Conflict Detection in Composite Institutions

Tingting Li[1], Tina Balke[2,1], Marina De Vos[1], Ken Satoh[3], and Julian Padget[1]

[1] University of Bath, Dept. of Computer Science, UK
{t.li,mdv,jap}@cs.bath.ac.uk
[2] University of Surrey, Centre for Research in Social Simulation, UK
t.balke@surrey.ac.uk
[3] National Institute of Informatics, Principles of Informatics Res. Division, Japan
ksatoh@nii.ac.jp

Abstract. Institutions provide a mechanism to specify policies for actors in open distributed systems. From a software engineering perspective it makes sense to group policies based on common goals or expectations. This allows them to be reused and combined. However, the combination of institutions with different agendas will almost certainly lead to conflicts – e.g. one set of policies permitting a certain action while another does not. We propose a way to detect such conflicts by analysing formalized statements of policy, or in the absence of conflicts, to verify that the combination of a given set of institutions is conflict-free.

1 Introduction

Institutions have long been studied in the agent community as a means for governing systems by specifying the policies that guide interactions between agents. Each institution consists of a set of policies. Viewed from the domain of the social sciences, policies are understood as instruments that are implemented with the aim of encouraging society to adopt certain norms [19][4]. In order to encourage society to adopt a certain norm, policies dictate which actions (or outcomes) are in principle permitted, empowered, prohibited or obligatory under given sets of conditions, as well as specifying the consequences of compliance or non-compliance.

Computer science research has largely focused on the modelling of either single institutions or multiple interacting institutions, all of which have been designed by the same designers for a particular system (see for example [3, 4, 12, 18]). So far, we have not found any work addressing the issues of modelling the composition of independently-designed institutions. We define composition as the combinination of the norms of several institutions such that their combined policies are consistent to the user. This could be done in a variety of ways.

The problem when composing different institutions is that each is designed for its own purpose, and not some common objective. This can result in situations where the policies of the individual institutions are inconsistent when they are

[4] Policies are not the only way norms can be implemented. They can also emerge as generally accepted social behaviour.

composed, giving rise to problems for the participants governed by the composed system. For example, it is unacceptable for a participant to have the permission to perform a certain action in one institution, while it is not permitted in another at the same time. This is why it is important to be able to detect and resolve these kind of conflicts at the design stage of the composition, before any agents are interacting with that system. This paper takes the first steps at solving the composition problem by presenting mechanisms to detect conflicts in composite institutions at design time.

Socio-economic systems are a prime example of systems where composite institutions are likely to occur. They are complex systems that can be viewed from different perspectives, each with its own policies on how participants should behave. These differences need to be combined into a coherent set of policies in order to make it workable for the participants. These policies, expressed as institutions, can be used to state for example the legal context, the technical protocols of the system and the social perspective expressing the "netiquette" for the social interaction between actors.

In order to illustrate our approach we present a small example of a virtual community, which we see as a socio-technical system governed by both social and formal institutions, that need to be composed and made conflict-free to meet the social requirements of the virtual community [11]. The essential source of conflict is that governance rules typically derive from two sources: a prescriptive kind of template that captures the general requirements and may be imposed from above and the much more specific rules that represent the desires of that particular virtual community, which taken together provide the means for each community to work out its own community-specific system of governance [16].This customization of policy is of particular importance in virtual communities, as it has been shown that members becoming actively involved in community moderation and standard settings is a necessary condition for the virtual social communities to become self-sustaining in the long run [6, 1]. One important feature of the governance of virtual communities therefore is to detect conflicts between the different set of policies early on when they being composed. According to [15] virtual communities: (i) consist of *people*, who interact socially with the goal of satisfying their own need, but at the same time (ii) have a *shared purpose* such as as an interest, need, information exchange, or service that provides a reason for the community. (iii) are governed by *policies*, both formal (e.g. laws, technical protocols and organizational rules) and social (e.g. rituals and informal interaction rules) and (iv) are enabled by means of *computer systems* to support and mediate social interaction and facilitate a sense of community.

The purpose of this paper is to demonstrate how conflicts can be detected at the design stage in composing institutions and so the remainder of the paper is structured as follows: We start by explaining our view of institutions and the corresponding computational model of a single institution (Section 2.1). These ideas are then extended to capture composite institutions in Section 3.1. Consequently, in Section 4 we focus on the detection of conflicts between the policies of composed institutions. After giving the conceptual and formal description of

conflicts and explaining how they can be detected, we use the case study of virtual communities outlined earlier to demonstrate our approach with a worked example. The paper ends with a short summary, conclusions and an outline of future work (Section 5).

2 Institutions

To provide some context for the theory that follows, this section begins with a brief overview of individual institutions and the terminology that we use.

Institutions as mechanisms to regulate systems have been studied at length in the literature (see [12–14, 18] for example). Probably the most relevant fact for this paper is the recognition of *policies* as instruments – use by policy makers – to implement norms and to encouraging society to adopt these norms [19]. In this paper we refer to institutions as a set of policies that encourage specific normative behaviour. We do not assume that participants in the system are necessarily norm compliant nor that they have internalised the norms or policies of the system.

2.1 Individual Institutions

The literature contains a number of frameworks and methods to model institutions (e.g. [7, 8, 10] to cite but a few). We follow the approach presented by Cliffe et al. [2, 3], which uses an event-driven model, where the events derive from the actions of the participants/users of the system. The institution is used by the participants to determine the most appropriate course of action based on the normative information available. The approach is centred around *observable events*, participants' actions and changes in the environment, that are interpreted in a given institutional context. The advantage of this framework is that the formal model can be translated to a corresponding *AnsProlog* program [9] – a logic program under answer set semantics – allowing for reasoning about and verification and validation of the institution and its policies.

