions and their effects
  - exhibit a formal semantics
  - exhibit a declarative semantics
  - have direct routes to implementation
- Will now be demonstrated
  - Part II: Event Calculus and the Voting Protocol
  - Part III: Action Language $C+$ and the Resource-Sharing Protocol
  - Part IV: Dynamic Specifications
  - Part V: Run-Time Configurations
  - Part VI: Open Issues

Engineering Norm-Governed Systems

Part II: Specifying and Executing Norm-Governed Systems in the Event Calculus

Alexander Artikis and Jeremy Pitt

Outline

- The Event Calculus (EC) (with thanks to Prof. Marek Sergot, Imperial College London).
- Running example: a voting protocol.
  - Specification in Event Calculus of:
    - Institutional Power.
    - Permission & Obligation.
    - Sanction.
- EC implementation routes.
- Voting protocol execution.

The Event Calculus (EC)

- General purpose language for representing events, and for reasoning about effects of events.
- An action language with a logical semantics. Therefore, there are links to:
  - Implementation directly in Prolog.
  - Implementation in other programming languages.
- Prolog:
  - specification is its own implementation;
  - hence executable specification.

Fluents and Events

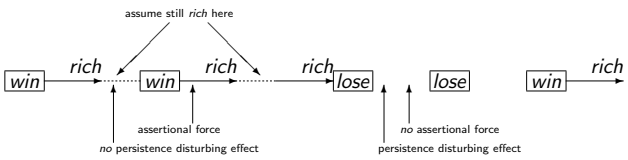
- Focus on events rather than situations; local states rather than global states
- Fluents
  - A fluent is a proposition whose value changes over time
  - A local state is a period of time during which a fluent holds continuously
- Events
  - *initiate* and *terminate* ...
  - ... a period of time during which a fluent holds continuously
- Example
  - $give(X, obj, Y)$ initiates $has(Y, obj)$
  - $give(X, obj, Y)$ terminates $has(X, obj)$
- A sequence of such events forms a narrative

Simplified Event Calculus

- Inertial fluents hold their values continuously
  - Values are assigned initially (at the start),
  - Values are given when asserted (initiated)
  - Values persist until disturbed (terminated)
  - Otherwise we have 'missing information'
- A formula of the form
  - *Event* terminates *fluent*
  - Has persistence disturbing effect, but no assertional force
- A formula of the form
  - *Event* initiates *fluent*
  - Has assertional force, but no persistence disturbing effect
- Suppose
  - *win_lottery* initiates *rich*
  - *lose_wallet* terminates *rich*

Example

- Given
  - *win_lottery* initiates *rich*
  - Winning the lottery initiates rich (but you might be rich already)
  - *lose_wallet* terminates *rich*
  - Losing your wallet terminates rich (but you might not be rich when you lose it)



Events and Narratives in the Simplified EC

- Events occur at specific times (when they 'happen')
  - Assume that all events are instantaneous
  - Aside: there is a refinement of EC for events which have duration
- Here, we will use non-negative integer time-points
  - Does not mean we assume that time is discrete
  - Does not mean that time points have to be integers
  - We only need a relative/partial ordering for events
  - For non-negative integers, $<$ will do
  - Read $<$ as 'earlier than' or 'before'
- A set of events, each with a given time, is called a *narrative*
  - Inference in the SEC is non-monotonic
  - Events in a narrative can be processed in a different order to that in which they occurred

General Formulation

- The narrative (what happens when) is represented by:
  - initially F
    - Fluent F holds at the initial time point (usually 0)
  - E happensat T
    - Event/action of type E occurred/happened at time T
- The effects of actions are represented by:
  - E initiates F at T
    - The occurrence of event of type E at time T starts a period of time for which fluent F holds
  - E terminates F at T
    - The occurrence of event of type E at time T ends a period of time for which fluent F holds
- The general query:
  - F holdsat T
    - Fluent F holds at time T
  - F holdsfor P
    - Fluent F holds for time period P (P is of the form $(T_1, T_2]$)

Method of Computation: The EC 'Engine'

F holdsat T $\leftarrow$
  E happensat Te $\wedge$
  $Te < T$ $\wedge$
  E initiates F at Te $\wedge$
  not (F brokenbetween Te and T)

F holdsat T $\leftarrow$
  $0 \leq T$ $\wedge$
  initially F $\wedge$
  not (F brokenbetween 0 and T)

F brokenbetween Te and T $\leftarrow$
  E' happensat Ti $\wedge$
  $Te \leq Ti$ $\wedge$
  $Ti < T$ $\wedge$
  E' terminates F at Ti

Notes

- Time comparisons are strict: therefore a fluent does *not* hold at the time point in which it is initiated
- Negation-as-failure (not(...)) ensures that inferences are non-monotonic
- Action pre-conditions can be expressed as integrity constraints
  - Some actions can't be performed at the same time
  - For example: $give(X, obj, Y) \wedge give(X, obj, Z) \wedge \text{not}(Y = Z)$
  - Every time the narrative changes, query the integrity constraints to check consistency
- A simple extension allows many-valued (as well as boolean) fluents
  - Form is $F = V$
  - For boolean valued fluents, $V \in \{true, false\}$
- There is a difference between:
  - $kill(X)$ initiates $alive(X) = false$ at T
  - $kill(X)$ terminates $alive(X) = true$ at T

Summary: So far

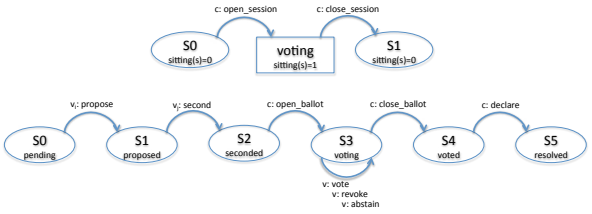
- Introduced (Simplified) Event Calculus
- Events and Fluents
- General computational engine
- Specification of a narrative
- Next: formalisation of a protocol in SEC
- Then: use it for computation

Voting Protocol: Informal Description

- Informal specification of a decision-making procedure according to *Robert's Rules of Order (Newly Revised)*
  - a committee meets and the chair opens a session
  - a committee member requests and is granted the floor
  - that member proposes a motion
  - another member seconds the motion
  - the members debate the motion
  - the chair calls for those in favour to cast their vote
  - the chair calls for those against to cast their vote
  - the motion is carried or not, according to the standing rules of the committee

Voting Protocol: Graphical Description

- Various options for graphical representation
  - UML Sequence diagrams
  - State diagrams



- Note certain simplifications to RONR specification
  - No floor request, debate or agenda
  - Voting, changing of votes etc., concurrently

An Event Calculus Specification

- Basic Items: Events and Fluents
- Institutional Powers
- Voting and Counting Votes
- Permission and Obligation
- Sanctions
- Objection

Actions

Action	Indicating...
$open\_session(Ag, S)$	open and close a session
$close\_session(Ag, S)$	
$propose(Ag, M)$	propose and second a motion
$second(Ag, M)$	
$open\_ballot(Ag, M)$	open and close a ballot
$close\_ballot(Ag, M)$	
$vote(Ag, M, aye)$	vote for or against a motion, abstain or change vote
$vote(Ag, M, nay)$	
$abstain(Ag, M)$	
$revoke(Ag, M)$	
$declare(Ag, M, carried)$	declare the result of a vote
$declare(Ag, M, not\_carried)$	

Fluents

Fluent	Range
$sitting(S)$	boolean
$status(M)$	$\{pending, proposed, seconded$ $voting(T), voted, resolved\}$
$votes(M)$	$N \times N$
$voted(Ag, M)$	$\{nil, aye, nay, abs\}$
$resolutions(S)$	list of motions
$qualifies(Ag, R)$	boolean
$role\_of(Ag, R)$	boolean
$\mathbf{pow}(Ag, Act)$	boolean
$\mathbf{per}(Ag, Act)$	boolean
$\mathbf{obl}(Ag, Act)$	boolean
$sanction(Ag)$	list of integers

Institutional Power

- Recall: an empowered agent performs a designated action in context which creates or changes an institutional fact.
- We want to express the effects of the designated protocol (speech) actions, in particular:
  - *vote*
  - *open_session* and *open_ballot*
  - *declare*
- For the specification of the effects of these actions, it is important to distinguish between:
  - the act of ('successfully') casting a vote, and
  - the act by means of which the casting of the vote is signalled (e.g. sending a message of a particular form via a TCP/IP socket connection).

