Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 181 (2007) 5–17
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Role-based Architectural Modelling of Socio-Technical Systems

Osama El-Hassan and José Luiz Fiadeiro

Department of Computer Science
University of Leicester
University Road, Leicester LE1 7RH, UK
oehe1,jose@mcs.le.ac.uk

Abstract

The focus of this paper is on how a role-based architectural approach can contribute to building normative models for evolvable and adaptable socio-technical systems, i.e. systems in which both software components and people play well-defined roles and need to interact to ensure that required global properties emerge. We propose a method that is associated with a set of new modelling primitives anchored on organisational roles and governed by social laws that handle the situations that may arise when the people involved deviate from prescribed behaviour and fail to play the role that they have been assigned as entities of the system.

Keywords: Role-based Modelling, Software Architecture, Coordination.

1 Introduction

Software technology is gradually increasing its scope from the core technical implementation of required functionalities to include processes and people who interact with the implemented systems. The term "socio-technical system" was coined [8] to refer to systems that incorporate a "social" dimension in the sense that people (or groups of people) need to be considered not as external users but as another class of components that, together with software and devices, perform roles that are key for the "good behaviour" of the whole system. In other words, interactions with people need to be brought inside the system, especially because the boundaries between social and software components may vary throughout its lifetime; tasks performed by humans can be (partially) replaced by or even shared with software applications, depending on the context of execution. The main problem that needs to be addressed when developing such socio-technical systems is that interactions between social and software components, although governed by organizational rules and policies, may affect the whole system behaviour in a way that cannot be totally predicted, let alone programmed.

doi:10.1016/j.entcs.2007.01.050

In other words, social components cannot be designed, as software and mechanical/hardware entities can, to comply with system rules; instead, they constitute what Michael Jackson calls a "biddable domain" [14]: they can be "*joined to adhere to a certain behaviour, but may or may not obey the injunction*". As a result, in systems in which social entities interact with software, one needs to anticipate what violations can take place so that software can be programmed to react to non-normative situations in ways that ensure agreed, possibly minimal, levels of service.

Sub-ideal situation is a state which can be reached because an obligation or a permission has been violated. Handling non-normative or sub-ideal situations in socio-technical systems needs to be treated as a first-class concern. Procedures that deal with such situations should not be buried in the code of the components as this would make it impossible to separate what in the software is implementing the functionality required of component services and what is handling violations of organisational norms. This separation is essential because the procedures for handling non-normative situations often depend on the role that social entities play within the larger system and can change as the organisation evolves in ways that are independent of its core business requirements. This suggests that such aspects should be modelled explicitly as a separate architectural dimension of systems.

The approach to socio-technical systems that we propose exploits and extends software architecture techniques originally developed by Andrade et. al. [3] in what has become known as the CCC-model (Coordination/Computation/ Configuration). This model promotes the externalisation of the mechanisms that are responsible for coordinating interactions within the system from the computations that are performed locally in the components and ensure required functionalities. However, the CCC-model is not equipped with primitives through which one can model the behaviour of human entities, and distinguish between normative behaviour and sub-ideal situations that may arise from violation of norms. The purpose of this paper is, precisely, to put forward a method and associated semantic primitives that enrich the CCC-model to address collaborative activities within organisational settings.

Modelling collaboration exhibits traits of complexity that are not found in coordination-based models like CCC. The major source of difficulty in collaboration modelling is the fact that interactions with people are part of the functional behaviour. This incurs the necessity of a new level of run-time reconfigurability in terms changing interactions policies and/or the architectural structure to ensure required level of quality. Our approach models organisational norms on expected human behaviour in terms of another class of connector types (social laws) defined over a set of social roles, each of which represents the abilities of a social entity within the organisation.

This paper is organised as follows. Section 2 gives an overview of related research areas from which we borrowed several concepts and techniques. Section 3 introduces the new architectural modelling primitives, with a special emphasis on our use of role-based modelling. Finally, section 4 outlines aspects of our approach that are not covered in this paper.

