Social Requirements Models for Services*

John Mylopoulos[1], Daniel Amyot[1], Luigi Logrippo[1,2], Alireza Parvizimosaed[1], and Sepehr Sharifi[1]

[1] School of EECS, University of Ottawa, Ottawa, Canada
{jmylopou, damyot, logrippo, aparv007, sshar190}@uottawa.ca
[2] Université du Québec en Outaouais, Gatineau, Canada

Abstract. Social dependance relationships were used in the *i** requirements modelling language to represent dependencies among social actors. We study the evolution of the notion of social dependency into that of commitment in the Azzurra specification language for business processes, and then into the notions of obligation and power in the Symboleo specification language for legal contracts. Our account focuses on the difference in the semantics of these relationships, the language used to talk about them, and how appropriate they are for capturing requirements for services.

Keywords: Requirements Model, Social Dependency, Service, Business Process, Legal Contract

1 Introduction

Services are social activities involving a *server* and a *client*, where the client *depends* on the server to deliver the service, be it the sale of an item, the delivery of food, or transportation from home to work. As such, services can be modeled as *social dependencies* in *i** [5], a requirements modeling language founded on the notions of *actor* (agent/role) and *social dependency*.

It turns out that social dependency is a very powerful concept that constitutes the foundation for social modelling and has spawned interesting offspring dependencies that serve as primitives for business process and legal contract modelling. The purpose of this chapter is to discuss the ontological nature and contrast three types of social dependence relationships: social dependencies (*i**), commitments (Azzurra) [4], as well as obligations and powers (Symboleo) [8,10]. These relationships have been studied for over three decades and have involved many collaborators beyond the authors.

2 Social Dependencies (*i**)

As shown in the top model of Fig. 1, in a social dependency the client wants something and the server is able and willing to deliver. However, the force of the

* This chapter shares much of the narrative with [7], but focuses on modeling requirements for *services* and has been written for a very different audience.

dependence can vary from weak to strong on either side of the service. Consider a car owner's dependence on a body shop to repair her car: it is usually weak on the car owner's side because there are other body shops that can do the job, and strong on the body shop's side because that's the mission of body shops.

Contrast this with one's dependence on a renowned surgeon for a rare medical operation. This one is strong on the client's side, because there are no substitute servers. The force of the dependence on the server's side is more important, as she is responsible for delivering the service. That force is defined along two dimensions: ability to deliver the service, and degree of commitment to deliver. The dependence on that surgeon is strong on ability and medium on commitment, as surgeons will postpone scheduled operations in case of emergencies.

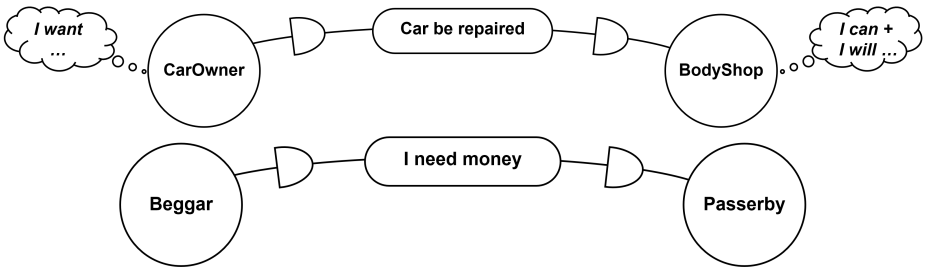

Fig. 1: Examples of social dependence relationships in i^* (adapted from Eric Yu's lectures on i^*)

Now, consider a beggar who depends on passersby to get some some money (the service) as Fig. 1 (bottom) shows. This is a social dependency too, and it is weak on both sides: The beggar can switch to another kind of dependency to get money, e.g, work, while the passersby have not even agreed explicitly with the beggar to deliver the service, they are just willing to do it, occasionally. Here, the dependency is established statistically: some passersby give money, and sooner or later this establishes a dependency for the beggar. Note that this dependence does not qualify as a service in the sense that there is no commitment on the part of the server to deliver the service.

i^* does recognize the importance of the *force* of a dependence by allowing three possible levels of force: *critical*, *committed* and *open* [1]. But, as discussed in the sequel, it turns out that in several areas of research, people have opted for defining specializations of social dependence relationships where the strength of dependence is built into their semantics. Commitments, obligations and powers are three such relationships.