Institutional Power

- Institutional power to open the ballot on a motion:

$$\mathbf{pow}(C, open\_ballot(C, M)) = true \text{ holdsat } T \leftarrow$$
$$status(M) = seconded \text{ holdsat } T \wedge$$
$$role\_of(C, chair) = true \text{ holdsat } T$$

- Institutional power to cast a vote:

$$\mathbf{pow}(V, vote(V, M, \_)) = true \text{ holdsat } T \leftarrow$$
$$status(M) = voting(\_) \text{ holdsat } T \wedge$$
$$role\_of(V, voter) = true \text{ holdsat } T \wedge$$
$$\text{not } role\_of(V, chair) = true \text{ holdsat } T \wedge$$
$$voted(V, M) = nil \text{ holdsat } T$$

Effects of Institutional Power (1)

- Chair performs $open\_ballot(C, M)$

$$open\_ballot(C, M) \text{ initiates } votes(M) = (0, 0) \text{ at } T \leftarrow$$
$$\mathbf{pow}(C, open\_ballot(C, M)) = true \text{ holdsat } T$$
$$open\_ballot(C, M) \text{ initiates } voted(V, M) = nil \text{ at } T \leftarrow$$
$$\mathbf{pow}(C, open\_ballot(C, M)) = true \text{ holdsat } T \wedge$$
$$role\_of(V, voter) = true \text{ holdsat } T$$
$$open\_ballot(C, M) \text{ initiates } status(M) = voting(T) \text{ at } T \leftarrow$$
$$\mathbf{pow}(C, open\_ballot(C, M)) = true \text{ holdsat } T$$

- Now voters have power to cast votes

Effects of Institutional Power (2)

- Casting and counting votes

$$vote(V, M, aye) \text{ initiates } votes(M) = (F1, A) \text{ at } T \leftarrow$$
$$\mathbf{pow}(V, vote(V, M)) = true \text{ holdsat } T \wedge$$
$$votes(M) = (F, A) \text{ holdsat } T \wedge$$
$$F1 = F + 1$$
$$vote(V, M, aye) \text{ initiates } voted(V, M) = aye \text{ at } T \leftarrow$$
$$\mathbf{pow}(V, vote(V, M, \_)) = true \text{ holdsat } T$$

- Power to revoke vote now granted (revocation without vote was 'meaningless')
- Power also used to advance status of motion, perform role assignment, etc.

Permission

- 'Right' aspect of enfranchisement
  - Agents have the power to vote
  - Agents have the permission to vote
    - In this case (although not always) power implies permission
  - Nobody should stop them from exercising their power
    - Therefore the chair's power to close the ballot is not always permitted

$$\mathbf{pow}(C, close\_ballot(C, M)) = true \text{ holdsat } T \leftarrow$$
$$status(M) = voting \text{ holdsat } T \wedge$$
$$role\_of(C, chair) = true \text{ holdsat } T$$
$$\mathbf{per}(C, close\_ballot(C, M)) = true \text{ holdsat } T \leftarrow$$
$$role\_of(C, chair) = true \text{ holdsat } T \wedge$$
$$status(M) = voting(T') \text{ holdsat } T \wedge T > T' + 10$$

Obligation

- 'Entitlement' aspect of enfranchisement
  - 'Access' to 'voting machine' is a 'physical' issue
  - Correct vote count: as above
  - A 'fair' outcome: obligation to declare the result correctly: e.g. a simple majority vote

$$\begin{aligned}\mathbf{obl}(C, \mathit{declare}(C, M, \mathit{carried})) = \mathit{true}\ \text{holdsat}\ T \leftarrow\\ \mathit{role\_of}(C, \mathit{chair}) = \mathit{true}\ \text{holdsat}\ T\ \wedge\\ \mathit{status}(M) = \mathit{voted}\ \text{holdsat}\ T\ \wedge\\ \mathit{votes}(M) = (F, A)\ \text{holdsat}\ T\ \wedge\\ F > A\end{aligned}$$

Sanction

- The chair always has the power to close a ballot
- It has permission to exercise the power only after some time has elapsed
- If it closes the ballot early, it may be sanctioned

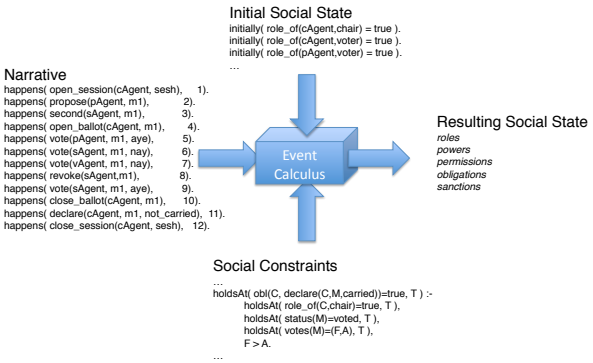
$$\begin{aligned}\mathit{close\_ballot}(C, M)\ \text{initiates}\ \mathit{sanction}(C) = [(102, M)|S]\ \text{at}\ T \leftarrow\\ \mathit{role\_of}(C, \mathit{chair}) = \mathit{true}\ \text{holdsat}\ T\ \wedge\\ \mathbf{per}(C, \mathit{close\_ballot}(C, M)) = \mathit{false}\ \text{holdsat}\ T\ \wedge\\ \mathit{sanction}(C) = S\ \text{holdsat}\ T\end{aligned}$$

- The sanction results in penalty only if someone objects
- Feature of RONR: 'anything goes unless someone objects'

The Event Calculus: Implementation Routes

- EC has been mainly used for narrative assimilation:
  - Given a narrative, check that it is consistent
  - Given a consistent narrative, check what holds when
- There are many EC variants and implementations, for different purposes or requirements
  - Discrete Event Calculus Reasoner, for proving properties & planning
  - Cached Event Calculus, for efficient narrative assimilation
  - etc. . . .
- We will show the use of the Simplified EC for narrative assimilation
  - Pre-process directly into Prolog

The Event Calculus: Narrative Assimilation



Example: The Voting Protocol

- EC Specification pre-processed into Prolog program
- Process narratives for consistency and 'what holds when'

agent	roles	powers	permissions	obligations	sanctions
cAgent	*chair voter*	*close_ballot close_session*	*close_ballot*		
pAgent	*voter proposer*				
sAgent	*voter proposer*	*vote*	*vote*		
vAgent	*voter*				
happens(*vote*(*sAgent*, *m1*, *aye*))					
cAgent	*chair voter*	*close_ballot close_session*	*close_ballot*	*close_ballot*	
pAgent	*voter proposer*				
sAgent	*voter proposer*				
vAgent	*voter*				
happens(*close_ballot*(*cAgent*, *m1*))					
cAgent	*chair voter*	*declare close_session*	*declare*(*carried*)	*declare*(*carried*)	
pAgent	*voter proposer*				
sAgent	*voter proposer*				
vAgent	*voter*				
happens(*declare*(*cAgent*, *m1*, *not_carried*))					
cAgent	*chair voter*	*close_session*	*close_session*		102
pAgent	*voter proposer*	*propose*	*propose*		
sAgent	*voter proposer*	*propose*	*propose*		
vAgent	*voter*				

Part II – Summary: The Event Calculus

- Representation of the (non-deterministic, conditional, indirect, delayed) effects of (concurrent) actions, default persistence of facts ('inertia'), etc.
- Structured specifications.
- Direct routes to implementation.
  - Efficient implementations for narrative assimilation.
  - The domains of variables may be unknown.
  - Can directly build upon Prolog's tracing facility.

Engineering Norm-Governed Systems

Part III: Specifying and Executing Norm-Governed Systems in the Action Language $C+$

Alexander Artikis and Jeremy Pitt

Outline

- The action language $C+$.
- Running example: a resource-sharing protocol.
  - Specification in $C+$ of:
    - Institutional Power.
    - Permission & Obligation.
    - Sanction.
- $C+$ implementation routes.
- Resource-sharing protocol execution.