2 Background

The new architectural modelling primitives — social laws and roles — that we propose take into account research in the area of normative systems [15,20] and role-based modelling [16], including recent work on Role-Based Access Controls (RBAC) and the role model introduced for policy-based management systems [18,23]. The separation of social and technical concerns was itself inspired in the work of Michael Jackson and colleagues on Problem Frames [14]. In this section, we provide a quick overview of the contribution of some of these areas to our approach.

2.1 The CCC Architecture

In "Software Architecture" [7,22], modelling techniques have been proposed for supporting interaction-centric approaches. More precisely, such techniques promote interconnections to first-class citizens (architectural connectors) by separating the code that, in traditional approaches, is included in the components for handling the way they interact with the rest of the system, from the code that is responsible for the computation that is responsible for the services offered by the components.

The particular architectural approach that we propose to extend has been developed within an industry/academia partnership [1,2,3]; it models connector types through coordination laws that bring together a number of event-condition-action (ECA) rules, each of which coordinates the joint behaviour that a group of components (partners) needs to execute in reaction to a trigger generated by another component or outside the system. In the CCC approach partners, over which *coordinations laws* are instantiated, are represented by means of *coordination interfaces.* In the sense of [1], coordination laws constitute the connector concept while coordination interfaces represent the roles of connector types that must be instantiated with components when a law is to be activated on them. As an example, consider the coordination of the way an in-charge doctor interacts with a respiratory-control system within the premises of an emergency room:

```
coordination interface respiratory-control
partner type DEVICE
types a:pressure, d:DOCTOR
operations
          in-charge(d):Boolean
          verify():pressure
          decrease(a):  post verify() = old verify()-a
          increase(a):  post verify() = old verify()+a

coordination interface doctor-in-charge
partner type DOCTOR
types a:pressure
events
       plus(a)
       minus(a)

coordination law restricted-respiratory
partners d:  doctor-in-charge, r:  respiratory-control
types a:pressure
attributes min,max:pressure
rules
     when d.minus(a) and r.in-charge(d)
          with r.verify - a ≥ min
          do r.decrease(a)
     when d.plus(a) and r.in-charge(d)
          with r.verify + a ≤ max
          do r.increase(a)
```

The above example shows that a coordination law defines how a number of partners interact. The partners are not named: they are abstracted as coordination interfaces that define types of system entities in terms of operations that instance partners need to make available and events that need to be observed. For instance, a respiratory-control device is required to provide operations through which one can verify, increase or decrease the current pressure being administered, and also to check if a given doctor is in charge of the device. In the case of a doctor, we do not require any operation to be provided, just the ability to observe two kinds of events that correspond to requests to increase or decrease the pressure. These events may correspond to buttons that the doctor needs to press, or communicate to a voice-enabled device, or by other mean.

Each rule of the coordination law identifies, under the "when" clause, a trigger to which the instances of the law will react - a request by a doctor for an increase or decrease of the pressure. The trigger can be just an event observed directly over one of the partners or a more complex condition built from one or more events. Under the "with" clause, we include conditions (guards) that should be observed for the reaction to be performed: that the changes in the pressure keep it within the specified bounds and that the doctor has been put in charge of the device. If any of the conditions fails, the reaction is not performed and the occurrence of the trigger fails. Explicit mechanisms can be defined for handling such failures.

Typically, the actions used in coordination rules invoke operations provided by the partners as identified in coordination interfaces. In the case of interconnections among software components, formal techniques can be used to reason about the correctness of the behaviour that emerges from the interactions thus established, which assume that, if given preconditions apply, invoked operations are executed establishing given post-conditions. In more sophisticated languages and models, one can also take into account exception handling. In the case of embedded systems, software components interact with mechanical/hardware components; one assumes that the "operations" identified in coordination interfaces capture phenomena that can be triggered by software. In such cases, "correctness" depends on the fact that the plant that is being controlled by the software is not faulty.