Formal Model The observable events ($\mathcal{E}_{ex}$) used by Cliffe et al. are external to the institution (and therefore also sometimes referred to as exogenous events). They capture the notion of events in the physical world. Besides these observable events, Cliffe et al. introduce institutional events ($\mathcal{E}_{inst}$) that are events generated by the institution, but which only have meaning in the institutional context. To give an example of this: an observable event in the physical world would be "shooting" someone. The corresponding institutional event would be the interpretation of this physical action as murder in the institutional context. The notion of *conventional generation* (by Searle [17] is used to generate institutional events from the occurrence of an exogenous event. Using the so-called "count-as" statements, events in one context count as events in another context. So, using the physical world as the first context and the institution as the second,

observed events "generate" institutional events. This can be further extended to institutional events generating other institutional events.

Institutional events are partitioned into institutional actions ($\mathcal{E}_{act}$) that denote changes in the institutional state and violation events ($\mathcal{E}_{viol}$), that signal the occurrence of violations. Violations may arise either from explicit generation, (i.e. from the occurrence of a non-permitted event), or from the non-fulfilment of an obligation.

An institution ($\mathcal{I}$) is represented as a set of *institutional facts* or fluents ($\mathcal{F}$) that evolves over time as a result of the occurrence of exogenous events which are interpreted in the institutional context. These fluents are either true (if present) or considered false (if absent) at a given time instant. Cliffe et al. further identify normative fluents that denote normative properties of the state such as 1. *permission* ($\mathcal{P}$) – which events may occur without causing a violation, 2. *power* ($\mathcal{W}$) – the capability to influence the institutional state, 3. *obligations* ($\mathcal{O}$) – a particular event is must happen before some other event (e.g. a timeout) otherwise a specific violation is generated, and 4. *domain fluents* ($\mathcal{D}$) that correspond to properties specific to the normative framework itself.

Changes in the institutional state are achieved through the definition of two relations: (i) the generation relation ($\mathcal{G}$), which implements counts-as by specifying how the occurrence of one (exogenous or institutional) event generates another (institutional) event, subject to the empowerment of the actor and the conditions on the state, and (ii) the consequence relation ($\mathcal{C}$), which specifies the initiation and termination of fluents, subject to the occurrence of some event under certain conditions on the institutional state.

The semantics of an institution is defined over a sequence, called a *trace*, of observed $\mathcal{E}_{ex}$. Starting from the initial state (Δ), each exogenous event is responsible for a state change, through initiation and termination of fluents. This is achieved by a three-step process: (i) the transitive closure of $\mathcal{G}$ with respect to a given exogenous event determines all the generated events, (ii) to this all violations of non-permitted events and non-fulfilled obligations are added, giving the set of all events whose consequences determine the new state, (iii) the application of $\mathcal{C}$ to this set of events identifies all fluents that are initiated and terminated with respect to the current state, so determining the next state.

For each trace, we can compute a sequence of states that constitutes the model of the institutional framework for that trace. For ease of reference, a brief summary of the institutional model is given in Figure 1(a).

Computational Model The formal model of an institution can be translated to an equivalent computational model using Answer set programming (ASP) [9]. ASP is a declarative programming paradigm using logic programs under the answer set semantics. A variety of programming languages for ASP exists. There are several efficient solvers for $AnsProlog$ and like all declarative languages has the advantage of describing the constraints and the solutions rather than writing algorithm to find the solutions to the problem.

$\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{G}, \mathcal{C}, \Delta \rangle$, where

1. $\mathcal{F} = \mathcal{W} \cup \mathcal{P} \cup \mathcal{O} \cup \mathcal{D}$
2. $\mathcal{G} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{E}_{inst}}$
3. $\mathcal{C} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{F}} \times 2^{\mathcal{F}}$ where $C(\phi, e) =$ $(\mathcal{C}^{\uparrow}(\phi, e), \mathcal{C}^{\downarrow}(\phi, e))$ where
   (i) $\mathcal{C}^{\uparrow}(\phi, e)$ initiates fluents
   (ii) $\mathcal{C}^{\downarrow}(\phi, e)$ terminates fluents
4. $\mathcal{E} = \mathcal{E}_{ex} \cup \mathcal{E}_{inst}$ with $\mathcal{E}_{inst} = \mathcal{E}_{act} \cup \mathcal{E}_{viol}$
5. Δ
6. State Formula: $\mathcal{X} = 2^{\mathcal{F} \cup \neg \mathcal{F}}$

(a)

$$p \in \mathcal{F} \Leftrightarrow \texttt{ifluent(p)}. \quad (1)$$
$$e \in \mathcal{E} \Leftrightarrow \texttt{event(e)}. \quad (2)$$
$$e \in \mathcal{E}_{ex} \Leftrightarrow \texttt{evtype(e,obs)}. \quad (3)$$
$$e \in \mathcal{E}_{act} \Leftrightarrow \texttt{evtype(e,act)}. \quad (4)$$
$$e \in \mathcal{E}_{viol} \Leftrightarrow \texttt{evtype(e,viol)}. \quad (5)$$
$$\mathcal{C}^{\uparrow}(\phi, e) = P \Leftrightarrow \forall p \in P \cdot \texttt{initiated(p,T)} \leftarrow \mathit{occurred}(e, T), EX(\phi, T). \quad (6)$$
$$\mathcal{C}^{\downarrow}(\phi, e) = P \Leftrightarrow \forall p \in P \cdot \texttt{terminated(p,T)} \leftarrow \mathit{occurred}(e, T), EX(\phi, T). \quad (7)$$
$$\mathcal{G}(\phi, e) = E \Leftrightarrow g \in E, \texttt{occurred(g,T)} \leftarrow \texttt{occurred(e,T)}, \texttt{holdsat(pow(e),I)}, EX(\phi, T). \quad (8)$$
$$p \in \Delta \Leftrightarrow \texttt{holdsat(p,i00)}. \quad (9)$$