The $C+$ Language

- Representation of the (non-deterministic, conditional, indirect, delayed) effects of (concurrent) actions, default persistence of facts ('inertia'), etc.
- Structured specifications.
- An action language with explicit transition system semantics. Therefore, there are direct links to:
  - Model checkers.
  - The $nC+$ language, specifically designed for modelling the institutional aspects of agent systems.
- Direct routes to implementation.

The $C+$ Language

- An action signature in $C+$ includes fluents and actions.
- An action description in $C+$ is a set of $C+$ rules that define a transition system.
  - Static rules:

    caused F if G
  - Dynamic rules:

    caused F if G after $\alpha \wedge H$
  - Action dynamic rules:

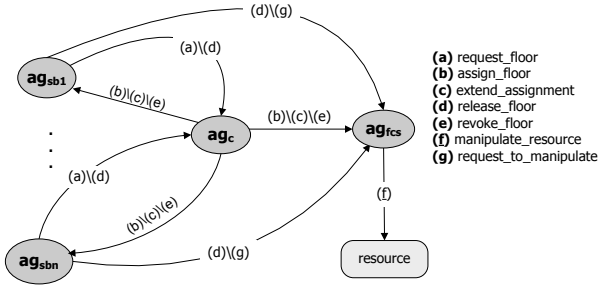
    caused α if H

The $C+$ Language

Abbreviations for causal rules:

α causes F if H	caused F if $\top$ after $\alpha \wedge H$
nonexecutable α if F	caused $\bot$ if $\top$ after $\alpha \wedge F$
default F	caused F if F
caused F iff G	caused F if G
	default $\neg F$
inertial F	caused F if F after F

A Resource-Sharing Protocol



Institutional Power

- We want to express the effects of the protocol actions, eg *assign_floor*.
- For the specification of the effects of this action, it is important to distinguish between:
  - the act of ('successfully') assigning the floor, and
  - the act by means of which the assignment of the floor is signalled (eg, sending a message of a particular form via a TCP/IP socket connection).

Institutional Power

- Institutional power to assign the floor:

  caused $powAssign(C, Sb)$ if
  $role\_of(C) = chair$,
  $\forall Sb'\ \ status(Sb') = 0$,
  $requested(Sb) \neq null$

- Effects of assigning the floor:

  $assign\_floor(C, Sb)$ causes $status(Sb) = CL{+}60$ if
  $powAssign(C, Sb)$,
  $clock = CL$

Permission & Obligation

Action: $assign\_floor(C, Sb)$

Power	Permission	Obligation
$role\_of(C) = chair$, $\forall Sb'\ status(Sb') = 0$, $requested(Sb) \neq null$	$role\_of(C) = chair$, $\forall Sb'\ status(Sb') = 0$, $best\_candidate = Sb$	$\bot$

Institutional Power, Permission & Obligation

Action: $request\_floor(Sb, C, M)$
Action: $revoke\_floor(C)$

Power	Permission	Obligation
$role\_of(Sb) = subject$, $requested(Sb) = null$	$role\_of(Sb) = subject$	$\bot$
$role\_of(C) = chair$, $status(Sb) > 0$, $clock \geq status(Sb)$, $best\_candidate \neq Sb$	$role\_of(C) = chair$, $status(Sb) > 0$, $clock \geq status(Sb)$, $best\_candidate \neq Sb$	$role\_of(C) = chair$, $status(Sb) > 0$, $clock \geq status(Sb)$, $best\_candidate = Sb'$, $Sb \neq Sb'$

Institutional Power, Permission & Obligation

Action: $request\_floor(Sb, C, M)$
Action: $revoke\_floor(C)$

Power	Permission	Obligation
$role\_of(Sb) = subject$, $requested(Sb) = null$	$role\_of(Sb) = subject$	$\bot$
$role\_of(C) = chair$, $status(Sb) > 0$, $clock \geq status(Sb)$, $best\_candidate \neq Sb$	$role\_of(C) = chair$, $status(Sb) > 0$, $clock \geq status(Sb)$, $best\_candidate \neq Sb$	$role\_of(C) = chair$, $status(Sb) > 0$, $clock \geq status(Sb)$, $best\_candidate = Sb'$, $Sb \neq Sb'$

Sanction

- Regimentation is not always desirable or practical.
- Sanctions are a means of dealing with 'undesirable' or 'unacceptable' behaviour. We specify:
  - when is an agent sanctioned, and
  - what is the penalty that the agent has to face.
- When? Eg:

  $assign\_floor(C, Sb)$ causes $sanctioned(C)$ if
  $role\_of(C) = chair$,
  $\neg perAssign(C, Sb)$

Sanction

Penalty for chair:

- temporarily lose the power to assign the floor:

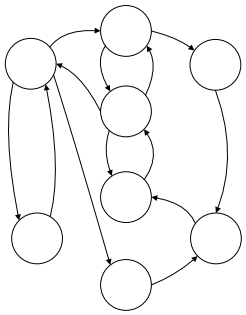
  caused $powAssign(C, Sb)$ if
  $role\_of(C) = chair$,
  $\forall Sb' \;\; status(Sb') = 0$,
  $requested(Sb) \neq null$,
  $\neg sanctioned(C)$

- ban, ie the agent is permanently disqualified from resource-sharing systems.

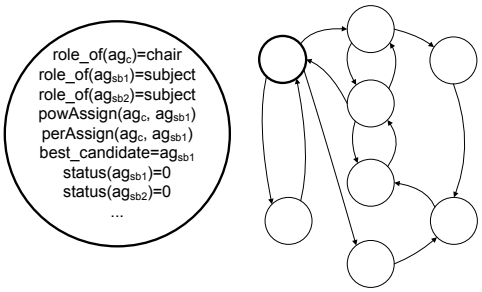
Penalty for subjects:

- requests for the floor are given a low priority;
- temporarily lose the power to request the floor;
- ban.

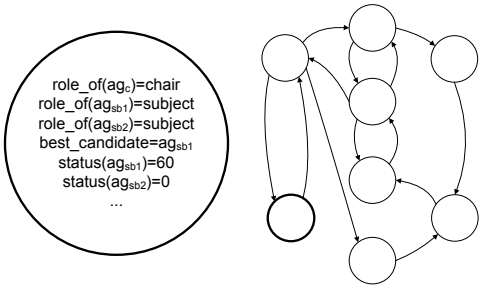
The $C+$ Language: Transition System Semantics



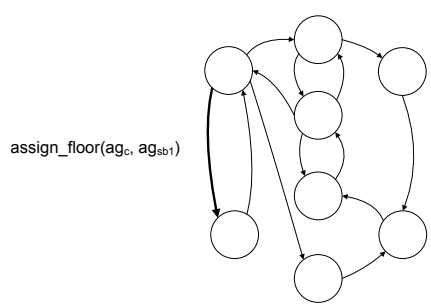
The $C+$ Language: Transition System Semantics



The $C+$ Language: Transition System Semantics

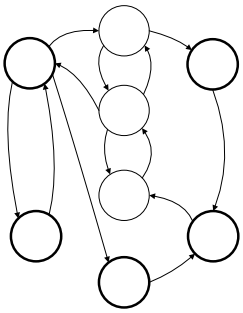


The $C+$ Language: Transition System Semantics



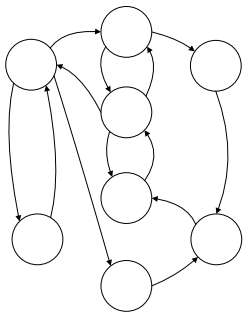
The $C+$ Language: Proving Properties

$s \models powRequest(Sb, C, M)$

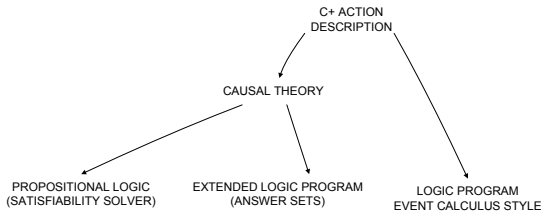


The $C+$ Language: Proving Properties

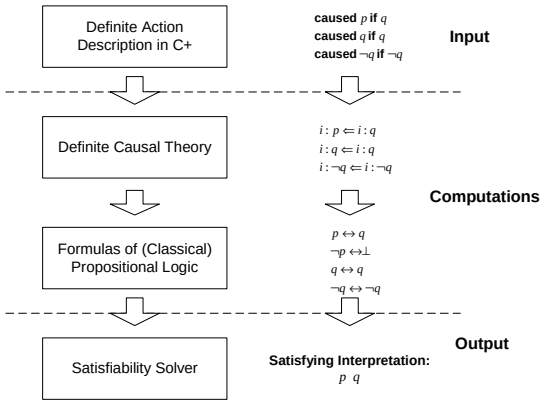
$s \not\models \neg perRevoke(C) \wedge oblRevoke(C)$



The $C+$ Language: Implementation Routes



$C+$ Implementation Routes: The Causal Calculator



The Causal Calculator

The Causal Calculator (CCALC) can be used:

- at design-time
  - to prove properties of the specification of a norm-governed system
- at run-time
  - to compute the system state current at each time

Executing the Specification:

Computing Protocol States

Compute all states s' such that:

- (s, ε, s') is a transition of D^{RS},
- $s \models powAssign(ag_c, ag_{sb1}) \wedge best\_candidate = ag_{sb3}$, and
- $\varepsilon \models assign\_floor(ag_c, ag_{sb1})$.