However, there is no provision in the CCC approach, nor in any other architectural approach that we know, to model social interactions, i.e. situations in which people (social components) are requested to perform given operations. Notice that this is not the case of the interaction between the doctor and the respiratory-control modelled above: the doctor is not being requested to perform any operation. A social interaction would occur if the doctor was requested to increase or decrease the pressure when alerted by a monitoring system that immediate intervention was required. Additionally a doctor may deviate from the prescribed behaviour e.g. increasing or decreasing the pressure level beyond the organisationally agreed limits.

In the case of such social interactions, "correctness" criteria do not apply. On the one hand, people are not like mechanical/hardware components that can be replaced if they are faulty. On the other hand, they cannot be assumed to execute "operations" when requested or to adhere to pre/post-conditions even if they respond to notifications. In summary, one needs a richer model of interaction that can capture the fact that coordination in the presence of social components cannot be causal.

2.2 Role-based Modelling

The focus of our approach to socio-technical systems is on modelling the social impact of incorporating people within systems in terms of combining human autonomy, capabilities, and responsibilities using role models. Humans play roles within the context of the collaborative activities in which they participate. These roles derive a substantial part of their semantics from the organizational structures to which they belong as defined in [12].

Notions of role can be found in many research areas of Software Engineering. An area from which we have drawn inspiration is Role-Based Access Control (RBAC) as used for analyzing access demands for information systems; in this area, roles provide a way of identifying tasks, user behaviour and related attributes like accountability. A general definition of role has been given by Kristensen [15]: "*role of an object is a set of properties which are important for an object to be able to behave in a certain way expected by a set of other objects*". This definition falls short of describing the notion of roles in a dynamic environment where agents can be assigned and revoked powers to perform actions and authorities to access resources. The challenge becomes harder when the activation of such powers and authorities should depend on the current context within a certain organizational settings.

Another aspect that is essential for our approach is the ability to compose roles in a systematic way to model capabilities of social components that are independent of the organisational contexts in which they operate. This allows us to deal with roles as types over which explicit relationships can be defined to address organisational constraints without committing to specific component instances and construct hierarchies that represent the organisational structure. Using roles to define types of social components also allows us to model dynamic properties such as role enactment as actions that can trigger reconfigurations on the way roles are assigned.

The collaboration models that we have in mind for social interactions are composed of human users, communication media, and objects on which they act, e.g. software, hardware, and mechanical. In our approach, we abstract away the communication media and adopt Castelfranchi's approach to defining communicative actions as an instrument of Behavioural Implicit Communication (BIC) [9]. His approach is comparable but differs in its essence from the well-known utterance approach of John Searle' theory (Speech acts) [20]. Speech act theory provides both specialized and explicit communicative symbols that have encoded implicit institutional/organization semantics, which may lead to switching the context of the collaborative behaviour of the participating agents. BIC is a more intuitive way to achieve collaboration without explicit communication but rather by exploiting uncodified behaviours to be contextually used as massages.

Putting forward actions for communication also meets the essence of workflow managements systems [24]. These are recognised among other collaboration-based technologies such as Computer Supported Cooperative Work precisely for the way they bring tasks allocation to a first class entity for collaboration rather than another communication protocol. Well-defined workflows pertained with clearly defined initiation actions, termination actions and final goals are necessary to overcome the difficulties of extracting intentions and goals of the performing agent provided that these actions performed intentionally.

3 Social Roles and Laws

Organisational norms, unlike ordinary rules, allow the entities that they coordinate to deviate from expected behaviour. Norms can be violated by social entities with or without justification. Therefore, flexible systems should provide not only room for these violations to take place, but also a way to respond to violations in a way that comply with the existing context. By ”context” we mean some representation of the cognitive state of the individual (or group of individuals) and the state of the world at a certain time. Research in deontic logic has introduced constructs that include violations and sub-idealities. *Contrary-to-duty* — CTD — is a deontic logic based formalism that allows an obligation to be evoked when another obligation is violated [15,21]. Our social modelling primitives are inline with CTD. In our work we concentrate on handling functional violations that aim to realise system goals in terms of business workflows.