(b)

Fig. 1. (a) Formal specification of the normative framework and (b) Translation of normative framework specific rules into *AnsProlog*

The basic components of the language are atoms, elements that can be assigned a truth value. An atom can be negated using *negation as failure*. *Literals* are atoms a or negated atoms not a. We say that not a is true if we cannot find evidence supporting the truth of a. Atoms and literals are used to create rules of the general form: $a \leftarrow b_1, ..., b_m, \text{not } c_1, ..., \text{not } c_n$, where a, b_i and c_j are atoms. Intuitively, this means *if all atoms b_i are known/true and no atom c_j is known/true, then a must be known/true*. We refer to a as the head and $b_1, ..., b_m, \text{not } c_1, ..., \text{not } c_n$ as the body of the rule. Rules with empty body are called *facts*. Rules with empty head are referred to as *constraints*, indicating that no solution should be able to satisfy the body. A *(normal) program (or theory)* is a conjunction of rules and is also denoted by a set of rules. The semantics of *AnsProlog* is defined in terms of *answer sets*, i.e. assignments of true and false to all atoms in the program that satisfy the rules in a minimal and consistent fashion. A program may have zero or more answer sets, each corresponding to a solution.

The mapping of an institution consists of three parts: a *base component* which is independent of the institutions being modelled, the *time component* and the *institution-specific component*. The base component deals with inertia of the fluents, the generation of violation events of non-permitted actions and unfulfilled obligations. Furthermore it terminates fulfilled and violated obligations. The time component defines the predicates for time and is responsible for generating a single observed event at every time instance. The mapping uses the following atoms: `ifluent(p)` to identify fluents, `evtype(e,t)` to describe the type of an event, `event(e)` to denote the events, `instant(i)` for time instances, `final(i)`

for the last time instance, `occurred(e, i)` to indicate that the (empowered institutional) event happened at time i, `observed(e, i)` that the (exogenous) event was observed at time i, `holdsat(p, i)` to state that the normative fluent p holds at i, and finally `initiated(p, i)` and `terminated(p, i)` for fluents that are initiated and terminated at i. Figure 1(b) provides the framework-specific translation rules, including the definition of all the fluents and events as facts. For a given expression $\phi \in \mathcal{X}$, we use the term $EX(\phi, T)$ to denote the translation of ϕ into a set of ASP literals of the form (`not`) `holdsat(f, T)`, denoting that some fluent `f` (does not hold) holds at time `T`, while the initial state of the normative framework is encoded as simple facts (`holdsat(f, i00)`).

2.2 Case Study: Virtual Communities

To demonstrate how conflicts can be detected when composing institutions we use, a (necessarily) simplified case study of virtual communities. In particular, we focus on detecting conflicts when composing a technical and a social institution which in combination serve to govern the virtual community from both a social and a technical perspective. In the specific example we look at handling the membership of users making inappropriate posts.

Table 1 provides the formal model of both the technical and the social institution.

From a policy perspective, the main conceptual difference between the two institutions is who is allowed and empowered to end the membership of a user who posted an inappropriate post. The technical institution states that only authorised persons (e.g. moderators) can have the right to remove the memberships of members having posted inappropriate content. By contrast,the social institution might prefer a more community-driven approach and allow all members to take the initiative of enforcing actions when inappropriate content is posted, by specifying that any member of the community can remove another members' membership after the content violation is confirmed by a moderator.

Both institutions have three observed events ($\mathcal{E}_{ex}$) to signal an agent posting, an agent posting inappropriately and the removal of the membership of an agent. The social institution has an additional event to indicate that an agent informs the moderator. Apart from the counts-as institutional actions and standard violations, we have one extra violation event to indicate the occurrence of `postViolation`. Domain fluents indicate who is a member, a moderator and who has posted inappropriately.

The main difference between the two institutions (and the cause for conflicts) is the initiation of fluents after the occurrence of the `inappropriatePost(Agent)` event. In the technical institution, the moderators are given the permission and the power to remove the membership of the `Agent`. In the social one, `inappropriatePost` triggers the initiation of the domain fluent `inappPost` for the offending agent and the permission to inform the moderator is granted to all participants. Upon informing the moderator (`informModerator`), the informing agent receives the permission and power to remove the offending agent from the

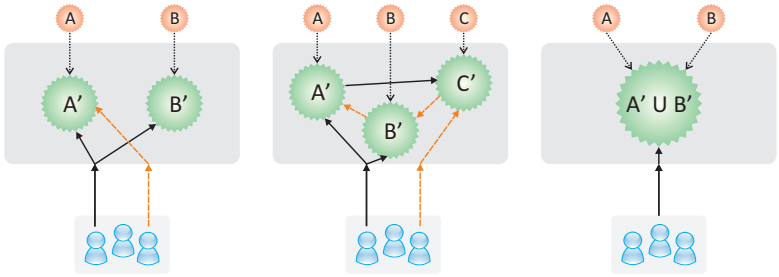


Fig. 2. Three views of composition. *(a):* composite institution. *(b):* Multi-institution. *(c):* merged institution.

community Termination of membership terminates permission to post. The generation function generates the `inappropriatePost(Agent)` when needed and connects the physical world with the institutional contexts. At the start, all agents are given permission and power to post to the community.

Using the translation to *AnsProlog* , Table 2 compares the `removeMembership` policy of both institutions.