CCALC computes that
$s' \models status(ag_{sb1}) = clock{+}60 \ \wedge \ sanctioned(ag_c)$

Executing the Specification:

Computing Protocol States

Protocol Run
$request\_floor(ag_{sb1},\ ag_c,\ file_{exe})$
$request\_floor(ag_{sb4},\ ag_c,\ file_{pdf})$
$request\_floor(ag_{sb2},\ ag_c,\ file_{exe})$
$assign\_floor(ag_c,\ ag_{sb4})$
$request\_to\_manipulate(ag_{sb4},\ ag_{fcs})$
$request\_floor(ag_{sb7},\ ag_c,\ file_{jpg})$
$request\_floor(ag_{sb3},\ ag_c,\ file_{exe})$
...

current state $\models best\_candidate = ag_{sb1} \wedge$
$powAssign(ag_c, ag_{sb1}) \wedge perAssign(ag_c, ag_{sb1})$

Executing the Specification:

Computing Protocol States

Protocol Run
$request\_floor(ag_{sb1},\ ag_c,\ file_{exe})$
$request\_floor(ag_{sb4},\ ag_c,\ file_{pdf})$
$request\_floor(ag_{sb2},\ ag_c,\ file_{exe})$
$assign\_floor(ag_c,\ ag_{sb4})$
$request\_to\_manipulate(ag_{sb4},\ ag_{fcs})$
$request\_floor(ag_{sb7},\ ag_c,\ file_{jpg})$
$request\_floor(ag_{sb3},\ ag_c,\ file_{exe})$
...

current state $\models best\_candidate = ag_{sb1} \wedge$
$powAssign(ag_c, ag_{sb1}) \wedge perAssign(ag_c, ag_{sb1}) \wedge$
$powAssign(ag_c, ag_{sb4})$

Executing the Specification:

Computing Protocol States

Protocol Run
$request\_floor(ag_{sb1},\ ag_c,\ file_{exe})$
$request\_floor(ag_{sb4},\ ag_c,\ file_{pdf})$
$request\_floor(ag_{sb2},\ ag_c,\ file_{exe})$
$assign\_floor(ag_c,\ ag_{sb4})$
$request\_to\_manipulate(ag_{sb4},\ ag_{fcs})$
$request\_floor(ag_{sb7},\ ag_c,\ file_{jpg})$
$request\_floor(ag_{sb3},\ ag_c,\ file_{exe})$
...

current state $\models best\_candidate = ag_{sb1} \wedge$
$powAssign(ag_c, ag_{sb1}) \wedge perAssign(ag_c, ag_{sb1}) \wedge$
$powAssign(ag_c, ag_{sb4}) \wedge$
$powAssign(ag_c, ag_{sb2})$

Executing the Specification:

Computing Protocol States

Protocol Run
$request\_floor(ag_{sb1},\ ag_c,\ file_{exe})$
$request\_floor(ag_{sb4},\ ag_c,\ file_{pdf})$
$request\_floor(ag_{sb2},\ ag_c,\ file_{exe})$
$assign\_floor(ag_c,\ ag_{sb4})$
$request\_to\_manipulate(ag_{sb4},\ ag_{fcs})$
$request\_floor(ag_{sb7},\ ag_c,\ file_{jpg})$
$request\_floor(ag_{sb3},\ ag_c,\ file_{exe})$
...

current state $\models status(ag_{sb4}) = clock{+}60 \wedge$
$powRequestManipulate(ag_{sb4}, ag_{fcs}) \wedge$
$sanctioned(ag_c)$

Summary: The $C+$ Language

- Representation of the (non-deterministic, conditional, indirect, delayed) effects of (concurrent) actions, default persistence of facts ('inertia'), etc.
- Structured specifications.
- An action language with explicit transition system semantics. Therefore, there are direct links to:
  - Model checkers.
  - The $nC+$ language, specifically designed for modelling the institutional aspects of agent systems.
- Direct routes to implementation.
  - Proving properties & narrative assimilation.
  - CCALC does not scale well to large transition systems.
  - The domain of each variable must be known.
  - No tracing facility.

Summary: The Event Calculus

- Representation of the (non-deterministic, conditional, indirect, delayed) effects of (concurrent) actions, default persistence of facts ('inertia'), etc.
- Structured specifications.
- Direct routes to implementation.
  - Efficient implementations for narrative assimilation.
  - The domains of variables may be unknown.
  - Can directly build upon Prolog's tracing facility.

Engineering Norm-Governed Systems

Part IV: Dynamic Specifications for Norm-Governed Systems

Alexander Artikis and Jeremy Pitt

Static vs Dynamic Specification

'Static' specification: developed at design-time, no support for run-time modification.

Due to environmental, social, or other conditions it is often desirable, or even necessary, to alter the system specification during the system execution. Eg:

- a malfunction of a large number of sensors in a sensor network
- manipulation of a voting procedure due to strategic voting
- an organisation conducts its business in an inefficient manner

'Dynamic' specification: developed at design-time, may be modified at run-time by the members of a system.

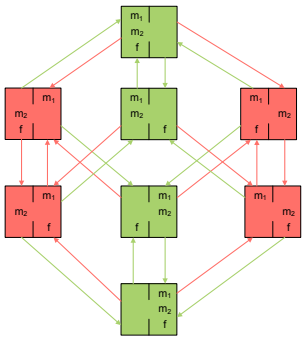
Outline

- Dynamic specification: Example
- Dynamic specifications in:
  - Argument systems
  - Formal (logic-based) study of multi-agent systems

Dynamic Specification: Example

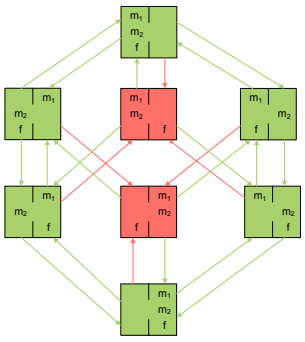
```
move(Ag)=P causes loc(Ag)=P
nonexecutable move(Ag)=P if loc(Ag)=P
nonexecutable move(Ag)=P if move(Ag2)=P and Ag<>Ag2
not-permitted loc(m1)=loc(f) if loc(m2)<>loc(f)
not-permitted loc(m2)=loc(f) if male(m1)<>loc(f)
```

} Degree of Freedom



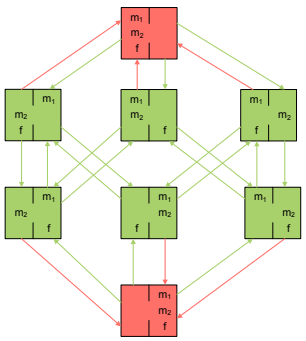
Dynamic Specification: Example

```
move(Ag)=P causes loc(Ag)=P
nonexecutable move(Ag)=P if loc(Ag)=P
nonexecutable move(Ag)=P if move(Ag2)=P and Ag<>Ag2
not-permitted loc(f)<>loc(m1) and loc(f)<>loc(m2)
```



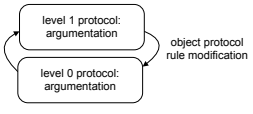
Dynamic Specification: Example

```
move(Ag)=P causes loc(Ag)=P
nonexecutable move(Ag)=P if loc(Ag)=P
nonexecutable move(Ag)=P if move(Ag2)=P and Ag<>Ag2
not-permitted loc(m1)=loc(m2)=loc(f)
```



Dynamic argument systems

- Organised adaptation is common in argument systems.
- Loui, eg, talked about the need for argumentation protocol change by means of meta-argumentation.
- Vreeswijk also investigated forms of meta-argumentation:



Dynamic argument systems

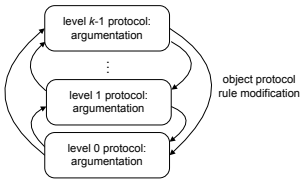
Observations:

- There are different protocols appropriate for different contexts (eg, quick and shallow reasoning when time is a constraint; restricted number of counter-arguments when there are many rules and cases).
- 'Points of order', by which a participant may steer the protocol to a desired direction, are standard practice in dispute resolution meetings.