The use of i^* to build requirement models for services goes back to Diana Lau's Masters thesis at the University of Toronto, presented in [6], but also [1] and [9]. All three proposals use the Tropos methodology [3] for deriving agent-oriented implementations from stakeholder requirements.

Figure 2 shows a social dependency model for an online retail service. The service involves several actors (circles), including the retailer, the customer, and the retailer's bank. They each have goals, represented as rounded-corner rectangles, and softgoals (fuzzy goals), represented as clouds: the retailer wants to maximize profits, the customer wants to own products and the bank wants to deliver secure transaction services, among others. There are also dependencies: the retailer depends on a direct supply vendor to ship products to retailer's customers, on the bank to deliver deposit/withdrawal/transfer services, etc.

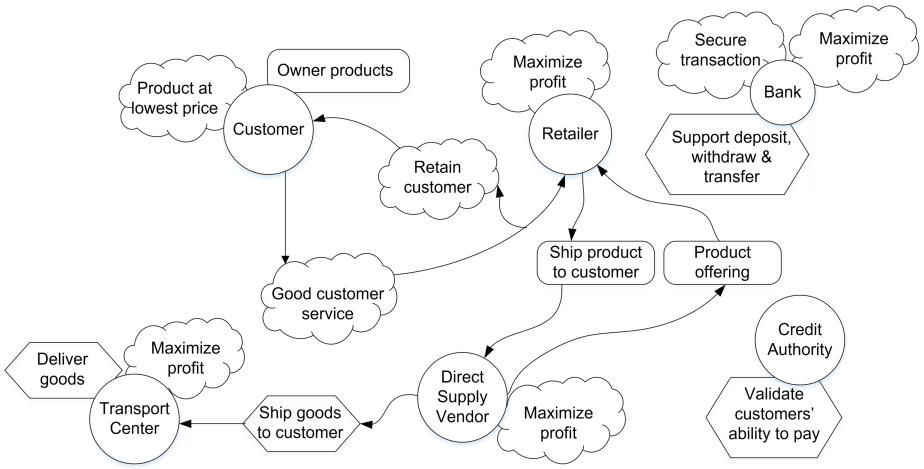

Fig. 2: Social dependency model for an online retail service (adapted from [6])

*i** has had many offshoots, including Tropos [3], a methodology for designing agent-oriented software systems, as well as the Goal-oriented Requirement Language (GRL), part of the User Requirements Notation (URN) standard [2].

3 Commitments (Azzurra)

Originating in the area of Multi-Agent Systems [11], *commitments* capture a social dependence where there is an explicit speech act executed "I want X – I commit to fulfill X". This kind of social dependence on a service has substantially more force than the beggar's dependence on passersby. It means that the server intends to deliver, provided that some conditions hold. So, commitments are social dependencies that always arise from intentions, rather than mere practice, and are established through speech acts. More formally, commitments are 4-tuples `C(debtor, creditor, antecedent, consequent)`, where creditor is the client, i.e., the beneficiary of the service, while debtor is the server. Moreover, the commitment is fulfilled when the consequent becomes true, provided that the antecedent is true. Moreover, commitments go through states, such as *created*,

active, *suspended*, *success* and *failure*. Allowable state transitions can be defined by state diagrams, as shown in Fig. 3. Note that commitments constitute a specialization of social dependencies, with better fleshed out semantics, proposed for use in multi-agent systems. They also come with a precise level of force for the debtors/servers who *intend* to fulfill what they are committed to, while the creditors/clients have the *right* to expect that the commitments will be fulfilled. The passersby mentioned earlier have no commitment towards the beggars and, in turn, they have no right to expect anything from them.

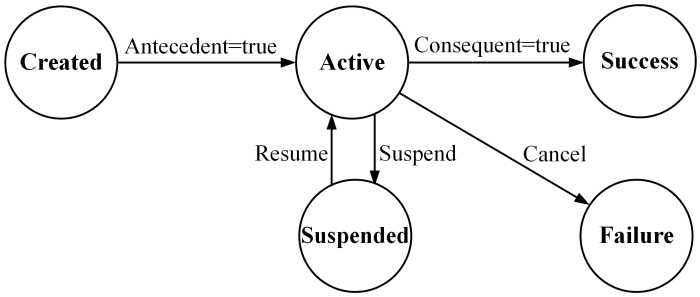

Fig. 3: The lifecycle of a commitment[3]

Azzurra is a conceptual modelling language for business processes. Its main thesis is that business processes being social artifacts, need to be defined in social terms, rather than system-oriented ones (e.g., Petri nets, BPMN and the like). Accordingly, business processes (aka *protocols*) are defined in terms of roles and commitments, with constraints attached. Azzurra models can be seen as requirements specifications for services, defined in terms of the business processes through which they will be delivered. Moreover, as specifications they describe what a business process is supposed to achieve without getting into the details of how to achieve it.