3 Composite Institutions

3.1 Combining Institutions

Having presented the single institutions, we now address the issue on how institutions can be combined. The literature suggests there are three ways of combining institutions, which we depict in Fig. 2. The A', B', C' and $A' \cup B'$ indicate the consistent states of the corresponding institutions or their consistent union. The circles around the institutions indicate their state. The arrows indicate triggered events and changes to the state as a result of the actions of two participants, differentiated by the use of solid and dashed lines.The options are as follows: (a) Composite institution: The first option is to treat all the individual institutions as separate individual entities which – if required – have been adjusted to handle policy conflicts. The composite institution thus is not a new institution, but rather a shared governance scope of the individual institutions. (b) Multi-institution: The second option offers a hierarchical structure of several interlinked institutions where one institutional change is triggered by another institution changing state and only influences the virtual community indirectly. The connected institutions have to be adapted to be avoid conflicts. [2] proposes multi-institutions but assumes that designers have avoided the possibility of conflicts. (c) Merged institution: Finally, it is also possible to join the policies all institutions into one "super-institution". In contrast to the first case, the focus is not on maintaining the autonomy of the initial individual institutions, but to create a completely new institution which becomes the interface between the actors in the system.

Technical Institution	Social Institution
$\mathcal{E}_{ex} = \{\texttt{post(Agent)}, \texttt{inappropriatePost(Agent)}, \texttt{removeMembership(Agent, Agent)}\}$	$\mathcal{E}_{ex} = \{\texttt{post(Agent)}, \texttt{inappropriatePost(Agent)}, \}$ $\{\texttt{informModerator(Agent)}, \texttt{removeMembership(Agent, Agent)}\}$
$\mathcal{E}_{act} = \{\texttt{intPost(Agent)}, \texttt{intRemoveMembership(Agent, Agent)}\}$	$\mathcal{E}_{act} = \{\texttt{intPost(Agent)}, \texttt{intRemoveMembership(Agent, Agent)}\}$
$\mathcal{E}_{viol} = \{\text{viol}(\texttt{post(Agent)}), \text{viol}(\texttt{inappropriatePost(Agent)}), \text{viol}(\texttt{removeMembership(Agent, Agent)})\}$ $\{\text{viol}(\texttt{intPost(Agent)}), \text{viol}(\texttt{intRemoveMembership(Agent, Agent)})\}$ $\{\texttt{postViolation(Agent)}\}$	$\mathcal{E}_{viol} = \{\text{viol}(\texttt{post(Agent)}), \text{viol}(\texttt{inappropriatePost(Agent)}), \text{viol}(\texttt{removeMembership(Agent, Agent)})\}$ $\{\text{viol}(\texttt{intPost(Agent)}), \text{viol}(\texttt{intRemoveMembership(Agent, Agent)})\}$ $\{\texttt{postViolation(Agent)}, \text{viol}(\texttt{informModerator(Agent)})\}$
$\mathcal{D} = \{\texttt{isModerator(Agent)}, \texttt{isMember(Agent)}\}$	$\mathcal{D} = \{\texttt{inappPost(Agent)}, \texttt{isMember(Agent)}\}$
$\mathcal{W} = \{\text{pow}(\texttt{intPost(Agent)}), \text{pow}(\texttt{intRemoveMembership(Agent, Agent)})\}$	$\mathcal{W} = \{\text{pow}(\texttt{intPost(Agent)}), \text{pow}(\texttt{intRemoveMembership(Agent, Agent)})\}$
$\mathcal{P} = \{\text{perm}(\texttt{post(Agent)}), \text{perm}(\texttt{inappropriatePost(Agent)}), \text{perm}(\texttt{removeMembership(Agent, Agent)})\}$ $\{\text{perm}(\texttt{intPost(Agent)}), \text{perm}(\texttt{intRemoveMembership(Agent, Agent)})\}$	$\mathcal{P} = \{\text{perm}(\texttt{post(Agent)}), \text{perm}(\texttt{inappropriatePost(Agent)}), \text{perm}(\texttt{removeMembership(Agent, Agent)})\}$ $\{\text{perm}(\texttt{intPost(Agent)}), \text{perm}(\texttt{intRemoveMembership(Agent, Agent)})\}$
$\mathcal{O} = \emptyset$	$\mathcal{O} = \emptyset$
$\mathcal{C}^{\uparrow}(\mathcal{X}, \mathcal{E})$: $\langle\{\texttt{isModerator(Agent1)}, \texttt{isMember(Agent)}\},$ $\texttt{inappropriatePost(Agent)}\rangle \rightarrow \{\text{perm}(\texttt{removeMembership(Agent, Agent1)}),$ $\text{perm}(\texttt{intRemoveMembership(Agent, Agent1)}),$ $\text{pow}(\texttt{intRemoveMembership(Agent, Agent1)})\}$	$\mathcal{C}^{\uparrow}(\mathcal{X}, \mathcal{E})$: $\langle\{\texttt{isMember(Agent1)}\},$ $\texttt{inappropriatePost(Agent)}\rangle \rightarrow \{\text{perm}(\texttt{informModerator(Agent1)})\}$ $\langle\{\texttt{isMember(Agent1)}, \texttt{inappPost(Agent)}\},$ $\texttt{informModerator(Agent1)}\rangle \rightarrow \{\text{perm}(\texttt{removeMembership(Agent, Agent1)}),$ $\{\text{perm}(\texttt{intRemoveMembership(Agent, Agent1)}),$ $\text{pow}(\texttt{intRemoveMembership(Agent, Agent1)})\}$ $\langle\emptyset, \texttt{inappropriatePost(Agent)}\rangle \rightarrow \{\texttt{inappPost(Agent)}\}$
$\mathcal{C}^{\downarrow}(\mathcal{X}, \mathcal{E})$: $\langle\{\texttt{isModerator(Agent1)}\},$ $\texttt{removeMembership(Agent, Agent1)}\rangle \rightarrow \{\texttt{isMember(Agent)}, \text{perm}(\texttt{post(Agent)}),$ $\text{perm}(\texttt{inappropriatePost(Agent)}),$ $\text{pow}(\texttt{intPost(Agent)}), \text{perm}(\texttt{intPost(Agent)})\}$	$\mathcal{C}^{\downarrow}(\mathcal{X}, \mathcal{E})$: $\langle\{\texttt{isModerator(Agent1)}\},$ $\texttt{removeMembership(Agent, Agent1)}\rangle \rightarrow \{\texttt{isMember(Agent)}, \text{perm}(\texttt{post(Agent)}),$ $\text{perm}(\texttt{inappropriatePost(Agent)}),$ $\text{pow}(\texttt{intPost(Agent)}), \text{perm}(\texttt{intPost(Agent)})\}$
$\mathcal{G}(\mathcal{X}, \mathcal{E})$: $\langle\{\texttt{isMember(Agent)}\}, \texttt{post(Agent)}\rangle \rightarrow \{\texttt{intPost(Agent)}\}$ $\langle\emptyset, \texttt{inappropriatePost(Agent)}\rangle \rightarrow \{\texttt{postViolation(Agent)}\}$ $\langle\{\texttt{isModerator(Agent1)}, \texttt{isMember(Agent)}\},$ $\texttt{removeMembership(Agent, Agent1)}\rangle \rightarrow \{\texttt{intRemoveMembership(Agent, Agent1)}\}$	$\mathcal{G}(\mathcal{X}, \mathcal{E})$: $\langle\{\texttt{isMember(Agent)}\}, \texttt{post(Agent)}\rangle \rightarrow \{\texttt{intPost(Agent)}\}$ $\langle\emptyset, \texttt{inappropriatePost(Agent)}\rangle \rightarrow \{\texttt{postViolation(Agent)}\}$ $\langle\{\texttt{isModerator(Agent1)}, \texttt{isMember(Agent)}\},$ $\texttt{removeMembership(Agent, Agent1)}\rangle \rightarrow \{\texttt{intRemoveMembership(Agent, Agent1)}\}$
$\Delta = \text{perm}(\texttt{post(Agent)}), \text{perm}(\texttt{inappropriatePost(Agent)}), \text{pow}(\texttt{intPost(Agent)}),$ $\text{perm}(\texttt{intPost(Agent)}), \texttt{isModerator(moderator)}, \texttt{isModerator(agent1)},$ $\texttt{isMember(agent1)}, \texttt{isMember(agent2)}$	$\Delta = \text{perm}(\texttt{post(Agent)}), \text{perm}(\texttt{inappropriatePost(Agent)}), \text{pow}(\texttt{intPost(Agent)}),$ $\text{perm}(\texttt{intPost(Agent)}), \texttt{isMember(agent1)}, \texttt{isMember(agent2)}$