Vreeswijk then:

- Defined a formal protocol for disputes in which points of order can be raised to allow (partial) protocol changes to be debated.
- A successful 'defense' meant that the parties in the dispute agreed to adopt a change in the protocol, and the rules of dispute were correspondingly changed.

Dynamic argument systems



Brewka's dynamic argument systems:

- More than one meta level is allowed.
- *Any* part of a protocol specification may be modified.
- Very general framework — eg the group in the debate may vary from level to level, the available actions may differ from level to level, etc.

Dynamic argument systems

Example run of a dynamic argument system:

$claim(pro, murderer(jack), 0)$
$claim(pro, on(blood, shoe), 0)$
$claim(opp, illegal\_info(on(blood, shoe)), 0)$
$concede(pro, illegal\_info(on(blood, shoe)), 0)$
$propose(opp, R, 0)$

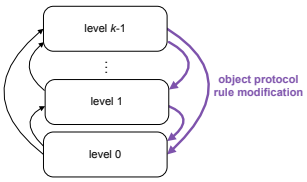
where R is a rule stating that if an agent Ag has a premise P and P was illegally obtained then the only legal action for Ag is to retract P

$decide(det, R, 1)$

At this point the only legal action available to *pro* is:
$retract(pro, on(blood, shoe), 0)$

Formal Models of Norm Change

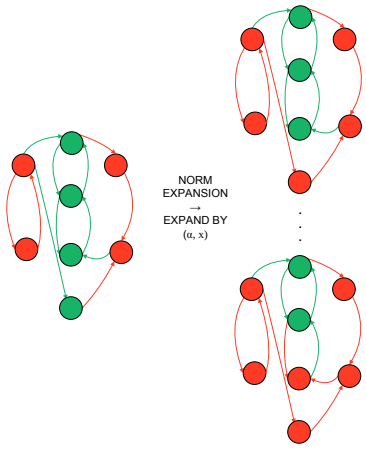
- Formal models of norm change are being actively investigated.
- See, eg, the 2010 and 2007 editions of the FMNC Workshop.



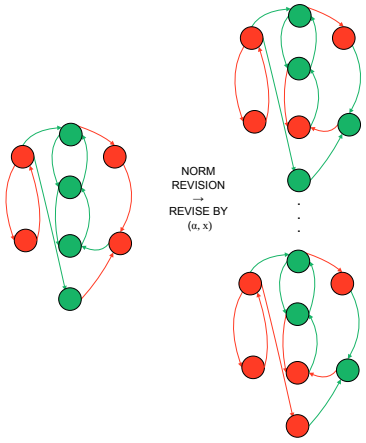
Formal Models of Norm Change

- Boella et al have presented a formal framework for norm change.
- The framework is produced by replacing the propositional formulas of the AGM framework of theory change with *pairs* of propositional formulas — the latter representing norms — and adopting several principles from input/output logic.
- The resulting framework includes a set of postulates defining norm change operations: norm expansion, norm contraction and norm revision.

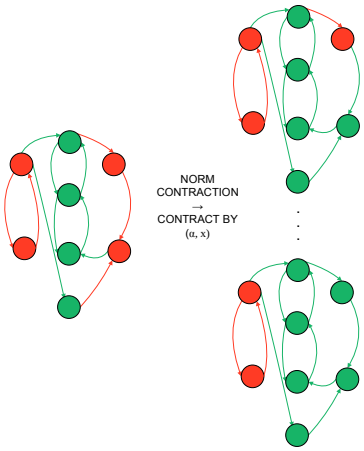
Formal Models of Norm Change: Norm Expansion



Formal Models of Norm Change: Norm Revision



Formal Models of Norm Change: Norm Contraction



Formal Models of Norm Change: Norm Contraction

Example postulates that must be satisfied by norm contraction:

- Inclusion: For any context b, if something is obligatory after norm contraction, then it was already obligatory before the contraction.
- Vacuity: If a norm (α, x) is contracted, but x is not obligatory in context α, then the contraction has no effect.
- Success: If we remove norm (α, x) from a norm-governed system N then N will not contain (α, x).
- Recovery: Contracting a norm-governed system by (α, x) and then expanding by the same (α, x) should leave N unchanged.

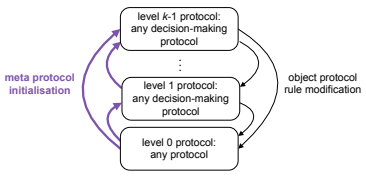
There are postulates defining the remaining norm change operations.

Formal Models of Norm Change: Abrogation and Annulment

Formalisation of further norm change operations (Governatori et al):

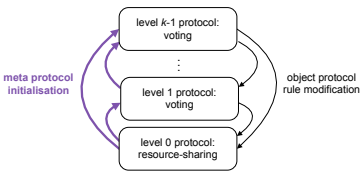
- Annulment: remove a norm as well as its effects, irrespective of when they were produced.
- Abrogation: remove a norm but do not remove the effects of the norm that were produced prior to the norm removal.
- Formalisation of norm change operations in a Temporal Defeasible Logic.

Dynamic Systems



- Any protocol for norm-governed systems can be in level 0.
- Any protocol for decision-making over rule modification can be in level n, $n > 0$.
- Emphasis is placed on the formalisation of the transition protocols: the procedures with which a meta-protocol is initiated.

Dynamic Systems

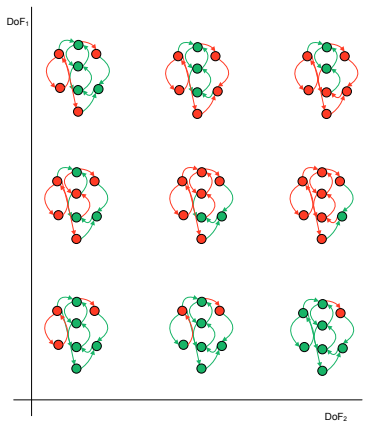


- Any protocol for norm-governed systems can be in level 0.
- Any protocol for decision-making over rule modification can be in level n, $n > 0$.
- Emphasis is placed on the formalisation of the transition protocols: the procedures with which a meta-protocol is initiated.

Dynamic Systems: Specification Space

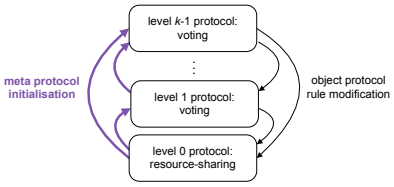
- We define the Degrees of Freedom (DoF) of a protocol.
- A protocol specification with n DoF creates an n-dimensional specification space, where each dimension corresponds to a DoF.
- A specification point represents a complete protocol specification — a specification instance.