As already motivated, our approach to modelling socio-technical systems is based on a new kind of architectural connector that can capture the organisational norms that apply to social components. Just like coordination laws rely on coordination interfaces to identify the capabilities required of the components to be able to be coordinated according to the rules of the law, social laws rely on social roles to capture the capabilities and normative aspects of social components.

3.1 Social Roles

Roles are abstract constructs that specify the behaviour expected of social components by means of operations and ascribed normative aspects that refer to certain organisational positions e.g. Head of Surgeons or a consistent set of assumed capabilities which is expected from a qualified doctor. More concretely, we distinguish between having the ability to perform an operation and having the qualification or authorisation to do so: a social component may have the ability to perform an operation and still trigger a role violation if it is not an instance of a role that has the right qualification. Here we use the word qualification to mean, for instance, that the organisation has empowered the social component to perform given operations. Delegation of services or operation is vital in any dynamic organisational setting, however, we are not investigating delegation is out of the scope of this research.

As discussed below, the execution of operations by a component when playing a role without the required qualification is governed by a social law. A social law specifies (social) rules that either impose sanctions or provide a configuration in which the operation can be safely executed, depending on the context in which the violation takes place. However, we need to stress that the execution of operations, even if by qualified components, can be governed by coordination laws and, as such, be refused in certain circumstances for operational reasons, not for deontic ones. We denote operations for which the role is qualified with [+]. We can also define a subsumption relation between operation: by declaring *op1* ⊃ *op2* we mean that *op1* can only be executed as part of *op2*, in which case a component qualified to do *op2* is also qualified to do *op1*.

The general structure of a social role is as follows:

```
social role rolename {specializes rolename}
types {{par}+:datatype*
operations{
            {'[+]'} opname {⊃ opname}
            }*
```

For instance, GPs are qualified to perform routine tasks of seeing patients and registering for shifts in wards. A GP can also perform minor operations but will trigger a role violation unless he/she is an instance of a role that is qualified to do so.

```
social role GP specializes Person
types p:patient,w:ward,op:operation
operations
           [+]seePatient(p)
           collectData(p) ⊃ seePatient(p)
           checkBloodPressure(p) ⊃ seePatient(p)
           [+]registerShift(w)
           minorOp(op,p)
```

Another example concerns gastroenterologists, which specialise registrar internals.

```
social role gastro specializes registrar_internal
types p:patient, op:operation
operations
           [+]gastroProc(p)
           setProgram(p) ⊃ gastroProc(p)
           takeBiopsy(p) ⊃ gastroProc(p)
           minorOp(op,p)
```

Instances of this role perform routine gastro-procedures but minor operations are still restricted to instances of specialisations that have the qualification to do so. Registrar surgeons are an example of such a specialisation:

```
social role registrar_surgeon specializes GP
types p:patient,op:operation
operations
           [+]minorOp(op,p)
           setupMonitor(op,p) ⊃ minorOp(op,p)
           majorOp(op,p)
```

Roles are organised in hierarchies; qualification to perform an operation is inherited. The picture below provides an example.

The overall importance of distinguishing between ability and qualification to perform an operation is that it reduces what are normally called normative positions [15] to deontically-governed role transitions. This is because, in the context of an