Table 1. Social and Technical Institutions

	Technical Policies	Social Policies
Permission of removeMembership	$\texttt{initiated(perm(removeMembership(agent1, agent2)), I)} :-$ $\texttt{occurred(inappropriatePost(agent1), I)},$ $\texttt{holdsat(isMember(agent1), I)},$ $\texttt{holdsat(isModerator(agent2), I)},$ $\texttt{holdsat(live(virtualcommunitytechnicalspecs), I)},$ $\texttt{instant(I)}.$	$\texttt{initiated(perm(removeMembershipRE(agent1, agent2)), I)} :-$ $\texttt{occurred(informModeratorRE(agent2), I)},$ $\texttt{holdsat(isMemberRE(agent2), I)},$ $\texttt{holdsat(inappPostRE(agent1), I)},$ $\texttt{holdsat(live(virtualcommunitysocialspecs), I)},$ $\texttt{instant(I)}.$
Permission of intRemoveMembership	$\texttt{initiated(perm(intRemoveMembership(agent1, agent2)), I)} :-$ $\texttt{occurred(inappropriatePost(agent1), I)},$ $\texttt{holdsat(isModerator(agent2), I)},$ $\texttt{holdsat(isMember(agent1), I)},$ $\texttt{holdsat(live(virtualcommunitytechnicalspecs), I)},$ $\texttt{instant(I)}.$	$\texttt{initiated(perm(intRemoveMembershipRE(agent1, agent2)), I)} :-$ $\texttt{occurred(informModeratorRE(agent2), I)},$ $\texttt{holdsat(isMemberRE(agent2), I)},$ $\texttt{holdsat(inappPostRE(agent1), I)},$ $\texttt{holdsat(live(virtualcommunitysocialspecs), I)},$ $\texttt{instant(I)}.$
Empowerment of intRemoveMembership	$\texttt{initiated(pow(intRemoveMembershipRE(agent1, moderator)), I)} :-$ $\texttt{occurred(informModeratorRE(moderator), I)},$ $\texttt{holdsat(isMemberRE(moderator), I)},$ $\texttt{holdsat(inappPostRE(agent1), I)},$ $\texttt{holdsat(live(virtualcommunitysocialspecs), I)},$ $\texttt{instant(I)}.$	$\texttt{initiated(pow(intRemoveMembership(agent1, agent2)), I)} :-$ $\texttt{occurred(inappropriatePost(agent1), I)},$ $\texttt{holdsat(isMember(agent1), I)},$ $\texttt{holdsat(live(virtualcommunitytechnicalspecs), I)},$ $\texttt{instant(I)}.$

Table 2. Comparison of Social and Technical Policies in *AnsProlog*

In this paper, we are interested in the first type, namely composite institutions. A set $\mathcal{I} = \{\mathcal{I}_1, \ldots, \mathcal{I}_n\}$ of institutions are combined as the composite institution $C_{\mathcal{I}}$. The institutions in $C_{\mathcal{I}}$ do not share state nor are they able to interact with each other (as is the case in multi-institutions).

While provided by different designers, we will assume for the sake of this paper that the individual institutions will use the same terminology to the same concepts. In other words, we assume that the institutions are semantically aligned. From our virtual community example, it should for example not happen that one institution uses the event `post(Agent)` to denote the a message is posted by `Agent` while another institution uses the event `postMessage(Agent)` to refer to same action.