Dynamic Systems: Specification Space



Transition Protocol:

Successful vs unsuccessful attempt to initiate a meta protocol

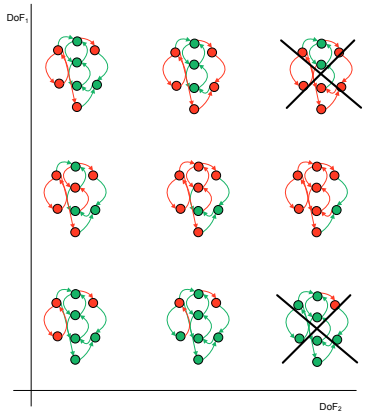


An agent initiates a meta protocol by exercising its institutional power to propose a specification change.

caused $powPropose(Ag, NSP, PL)$ if
$spec\_point(PL) = SP$, $SP \neq NSP$,
$protocol(PL{+}1) = idle$,
$properties(NSP, PL)$

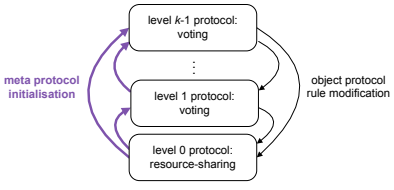
Specification Space:

Successful vs unsuccessful attempt to initiate a meta protocol



Transition Protocol:

Controlling run-time specification change



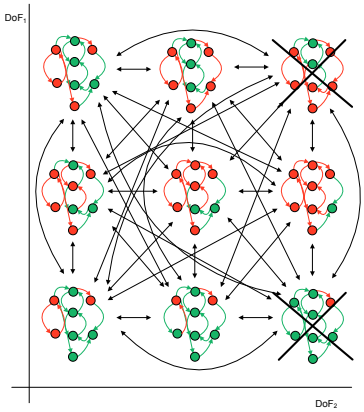
- We evaluate a proposal for specification change.
- We constrain the enactment of proposals that do not meet the evaluation criteria.

Transition Protocol:

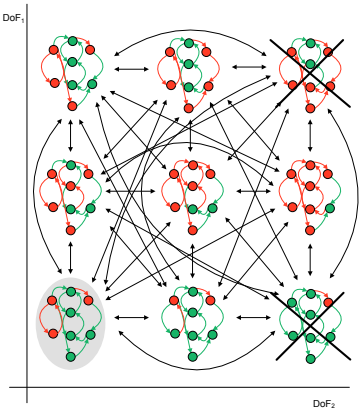
Evaluating & Constraining Proposals for Specification Change

- We translate a specification space into a metric space:
  - eg, we define an application-specific distance function
- We compute the 'distance' between the current specification point and the proposed specification point.
- We forbid the adoption of a specification point that is 'too far' from the current specification point.

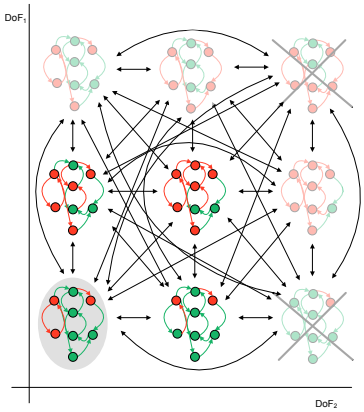
Specification Space as a Metric Space



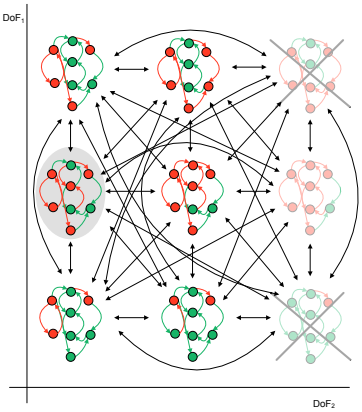
Specification Space as a Metric Space



Constraining Specification Change

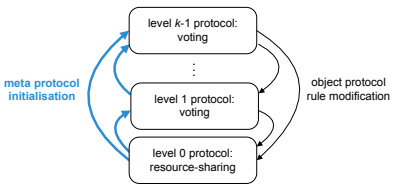


Constraining Specification Change



Executing the Specification

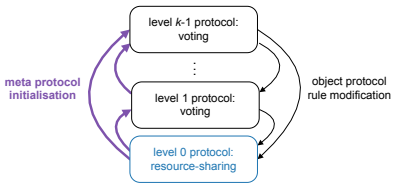
Proving Properties



- $s \models powPropose(Ag, NSP, PL) \wedge spec\_point(PL) = SP \wedge distance(SP, NSP) > threshold(PL)$
- $s \not\models powPropose(Ag, NSP, PL) \wedge protocol(PL') = executing \wedge PL' > 0$

Executing the Specification

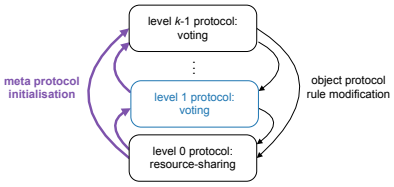
Proving Properties



- $s \models spec\_point(0) = sp_2 \wedge powRequest(Sb, C, M)$
- $s \models spec\_point(1) = sp_3 \wedge powVote(Ag, V)$

Executing the Specification

Proving Properties



- $s \models spec\_point(0) = sp_2 \wedge powRequest(S, C, M)$
- $s \models spec\_point(1) = sp_3 \wedge powVote(Ag, V)$

Executing the Specification

Computing Protocol States

Compute all states s' such that:

- (s, ε, s') is a transition of D^{DRS},
- $s \models spec\_point(0) = sp_2 \wedge$
  $distance(sp_1, sp_2) > threshold(0)$, and
- $\varepsilon \models propose(ag_{sb3}, sp_1, 0)$.

CCALC computes that
$s' \models protocol(1) = executing \wedge sanctioned(ag_{sb3})$

Executing the Specification:

Computing Protocol States

Protocol Run

$request\_floor(ag_{sb1}, ag_c, file_{pdf})$
$request\_floor(ag_{sb2}, ag_c, file_{jpg})$
$request\_floor(ag_{sb3}, ag_c, file_{exe})$
$request\_floor(ag_{sb5}, ag_c, file_{pdf})$
  $propose(ag_{sb3}, sp_5, 1)$
    $vote([ag_{sb2}, ag_{sb3}, ag_{sb4}, ag_{sb5}, ag_{sb6}], for, 2)$
    $vote(ag_{sb1}, against, 2)$
    $declare(ag_c, carried, 2)$
  $propose(ag_{sb5}, sp_3, 0)$
    $vote([ag_{sb3}, ag_{sb4}, ag_{sb6}], for, 1)$
    $vote([ag_{sb1}, ag_{sb2}], against, 1)$
    $declare(ag_c, carried, 1)$
$assign\_floor(ag_c, ag_{sb3})$
...

Protocol Run

$request\_floor(ag_{sb1}, ag_c, file_{pdf})$
$request\_floor(ag_{sb2}, ag_c, file_{jpg})$
$request\_floor(ag_{sb3}, ag_c, file_{exe})$
$request\_floor(ag_{sb5}, ag_c, file_{pdf})$
  $propose(ag_{sb3}, sp_5, 1)$
    $vote([ag_{sb2}, ag_{sb3}, ag_{sb4}, ag_{sb5}, ag_{sb6}], for, 2)$
    $vote(ag_{sb1}, against, 2)$
    $declare(ag_c, carried, 2)$
  $propose(ag_{sb5}, sp_3, 0)$
    $vote([ag_{sb3}, ag_{sb4}, ag_{sb6}], for, 1)$
    $vote([ag_{sb1}, ag_{sb2}], against, 1)$
    $declare(ag_c, carried, 1)$
$assign\_floor(ag_c, ag_{sb3})$
...

current state $\models protocol(2) = executing$

Protocol Run

$request\_floor(ag_{sb1}, ag_c, file_{pdf})$
$request\_floor(ag_{sb2}, ag_c, file_{jpg})$
$request\_floor(ag_{sb3}, ag_c, file_{exe})$
$request\_floor(ag_{sb5}, ag_c, file_{pdf})$
  $propose(ag_{sb3}, sp_5, 1)$
    $vote([ag_{sb2}, ag_{sb3}, ag_{sb4}, ag_{sb5}, ag_{sb6}], for, 2)$
    $vote(ag_{sb1}, against, 2)$
    $declare(ag_c, carried, 2)$
  $propose(ag_{sb5}, sp_3, 0)$
    $vote([ag_{sb3}, ag_{sb4}, ag_{sb6}], for, 1)$
    $vote([ag_{sb1}, ag_{sb2}], against, 1)$
    $declare(ag_c, carried, 1)$
$assign\_floor(ag_c, ag_{sb3})$
...