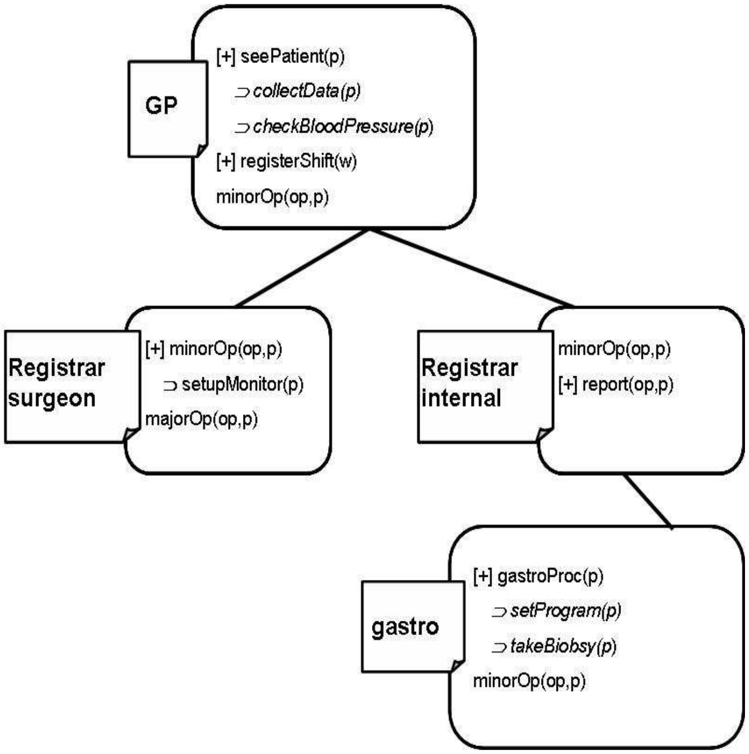

Fig. 1. A Role Hierarchy

architectural approach to system development, it is easy to model role transition in terms of dynamic reconfiguration. In the absence of such an explicit hierarchy, sub-ideal situations would have to be resolved just by means of sanctions. Instead, we take a more positive and active approach by enabling a reconfiguration if the current context allows such a deviation of the norm to be tolerated. This is precisely the goal of social laws as discussed below.

3.2 Social laws

Social laws are the primitives that we propose for complementing coordination laws when modelling social interactions. Social laws capture normative aspects of collaboration using deontic concepts such as obligations and interdictions applied to actions as performed by social components when acting according to a given social role. If we consider the inclusion of the organisational dimension in one hand, and the biddability of social components of the system in the other, we find a gap that needs to be filled if we want to be able to reason about the behaviour that emerges from the collaboration in place within a system. The notion of collaboration that we adopt is in line with [4]: "*The coordination in organization and societies cannot be accounted for without considering social laws of the organisation and the way they constrain the behaviour of individual agents*". It requires highly integrated and flexible laws between people, processes and technological components that can govern the interactions that emerge according to the policies of the organisation. In order to make such concepts applicable, social laws need to incorporate three main components: roles, norms and sanctions.

Our approach also builds on normative positions [15,21], which is an extension of CTD that introduces control to model norms in an effective way. A normative position represents all logically possible (norms, control and influence) relations between a social role and other partners in a certain configuration. We argue that human biddability can be modelled as a set of normative positions that apply to the set of roles involved.

A social law defines what actions should be taken when an operation is initiated by a social component acting according to a given social role — the anchor role of the social law — which is not qualified to do so. That is to say, social laws provide a context for the system to react and adapt to a sub-ideal situation generated by a violation that is committed by a social entity. The reaction, which is normally taken by an adaptation or a reconfiguration manager, can consist of either the imposition of sanctions or a reconfiguration of the system. The latter can be performed so as to put in place a context in which the social component can proceed with the operation in spite of the fact that it is not qualified, for instance a doctor having to perform a minor operation in a life-critical situation. For this purpose, new equipment and/or social components may need to be added to the system configuration to assist the doctor, and the software components that control the system may need to be reconfigured, say to enable the doctor to perform operations that, in normative states, should not be enabled. This sort of reaction captures what is sometimes called the role-binding anomaly as described in [17].