3.2 Composite Traces

While a composite institution is not a institution in its own right (i.e. it does not have its own formal model), the participants can interact with it as if it were one, as shown in Figure 2. The composite institution provides a wrapper around the individual institutions allowing the participants to interact with one entity rather than having to determine which of the individual institutions they wish to interact with. In reality, the composition passes the exogenous events to the correct corresponding individual institutions if appropriate.

To be able to analyse the behaviour of the composition, we introduce the notion of a composite trace. Composite traces are sequences of events created from the observed events of the individual institutions.

Definition 1. *Given an composite institution $C_{\mathcal{I}}$ consisting of institutions $\mathcal{I} = \mathcal{I}_1, \ldots, \mathcal{I}_n$. A* composite trace *is a sequence $\langle e_1, \ldots e_m \rangle$ such that $\forall e_i, 1 \leq i \leq m : \exists 1 \leq j \leq n : e_i \in \mathcal{E}^j_{ex}$.*

3.3 Null Events

The composite traces capture the state transitions of each of the individual institutions. From the composite trace, we can separate the traces for each of the individual institutions by selecting those exogenous events that each recognizes. Unfortunately these individual traces could be of different lengths. This means that the states will be associated with different time steps, making it impractical, if not impossible, to reason about institutions in the same "time" frame.

To facilitate the temporal alignment of events, we extend each institution's formal model with a null event. The occurrence of the observed null event does not change the state (i.e. the null event is not used in either $\mathcal{G}$ or $\mathcal{C}$) but when incorporated into the model, it advances the state counter. The null event also needs to be permitted from the start. So for each institution $\mathcal{I}_i \in C_{\mathcal{I}}$ with $\mathcal{I} = \{\mathcal{I}_1, \ldots, \mathcal{I}_n\}$ we have that $e_{null} \in \mathcal{E}^i_{ex}$ and perm($\mathtt{e_{null}}$) $\in \Delta$.

With the addition of null events, we can define synchronised traces by replacing each unknown observed event in an institution's trace by the null-event.

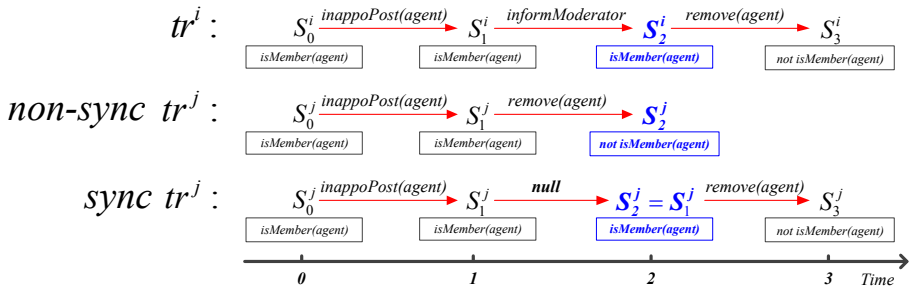


Fig. 3. Example of Trace Synchronization

Definition 2. *Given a composite trace* $CTR = \langle e_1, \ldots, e_t \rangle$ *for a composite institution* $C_{\mathcal{I}}$ *and an institution* $\mathcal{I}_i \in \mathcal{I}$*, the* synchronised trace *for* $\mathcal{I}_i$ *w.r.t. CRT is the trace* $\langle a_1, \ldots, a_t \rangle$ *with* $a_k = e_k$ *if* $e_k \in \mathcal{E}^i_{ex}$ *and with* $a_k = e_{null}$ *otherwise.*

These synchronised traces are needed for conflict detection. We demonstrate this using our virtual communities example. Suppose a composite trace $CTR = \langle$`inappropriatePost(agent1)`, `informModerator(agent2)`, `removeMembership(agent1, agent2)` $\rangle$. As mentioned in Section 2.2, `informModerator(agent2)` is only observable by the social institution i and not the technical institution j. The separate event traces (tr) for institutions i and j are as below:

$$tr_i = \langle \texttt{inappropriatePost(agent1)}, \texttt{informModerator(agent2)}, \texttt{removeMembership(agent1, agent2)} \rangle$$
$$tr_j = \langle \texttt{inappropriatePost(agent1)}, \texttt{removeMembership(agent1, agent2)} \rangle$$

where the event `informModerator` is missing from the trace of the technical institution j. We assume that `agent2` is a moderator. The states transitions without synchronisation are shown as the first two traces in Fig.3. At time instant 2, both institutions disagree on the membership of agent. We call this a conflict. However, this kind of conflict is not desirable because they are actually caused by the asynchronous occurrence of events. By synchronising the event traces, as shown in the last trace in Fig.3, a null event `null` is generated for the trace tr^j at the time of the occurrence of event `informModerator(agent2)`. The states S^j_1 and S^j_2 are identical, as the null event does not effect the state of j. Consequently we have removed the false conflict on the fluent `isMember(agent)` by synchronising the event traces.

Having defined composite and synchronised traces, we can now define a composite model.

Definition 3. *Given a composite trace CTR for a composite institution* $C_{\mathcal{I}}$ *with* $\mathcal{I} = \{\mathcal{I}_1, \ldots, \mathcal{I}_n\}$*, the corresponding composite model is the set of models* M_i *with* $1 \leq i \leq n$ *such that* M_i *is the model corresponding to the synchronised trace of institution* i*.*

Hence a composite model is a set of sequences over states. Each such sequence contains the state transitions for a synchronised trace obtained from CTR. We refer to the models corresponding to synchronised traces as synchronised models.