current state $\models protocol(1) = executing \wedge$
$sanctioned(ag_{sb5})$

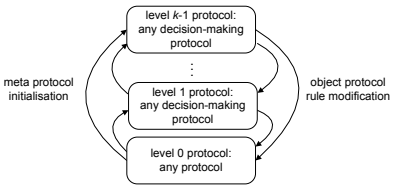
Protocol Run

$request\_floor(ag_{sb1}, ag_c, file_{pdf})$
$request\_floor(ag_{sb2}, ag_c, file_{jpg})$
$request\_floor(ag_{sb3}, ag_c, file_{exe})$
$request\_floor(ag_{sb5}, ag_c, file_{pdf})$
  $propose(ag_{sb3}, sp_5, 1)$
    $vote([ag_{sb2}, ag_{sb3}, ag_{sb4}, ag_{sb5}, ag_{sb6}], for, 2)$
    $vote(ag_{sb1}, against, 2)$
    $declare(ag_c, carried, 2)$
  $propose(ag_{sb5}, sp_3, 0)$
    $vote([ag_{sb3}, ag_{sb4}, ag_{sb6}], for, 1)$
    $vote([ag_{sb1}, ag_{sb2}], against, 1)$
    $declare(ag_c, carried, 1)$
$assign\_floor(ag_c, ag_{sb3})$
...

current state $\models spec\_point(0) = sp_3$

Summary: Dynamic Specifications

- Dynamic specification: developed at design-time, modified at run-time by the members of a system
- Degrees of Freedom: specification components that may change at run-time
- Specification change operations range from simple to very complex
- Procedures according to which an agent may successfully change a specification range from simple to very complex



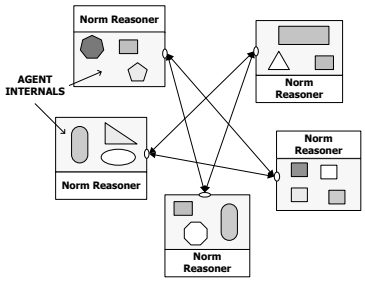
Engineering Norm-Governed Systems

Part V:
Run-Time Configurations
for Norm-Governed Systems

Alexander Artikis and Jeremy Pitt

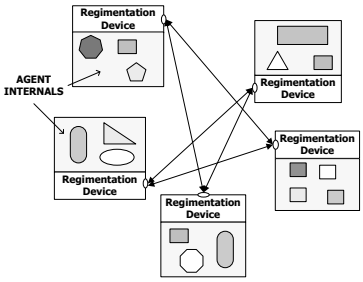
Run-Time Configurations: Total Distribution

The normative positions current at each time are computed in various distributed configurations — total distribution:



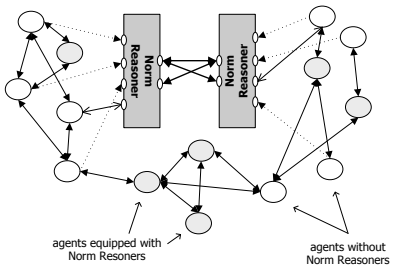
Run-Time Configurations: Regimentation

The normative positions current at each time are computed in various distributed configurations — regimentation:



Run-Time Configurations: Partial Distribution

The normative positions current at each time are computed in various distributed configurations — partial distribution:



Run-Time Configurations

- Offered services:
  - prediction
  - planning
  - postdiction
  - ...
- Policies for publicising the protocol state
- Distributed implementations of automated reasoning tools
- Specification change

Specification Change: Architectural Issues

- When a specification changes, all entities of a system should be aware of the new specification instance immediately, and in a virtually atomic manner.
- However, this is not possible in large, distributed, heterogeneous systems:
  - Some entities may be temporarily unreachable, eg due to network failure.
  - Some entities may become aware of the new specification instance, but may not be 'able' to comply with the new instance because they are in the middle of a transaction governed by the previous specification instance.
- Consequently, it is often the case that different entities operate under different specification instances — convergence period.
- Conflicts are likely during convergence periods.

Specification Change: Architectural Issues

Approach of the Law-Governed Interaction (LGI) framework.

- Communication under specification dispersion:
  - Messages from the future: Messages sent by agents operating under newer specification instances are blocked by the regimentation device of the receiver.
  - Messages from the past: Each new specification instance states how to deal with messages coming from agents operating under the previous specification instance.
- Specification change process:
  - Seeding: A new specification instance is disseminated to a subset of the members of a system.
  - Peer-to-Peer convergence: When receiving a message from the past ask the sender to operate under the new specification instance. Similarly, when receiving a message from the future update to the new specification instance.

Engineering Norm-Governed Systems

Part VI: Open Issues

Alexander Artikis and Jeremy Pitt

Engineering Norm-Governed Systems: Open Issues

- Efficient automated reasoning tools supporting design-time & run-time activities (proving properties & narrative assimilation).
- Complexity of norm change operations for software systems (as opposed to human legal systems).
- When is adaptation performed? Mechanisms for identifying the conditions in which a specification change is desirable/necessary.
- Run-time configurations for norm-governed systems.
- Techniques for dealing with specification dispersion (in non-regimented systems).

Engineering Norm-Governed Systems
Indicative References

Alexander Artikis and Jeremy Pitt

- Part I:
  - Open Systems: [25, 18].
  - Norm-Governed Systems: [35, 33, 27, 32, 38, 45, 52, 37, 41].
- Part II:
  - Event Calculus: [29, 10, 49, 11, 34, 50, 51].
  - Specifying norm-governed systems in the Event Calculus: [42, 55, 3, 16, 17].
- Part III:
  - $C+$: [19, 1], `http://userweb.cs.utexas.edu/users/tag/cc/`.
  - Specifying norm-governed systems in (extended versions of) $C+$: [47, 15, 48, 13, 14, 46, 4, 12].
  - Comparison of $C+$, Event Calculus and other action languages: [39, 40].
  - Comparison of $C+$ and Event Calculus with respect to engineering norm-governed systems: [5].
- Part IV:
  - Rooms example: [47].
  - Dynamic argument systems: [31, 54, 8].
  - Formal models of norm change: [9, 6, 20, 21, 22, 23, 24], `http://www.cs.uu.nl/events/normchange2/`.
  - Dynamic Systems: [2].
- Part V:
  - Run-time configurations: [53].
  - Regimentation devices: [26, 43, 28, 7, 30].
  - Law-Governed Interaction: [37, 36, 44], `http://www.moses.rutgers.edu/`.

References

[1] V. Akman, S. Erdogan, J. Lee, V. Lifschitz, and H. Turner. Representing the zoo world and the traffic world in the language of the Causal Calculator. *Artificial Intelligence*, 153(1–2):105–140, 2004.

[2] A. Artikis. Dynamic protocols for open agent systems. In *Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 97–104. ACM, 2009.

[3] A. Artikis and M. Sergot. Executable specification of open multi-agent systems. *Logic Journal of the IGPL*, 18(1):31–65, 2010.

[4] A. Artikis, M. Sergot, and J. Pitt. An executable specification of a formal argumentation protocol. *Artificial Intelligence*, 171(10–15):776–804, 2007.

[5] A. Artikis, M. Sergot, and J. Pitt. Specifying norm-governed computational societies. *ACM Transactions on Computational Logic*, 10(1), 2009.

[6] G. Boella, G. Pigozzi, and L. van der Torre. Normative framework for normative system change. In *Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 169–176. ACM Press, 2009.

[7] J. Bradshaw, A. Uszok, R. Jeffers, N. Suri, P. Hayes, M. Burstein, A. Acquisti, B. Benyo, M. Breedy, M. Carvalho, D. Diller, M. Johnson, S. Kulkarni, J. Lott, M. Sierhuis, and R. Van Hoof. Representation and reasoning about DAML-based policy and domain services in KAoS. In J. Rosenschein, T. Sandholm, M. Wooldridge, and M. Yokoo, editors, *Proceedings of Conference on Autonomous Agents and Multi Agent Systems (AAMAS)*, pages 835–842. ACM Press, 2003.

[8] G. Brewka. Dynamic argument systems: a formal model of argumentation processes based on situation calculus. *Journal of Logic and Computation*, 11(2):257–282, 2001.