Another situation in which a social law detects a violation is when an operation is initiated in a context in which it is not permitted according to some organisational norm. For instance, although a surgeon is qualified to perform a minor operation, the rules of the hospital are such that the consent of the patient is needed before initiating any operation. However, in a life-critical situation, it may be impossible to obtain consent and, in spite of this, the surgeon should be allowed to proceed. In this case, a reconfiguration should again be triggered, implying a change in the structure of the system in terms of adding/replacing components and/or coordination contracts. Social laws have the following general structure:

```
 social law name
 anchor role social role
 partners
          {social role, coordination interface}*
 types {{par}+:datatype*
 {violation rule
                 when trigger
                 if  condition
                 reconfiguration task
                 sanction {operation}*
}*
```

Besides the anchor role, a social law identifies other partners through either social roles or coordination interfaces. The former are useful for reconfiguration operations and the latter for both detecting triggers and reconfigurations as explained below. There are three kinds of triggers for violation rules: (1) operations of the anchor role that are executed by social components that have no qualification; (2) operations for which the anchor role is qualified but they are initiated in a context in which they are not permitted; (3) operations of the anchor role that are not executed in contexts in which they are required.

The first takes the form:

```
unqualified operation
```

The second takes the form:

```
operation and not enabling state
```

The third are of the form:

```
active state and not operation
```

Notice that, in order to detect the violation of the enabling state (permission), we need a coordination interface that provides an operation that returns a Boolean value and, in order to detect the violation of the obligation, we need a coordination interface that provides an event. The "negated operation" holds in the states in which the operation has not been scheduled for execution by the component that instantiates the anchor role. Definitions of permission and obligations have been adapted from [6]. Sanctions are used to impose organisationally agreed procedures which react to the envisaged violation, that is to be absorbed and then halted cannot be handled through a reconfiguration requires instead that punitive actions be taken, possibly with the assistance of system stakeholders identified as partners.

As an example, consider the social law that applies to minor operations. Such procedures involve a social role, a GP, who is the anchor role in the sense that the social laws will apply to the actions performed by instances of this role. In addition, three coordination interfaces are required to ensure that the GP interacts with the right components: the device that is monitoring the procedure - *monitor-procedure*, and the software component that provides access to administrative data - *administrator*. In the configuration of the system, there will be coordination laws modelling the way these three components interact. Because of lack of space, we are not able to provide the definition of the relevant coordination interfaces and laws.

```
 social law minor-operation
 anchor role d:GP
 partners
anchor role d:GP
 type p:patient, op:operation
 partners
         a:administrator
         m:monitor-procedure
 when d.minorOp(op,p) and not a.ensureConsent(op,d,p)
         if m.alarm(p)
         reconfiguration reconfMinor(d,op)
         sanction a.record(d,op,"no_consent")
 when unqualified d.minorOp(op,p)
         if m.alarm(p)
         reconfiguration reconfUnqual(d,op,p)
         sanction a.record(d,op,"unqualified")
```

The social law has two rules triggered by the same event: the moment in which the doctor initiates the operation on the patient. The first rule handles the situation in which there is no record of consent having been given by the patient for the doctor to perform that operation. If the monitor detects that there is an emergency situation, then a reconfiguration of the context is performed to put in place the components and coordination contracts that are required for the operation to proceed. This may involve, for instance, providing access to further information registered on the patient's file, say on allergies. However, if the monitor does not detect an emergency, sanctions apply by recording the violation in the doctor's file.

The second rule is activated if the actual doctor is not qualified to perform minor operations, which is possible because the doctor's role matches one of the roles in the non-surgical branch of the doctors role hierarchy: GP, registrar internal or gastro. In this case, we have to distinguish again if there is an emergency. For simplicity, we used the same alarm condition provided by the monitor. If an emergency is indeed detected, a reconfiguration of the context is performed to allow the doctor to proceed, for instance unblocking actions that, in normative states, should be forbidden to the doctor. Otherwise, sanctions apply. Notice that the reconfiguration operation takes the doctor as a parameter: the hospital may have different rules about the context that should be present during an operation depending on the type of doctor.