4 Conflict Detection

4.1 Conflict Traces

As noted in the introduction, institutions are typically designed to fulfill individual normative goals and are initially expected to work independently. Therefore, composing institutions is likely to cause conflicts between the policies of the individual institutions and causing problems for agents when they interact with the composite institution. Hence the importance of resolving conflicts between individual institutions when creating the composite institution. We start with our definition of conflict w.r.t. composite institutions. While the institutional model is event-based, the conflicts can be detected in the state. We say that a composite institution is in conflict if there exists a composite trace for which two corresponding synchronised models are inconsistent. Inconsistency occurs when in corresponding states a fluent is true in one and false in the other.

Definition 4. *Given a composite institution $C_{\mathcal{I}}$ with a composite trace CTR. CTR is a* conflict trace *iff:*

- $\exists \mathcal{I}_i, \mathcal{I}_j \in \mathcal{I}$ *with synchronised models* $M_i = \langle S_0^i, \ldots S_t^i \rangle$ *and* $M_j = \langle S_0^j, \ldots S_t^j \rangle$ *such that*
- $\exists f \in (\mathcal{F}^i \cap \mathcal{F}^j)$ *such that*
- $\exists k, 0 \leq k \leq t$ *such that*
- $f \in S_k^i$ *and* $\neg f \in S_k^j$

Definition 5. *A composite institution is* conflict-free *iff it does not admit any conflict traces.*

4.2 Detection of Policy Conflict

With all the theory in place, we can now discuss the computational mechanism for detecting conflicts of composite definitions. As mentioned in Section 2.1, we use *AnsProlog* for implementing the computational model of institutions. As a result we can use the same technique for determining conflict traces and detecting conflicts. Instead of answer sets representing all observed traces, we want our answer sets to represent conflict traces, i.e. composite traces and their models that produce a conflict.

For simplicity, we will only introduce our method for two institutions being composed but the method can be extended to as many institutions as desired.

At first sight, one might think that simply putting together the *AnsProlog* implementations of the individual institutions and the constraints for selecting the conflict traces would be sufficient. Unfortunately, since we are looking for inconsistency this impossible. The constraints would be $\texttt{conflict} :- \texttt{holdsat}(\texttt{F}, \texttt{T})$,

not holdsat(F, T). and : −not conflict. Regrettably, this will never result in an answer being returned. The first rule will never hold, so conflict will never be true and therefore the constraint cannot be satisfied. To solve this problem, the events and fluents of one of the two institutions are renamed (this can be done automatically by for example adding RE to each fluent or event) and adding a set of facts of the form rename(F, FRE) to the program to indicate that F and FRE are actually the same. For example, post(Agent) becomes postRE(Agent). This allows us to express the conflict selection in *AnsProlog* as follows. To indicate conflict we introduce, for efficiency reasons, two conflict atoms, one with no arguments and one with three arguments. We use two rules to take into account that the positive occurrence of the conflicting fluent can either occur in the renamed and non-renamed institution. A constraint is used to express that we are only interested in answer sets representing conflict traces. We also introduce two rules for information purposes, using the atom conflict/3. The first of its arguments gives the fluent that appears positively, the second the negative fluent and the third argument the time instance the conflict occurs. This information can be used later to resolve the conflict.

```
conflict :− holdsat(F, I), not holdsat(FRE, I), rename(F, FRE),
            instant(I).
conflict :− holdsat(FRE, I), not holdsat(F, I), rename(F, FRE),
            instant(I).
         :− not conflict.
conflict(F, FRE, I) :− holdsat(F, I), not holdsat(FRE, I), rename(F, FRE),
                       instant(I).
conflict(FRE, F, I) :− holdsat(FRE, I), not holdsat(F, I), rename(F, FRE),
                       instant(I).
```

Individual institutions generate their traces by enforcing that each answer set admits one and only observed atom for each time instance. With observed events possibly being recognised by two institutions, we need two observed events one for the original event and one for its renamed counterpart. So we replicate the original observed event code using a new atom called compObserved.

```
compEvent(E) :− evtype(ERE, ex), evinst(E, In), rename(E, ERE), instRE(In).
compEvent(E) :− evtype(E, ex), evinst(E, In), inst(In).
{compObserved(E, I)} :− compEvent(E), instant(I), not final(I).
ev(I) :− compObserved(E, I), instant(I).
      :− not ev(I), instant(I), not final(I).
      :− compObserved(E1, I), compObserved(E2, I), E1! = E2,
         instant(I), compEvent(E1), compEvent(E2).
```

The first two rules make sure that the observable events of both institutions in their original form can be considered in a composite trace. The third rule generates `compObserved/2` when needed. If generated a matching `ev/1` atom is provided. The first constraint guarantees that least one `ev` is produced for any non-final time instant. The second constraints makes sure it is only one.

These `compObserved` atoms will form our composite traces from which we can generate the synchronised traces using the `observed` atoms. Since the `compObserved` events use the non-renamed version we need to make sure that they are renamed when needed. This is done by the program below:

```
observed(ERE, I) :- compObserved(E, I), rename(E, ERE), evinst(ERE, In),
                    instJ(In), instant(I).
observed(E, I) :- compObserved(ERE, I), rename(E, ERE), evinst(E, In),
                  instI(In), instant(I).
observed(enull, I) :- compObserved(ERE, I), rename(E, ERE), not evinst(E, Inst),
                      instI(Inst), instI(In), instant(I).
observed(enullRE, I) :- compObserved(E, I), rename(E, ERE), not evinst(ERE, Inst),
                        instJ(Inst), instant(I).
```

4.3 Conflicts in the case Study

We can now apply our conflict detection mechanism to our use case. Due to space limitation, we only present the conflicts detected at time 3 for the composite trace: $CTR = \langle$`inappropriatePost(agent1)`, `informModerator(agent2)`, `removeMembership(agent1, agent2)`$\rangle$. Other traces give the same conflicts, but only at different time instances.