[9] J. Broersen. Issues in designing logical models for norm change. In G. Vouros, A. Artikis, K. Stathis, and J. Pitt, editors, *Proceedings of Internation Workshop in Oprganised Adaptation on Multi-Agent Systems (OAMAS)*, volume LNCS 5368, pages 1–17. Springer, 2009.

[10] L. Chittaro and A. Montanari. Efficient temporal reasoning in the cached event calculus. *Computational Intelligence*, 12(3):359–382, 1996.

[11] L. Chittaro and A. Montanari. Temporal representation and reasoning in artificial intelligence: Issues and approaches. *Annals of Mathematics and Artificial Intelligence*, 28(1–4):47–106, 2000.

[12] A. Chopra and M. Singh. Contextualizing commitment protocols. In *Proceedings of Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1345–1352. ACM, 2006.

[13] R. Craven. *Execution Mechanisms for the Action Language $\mathcal{C}+$*. PhD thesis, University of London, September 2006.

[14] R. Craven and M. Sergot. Distant causation in C+. *Studia Logica*, 79(1):73–96, 2005.

[15] R. Craven and M. Sergot. Agent strands in the action language $nC+$. *Journal of Applied Logic*, 6(2):172–191, 2008.

[16] A. Farrell, M. Sergot, M. Sallé, and C. Bartolini. Using the event calculus for tracking the normative state of contracts. *International Journal of Cooperative Information Systems*, 4(2–3):99–129, 2005.

[17] N. Fornara and M. Colombetti. *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, chapter Formal specification of artificial institutions using the event calculus. IGI Global, 2009.

[18] L. Gasser. Social conceptions of knowledge and action: DAI foundations and open systems semantics. *Artificial Intelligence*, 47(1-3):107–138, 1991.

[19] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1–2):49–104, 2004.

[20] G. Governatori, V. Padmanabhab, and A. Rotolo. Rule-based agents in temporalised defeasible logic. In *Proceedings of Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, LNCS 4099, pages 31–40. Springer, 2006.

[21] G. Governatori, M. Palmirani, R. Riveret, A. Rotolo, and G. Sartor. Norm modifications in defeasible logic. In *Proceedings of Conference on Legal Knowledge and Information Systems (JURIX)*, pages 13–22. IOS Press, 2005.

[22] G. Governatori and A. Rotolo. Changing legal systems: Abrogation and annulment. Part I: Revision of defeasible theories. In R. van der Meyden and L. van der Torre, editors, *Proceedings of Conference on Deontic Logic in Computer Science (DEON)*, LNCS 5076, pages 3–18. Springer, 2008.

[23] G. Governatori and A. Rotolo. Changing legal systems: Abrogation and annulment. Part II: Temporalised defeasible logic. In G. Boella, G. Pigozzi, M. Singh, and H. Verhagen, editors, *Proceedings of Workshop on Normative Multiagent Systems (NORMAS)*, pages 112–127, 2008.

[24] G. Governatori, A. Rotolo, R. Riveret, M. Palmirani, and G. Sartor. Variants of temporal defeasible logics for modelling norm modifications. In *Proccedings of International Conference on Artificial Intelligence & Law (ICAIL)*. ACM, 2007.

[25] C. Hewitt. Open information systems semantics for distributed artificial intelligence. *Artificial Intelligence*, 47:79–106, 1991.

[26] A. Jones and M. Sergot. On the characterisation of law and computer systems: the normative systems perspective. In *Deontic Logic in Computer Science: Normative System Specification*, pages 275–307. J. Wiley and Sons, 1993.

[27] A. Jones and M. Sergot. A formal characterisation of institutionalised power. *Journal of the IGPL*, 4(3):429–445, 1996.

[28] M. Klein, J. Rodriguez-Aguilar, and C. Dellarocas. Using domain-independent exception handling services to enable robust open multi-agent systems: the case of agent death. *Journal of Autonomous Agents and Multi-Agent Systems*, 7(1–2):179–189, 2003.

[29] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–96, 1986.

[30] A. Lomuscio and M. Sergot. A formulation of violation, error recovery, and enforcement in the bit transmission problem. *Journal of Applied Logic*, 2(1):93–116, 2003.

[31] R. Loui. Process and policy: Resource-bounded non-demonstrative argument. Technical report, Washington University, Department of Computer Science, 1992.

[32] D. Makinson. On the formal representation of rights relations. *Journal of Philosophical Logic*, 15:403–425, 1986.

[33] J.-J. Meyer and R. Wieringa. *Deontic Logic in Computer Science: Normative System Specification*. J. Wiley and Sons, 1993.

[34] R. Miller and M. Shanahan. The event calculus in a classical logic — alternative axiomatizations. *Journal of Electronic Transactions on Artificial Intelligence*, 3(A):77–105, 1999.

[35] N. Minsky. The imposition of protocols over open distributed systems. *IEEE Transactions on Software Engineering*, 17(2):183–195, 1991.

[36] N. Minsky and T. Murata. On manageability and robustness of open multi-agent systems. In *Software Engineering for Multi-Agent Systems II, Research Issues and Practical Applications*, LNCS 2940, pages 189–206. Springer, 2004.

[37] N. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(3):273–305, 2000.

[38] Y. Moses and M. Tennenholtz. Artificial social systems. *Computers and Artificial Intelligence*, 14(6):533–562, 1995.

[39] E. Mueller. *Commonsense Reasoning*. Morgan Kaufmann, 2006.

[40] E. Mueller. Event calculus and temporal action logics compared. *Artificial Intelligence*, 170(11):1017–1029, 2006.

[41] E. Ostrom. *Governing The Commons: The Evolution of Institutions for Collective Action*. Cambridge University Press, 1990.

[42] J. Pitt, L. Kamara, M. Sergot, and A. Artikis. Voting in multi-agent systems. *Computer Journal*, 49(2):156–170, 2006.

[43] J. Rodriguez-Aguilar, F. Martin, P. Noriega, P. Garcia, and C. Sierra. Towards a test-bed for trading agents in electronic auction markets. *AI Communications*, 11(1):5–19, 1998.

[44] C. Serban and N. Minsky. In vivo evolution of policies that govern a distributed system. In *International Symposium on Policies for Distributed Systems and Networks*, pages 134–141. IEEE, 2009.

[45] M. Sergot. A computational theory of normative positions. *ACM Transactions on Computational Logic*, 2(4):522–581, 2001.

[46] M. Sergot. Modelling unreliable and untrustworthy agent behaviour. In B. Dunin-Keplicz, A. Jankowski, A. Skowron, and M. Szczuka, editors, *Proceedings of Workshop on Monitoring, Security, and Rescue Techniques in Multiagent Systems (MSRAS)*, Advances in Soft Computing, pages 161–178. Springer-Verlag, 2004.

[47] M. Sergot. Action and agency in norm-governed multi-agent systems. In A. Artikis, G. O'Hare, K. Stathis, and G. Vouros, editors, *Proceedings of ESAW VIII*, LNAI 4995, pages 1–54. Springer, 2008.

[48] M. Sergot and R. Craven. The deontic component of action language $nC+$. In L. Goble and J-J. Ch. Meyer, editors, *Deontic Logic in Computer Science (DEON)*, LNAI 4048, pages 222–237. Springer, 2006.

[49] M. Shanahan. The event calculus explained. In M. Wooldridge and M. Veloso, editors, *Artificial Intelligence Today*, LNAI 1600, pages 409–430. Springer, 1999.

[50] M. Shanahan. An abductive event calculus planner. *Journal of Logic Programming*, 44:207–239, 2000.

[51] M. Shanahan and M. Witkowski. Event calculus planning through satisfiability. *Journal of Logic and Computation*, 14(5):731–745, 2004.

[52] M. Singh. Agent communication languages: rethinking the principles. *IEEE Computer*, 31(12):40–47, 1998.

[53] V. Urovi, S. Bromuri, K. Stathis, and A. Artikis. Towards runtime support for norm-governed multi-agent systems. In *Proceedings of Knowledge Representation (KR)*, 2010.

[54] G. Vreeswijk. Representation of formal dispute with a standing order. *Artificial Intelligence and Law*, 8(2/3):205–231, 2000.

[55] P. Yolum and M. Singh. Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence*, 42(1–3):227–253, 2004.