Notice that both rules can apply:the doctor may not be qualified and the patient may not have given consent. In the case of an emergency, both reconfigurations apply; otherwise, both sanctions are implemented.

Due to lack of space, we are not able to discuss the reconfiguration language in which the tasks *reconfMinor(d,op)* and *reconfUnqual(d,op,p)* are defined. See instead [2] for the reconfiguration language used in CCC and [25] for a semantics of reconfiguration based on graph-transformations.

4 Concluding Remarks

Our goal in this paper was to put forward a set of modelling primitives that can support the specification of socio-technical systems that are flexible enough to respond to evolving social and organisational contexts. This includes mechanisms through which systems can react to situations in which the people who play a role as components of the system can deviate from organisationally prescribed norms. Such reactions should lead to reconfigurations performed in a way that is flexible and predictable, and conforms to the contexts in which violations occur.

A further concern was to separate the modelling of such social aspects from the computational and coordination aspects that capture the way the system fulfills given business goals so that they can all evolve independently of each other. That is, the organisation should be able to change its business goals without having to change its norms and vice-versa. This is why social laws are defined independently of the coordination laws that regulate interactions. However, once instantiated in a given configuration of a system, both social and coordination laws come together to model the interconnections established at run-time between software and human entities, reflecting the actual capabilities of the entities that play the designated social roles. That is to say, in order to understand how a system behaves in a given state, we need to now which coordination and which social laws apply to the components present it that state and which roles these components play in the system.

Social laws react to deviations from normative behaviour in two possible ways: either through sanctions or "facilitations" - reconfiguration operations that may add or remove coordination contracts and components to put in place a new configuration that reflects a new context in which the system can recover from a sub-ideal situation.

We admit that our examples are illustrative in the sense that they are not based on real-life socio-technical settings, however, they were inspired by lengthy discussions with the medical staff at a busy hospital. These discussions have shaped a candidate case study that we intend to investigate in the future. We envision that our case study will exploit reconfiguration rules to make systems self-adaptable to roles in which humans use them and to the evoloving organisational policies that monitor and react to such usage. A monitoring system in a theatre room which facilitates access to medical records and reports to the hospital administration would be an ideal example.

We are currently enriching the way reconfigurations rules are modelled in the CCC-approach [2]; we have in mind to explore the way socio-technical systems are addressed in Jackson's Problem Frames approach [11], capitalising on preliminary work that related Problem Frames and the CCC approach [5].

Acknowledgements

We would like to thank Luís Andrade and Georgios Koutsoukos from ATX Software SA, Tom Maibaum from McMaster University, Marek Sergot from Imperial College, Reiko Heckel from University of Leicester, John Hall and Michel Wermelinger from the Open University, for many discussions and comments that have helped us to shape our ideas. We would also like to thank the referees for many helpful and encouraging remarks.

References

[1] Allen, R. and G. Garlan, "A Formal Basis for Architectural Connectors", *ACM TOESM*, vol. 6, pp. 213-249, 1997.

[2] Andrade, L. F.,J., Gouveia, J. L. Fiadeiro and G. Koutsoukos, "Separating Computation, Coordination and Configuration", *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 14, pp. 533-370, 2002.

[3] Andrade, L. F.,J. L. Fiadeiro, "Architecture Based Evolution of Software Systems", in M. Bernardo and P. Inverardi, (eds.), *Formal Methods for Software Architectures*, LNCS 2804, Springer-Verlag, pp. 148-181, 2003.

[4] Barbuceanu, M.,T. Gray and S. Mankovski, "Role of Obligation in Multi-Agent Coordination", *Applied Artificial Intelligence*, vol. 13, pp. 11-38, 1999.

[5] Barroca, L.,J. L. Fiadeiro, M. Jackson, R. Laney and B. Nuseibeh, "Problem Frames: A Case for Coordination", in *Proc. 6th International Conference on Coordination Models and Languages (Coordination'04)*, LNCS, Springer-Verlag, p.p. 5-19, 2004.