The first type of conflicts concerns the permission to remove membership: it indicates that `agent2`, as a normal member, is permitted to remove the membership of others (e.g. `agent1`), under the rules of the social institution, however, this permission is not given with regard to the technical institution, which only allows a `moderator` agent to remove membership.

```
conflict( perm(removeMembershipRE(agent1, agent2)),
          perm(removeMembership(agent1, agent2)), 3)
conflict( perm(removeMembership(agent1, moderator)),
          perm(removeMembershipRE(agent1, moderator)), 3)
```

The second type of conflict concerns the permission to generate the institutional event `intRemoveMembership/2`. Similarly, this permission for a normal member to remove membership only appears in the social institution, while only

the moderator is allowed to remove membership in the technical institution.

```
conflict( perm(intRemoveMembershipRE(agent1, agent2)),
          perm(intRemoveMembership(agent1, agent2)), 5)
conflict( perm(intRemoveMembership(agent1, moderator)),
          perm(intRemoveMembershipRE(agent1, moderator)), 5)
```

The third type of conflict occurs with the empowerment of the institutional event `intRemoveMembership/2`. The event is empowered under different conditions.

```
conflict( pow(intRemoveMembershipRE(agent1, agent2)),
          pow(intRemoveMembership(agent1, agent2)), 5)
conflict( pow(intRemoveMembershipRE(agent1, moderator)),
          pow(intRemoveMembership(agent1, moderator)), 5)
```

The results presented above show all the expected conflicts between the social institution and technical institution. In contrast to the false conflict on fluent `isMember(agent)` in Fig. 3 caused by the asynchronism of event traces, these detected conflicts are the result of the different goals of the independent component institutions.

5 Summary and Conclusions

When composing institutions written by different designers with possible (slightly) different objectives, conflicts between the modelled policies can occur. In this paper, we presented a mechanism for detecting this, occasionally subtle, differences. Our mechanism is based on the institutional model and *AnsProlog* implementation by Cliffe et al. Based on this existing institutional framework, in this paper we firstly presented our notion of composite institutions and described how conflicts can be detected in them. We applied our approach to a case-study of a socio-technical system – namely a virtual community – in which conflicts between a social and a technical institution needed to be determined for the effective functioning of this community.

In this paper, we only focused on the detection of conflicts. An obvious next step is the resolution of the detected conflicts. This poses several interesting challenges such as including mechanisms to specify which policies or institutions are given a priority in case of conflict; as well as the question to trace back to which policies cause the conflicts we detected. By definition, our mechanism detects conflicts as a result of conflicting fluents (a fluent being true and false at the same time in different institutions). The fluents however are only the result of the exogenous events and the interpretation of those events in the institution through their policies. Thus, we need to determine which policies caused conflicts. One possible avenue for adjusting individual institutions is to use the conflict information as a use-case that can be presented to an inductive

learner. In [5], the authors use inductive learning to refine institutions which they call normative frameworks. Another promising future direction is the extension of the current model of composite institution to multi-institution and merged institution.

References

1. D. C. Andrews. Audience-specific online community design. *Communications of the ACM*, 45(4):64–68, April 2002.
2. O. Cliffe. *Specifying and Analysing Institutions in Multi-agent Systems Using Answer Set Programming*. PhD thesis, University of Bath, 2007.
3. O. Cliffe, M. De Vos, and J. A. Padget. Answer set programming for representing and reasoning about virtual institutions. In K. Inoue, K. Satoh, and F. Toni, editors, *Computational Logic in Multi-Agent Systems*, volume 4371 of *Lecture Notes in Computer Science*, pages 60–79. Springer Berlin / Heidelberg, 2007.
4. O. Cliffe, M. De Vos, and J. A. Padget. Specifying and reasoning about multiple institutions. In *Coordination, Organizations, Institutions, and Norms in Agent Systems II*, volume 4386 of *LNAI*, pages 67–85. Springer Berlin / Heidelberg, 2007.
5. D. Corapi, A. Russo, M. D. Vos, J. A. Padget, and K. Satoh. Normative design using inductive learning. *TPLP*, 11(4-5):783–799, 2011.
6. A. de Moor and J. Wagenvoort. Conflict management in an online gaming community. In *Proceedings of the Community Informatics Research Network (CIRN) 2004 Conference*, 2004.
7. V. Dignum. *A Model for Organizational Interaction: Based on Agents, Founded in Logic*. PhD thesis, Utrecht University, 2003.
8. M. Esteva. *Electronic Institutions: from specification to development*. PhD thesis, Insitut d'Investigació en Intelligència Artificial, 2003.
9. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3–4):365–386, 1991.
10. J. F. Hübner, J. S. Sichman, and O. Boissier. Developing organised multiagent systems using the $moise^+$ model: programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3/4):370–395, December 2007.
11. R. Kling, G. McKim, J. Fortuna, and A. King. Scientific collaboratories as socio-technical interaction networks: A theoretical approach. In *Americas Conference on Information Systems*, 2000.
12. P. Noriega. *Agent mediated auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autònoma de Barcelona, 1997.
13. D. C. North. Institutions matter. Economic History 9411004, EconWPA, 1994.
14. E. Ostrom. *Governing the Commons: the Evolution of Institutions for Collective Action*. Cambridge University Press, 1990. 18th printing (2006).
15. J. Preece. Supporting community and building social capital. *Communications of the ACM*, 45(4):37–39, 2002.
16. H. Rheingold. *The Virtual Community: Homesteading on the Electronic Frontier*. MIT Press, revised edition edition, 2000.
17. J. R. Searle. *The Construction of Social Reality*. Free Press, 1995.
18. J. Vázquez-Salceda. *The role of norms and electronic institutions in multi-agent systems applied to complex domains. The HARMONIA framework*. PhD thesis, Technical University of Catalonia, 2003.
19. G. Vickers. Values, norms and policies. *Policy Science*, 4:103–111, 1973.