[6] Boella, G., and L. der Torre, "Permissions and Obligations in Hierarchical Normative Systems", in *Proc. ICAIL*, Edinburgh, Scotland, 2003.

[7] Bass, L., P. Clements and R. Kazman, *Software Architecture in Practice*, Addison Wesley, 1998.

[8] Brier, B., L. Rapanotti and J. Hall, "Problem Frames for Socio-technical Systems: predictability and change", in *ICSE, 1st. International Workshop on Advances and Applications of Problem Frames (WAAPF 2004)*, Edinburgh, Scotland, 2004.

[9] Castelfranchi, C., and F. Giardini, "Silent Agents. Behavioural Implicit Communication for M-A Coordination and HMI", in *2nd Annual Symposium on Autonomous Intelligent Networks and Systems*, Menlo Park, CA, 2003

[10] Dustdar, S., "Caramba-A Process-Aware Collaboration System Supporting Ad hoc and Collaborative Processes in Virtual Teams", *Distributed and Parallel Databases*, vol. 15, pp. 45-66, 2004.

[11] Hall, J., M. Jackson, R. Laney, B. Nuseibeh and L. Rapanotti, "Relating Software Requirements and Architectures Using Problem Frames", in *Proc. Requirements Engineering (RE'02)*, Essen, Germany, IEEE Computer Society Press, pp. 137-144, 2002.

[12] Hulstijn, J., "Roles in Dialogue", in Kruijff-Korbayova (ed.), *Proc. 7th Workshop on the Semantics and Pragmatics of Dialogue*, pp. 43-50, 2003.

[13] Jackson, M., *Problem Frames: Analyzing and structuring software development problems*, Addison Wesley, 2001.

[14] Jackson, M., *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices*, Addison Wesley, 1995.

[15] Jones, A. and M. Sergot, "A Formal Characterisation of Institutionalised Power", *Journal IGPL*, vol. 4, pp. 427-444, 1996

[16] Kristensen, B., "Object-Oriented Modelling with Roles", in *2nd International Conference on Object-Oriented information Systems*, Dublin, Ireland, 1996.

[17] Lee, J.-S. and C.-H. Bae, "An Enhanced Role Model for Alleviating the Role-Binding Anomaly", *Software: Practice & Experience*, vol. 32, pp. 1317-1344, 2002.

[18] Lupu, L. and M. Sloman, "A Policy Based Role Object Model", in *Proc. 1st international Enterprise Distributed Object Computing Workshop(EDOC'97)*, IEEE Computer Society, pp. 36-47, 1997.

[19] Sandhu, R. S., E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-Based Access Control Models", *IEEE Computer*, vol. 29, pp. 38-48, 1996.

[20] Searle, J. R., "Speech Acts, Mind, and Social Reality", in G. Grewendorf and G. Meggle (eds.), *Studies in Linguistics and Philosophy*, vol. 79, pp. 3-16, 2002.

[21] Sergot, M., "Normative Positions", in H. Prakken and P. McNamara (eds.), *Norms, Logics and Information systems: new studies in Deontic Logic*, IOS Press, pp. 289-310, 1998.

[22] Shaw, M. and D. Garlan, *Software Architecture*, Prentice Hall, 1996.

[23] Sloman, M. and E. Lupu, "Conflicts in Policy-based Distributed Systems Management", *IEEE Trans. on Software Eng.*, vol. 25(6), pp.852-869, 1999.

[24] van der Aalst, W.M.P, and K. M. van Hee, "'Workflow Management: Models, Methods, and Systems", MIT Press, Cambridge, MA, 2002.

[25] Wermelinger, M., A. Lopes and J. L. Fiadeiro, "A Graph Based Architectural (Re)configuration Language", in *Proc. of ESEC/FSE'01*, ACM Press, 21-32, 2001.