Table 1 presents an Azzurra protocol for fracture treatment (adopted from [4]). The protocol includes as parameters a hospital number that serves as key for treatment instances, a patient, a specialist; it also includes role parameters, such as a radiologist and a surgeon, as shown.

There are nine commitments for this protocol, each using a <*trigger*> $\twoheadrightarrow$ <*commitment*> format. The first, C_1, is triggered when the protocol is instantiated, has as roles the specialist (server) and patient (client), it is unconditional (antecedent= true) and is fulfilled when the patient is examined, then diagnosed and then de-hospitalized. The second commitment, C_2, is triggered if there is no need for X-rays and fulfilled when a sling is made for the patient. Protocol refinements constrain the agents that participate in a protocol instance. For example, agents may be constrained on how many concurrent commitments they

[3] Figure 3 is actually a simplification of a commitment's lifecycle in Azzurra.

Table 1: An Azzurra protocol for fracture treatment

protocol Treatment(**key** hospnr, pt : Patient, sp : Specialist){
ag-variables: rc : RehabCenter, ra : Radiologist, or : Orthopedist, su : Surgeon, nu : Nurse;
commitments:
init $\twoheadrightarrow$ C_1 : C(sp, pt, $\top$, Examined . Diagnosed . Dehospd) final
NoXRayNeeded $\twoheadrightarrow$ C_2 : C(or, sp, $\top$, SlingMade)
XRayRequested $\twoheadrightarrow$ C_3 : C(ra, sp, $\top$, XRayPerformed)
XRayRequested $\twoheadrightarrow$ C_4 : C*(sp, ra, XRayPerformed, FractAssessed)
FractAssessed $\twoheadrightarrow$ C_5 : C(or, sp, $\top$, ((Fixated$\oplus$Plastered) $\vee$ fulfill(C_6) $\vee$ SlingMade))
FractAssessed $\twoheadrightarrow_{\leq 2h}$ C_6 : C*(su, or, SurgeryRequested, Operated)
Operated[$\neg$fused] $\twoheadrightarrow$ C_7 : C(nu, pt, $\top$, RcChosen(rc))
RcChosen(rc) $\twoheadrightarrow$ C_8 : C(rc, pt, $\top$, fulfil-p(RehabGiven, key=hospnr, pat-id=pt, ref-sp=sp))
MedPrescribed(m) $\twoheadrightarrow$ C_9 : C(nu, sp, $\top$, MedApplied(m))
can-deleg-no-resp(C_3)
deadline(C_2, 2h)
protocol refinements:
role-confl(Radiologist, Orthopedist)
kb:
implies(XRayRequested, Diagnosed)
implies(NoXRayNeeded, Diagnosed)
implies(MedPrescribed(m), Diagnosed)
mutExcl(XRayRequested, NoXRayNeeded) }

have for a given role, such as a surgeon for treatment protocol instances (max-per-role). Finally, the knowledge base (KB) defines some domain axioms, which can be used to reason about propositions serving as triggers, antecedents or consequents of commitments.

Azzurra supports two types of reasoning for protocols. `ENACTPROTOCOL` determines how an event updates the state of a protocol instance and of the commitment instances therein. `CHECKCOMPLIANCE` checks whether an occurred event violates the specification of a protocol instance. This corresponds to identifying commitments that are not created/fulfilled, unexpected commitment operations and protocol constraint violations.

In summary, commitments specialize and improve the formalization of social dependence relationships. They also come with a language richer than *i** for modeling business processes in an outcome-oriented approach. Finally, and most importantly, Azzurra is a more appropriate language than *i** for describing requirements for services, as it commits the server to deliver, and gives the client the right to expect the service.

4 Obligations and Powers (Symboleo)

Obligations are commitments with legal force. The legal force is defined through *powers* that a creditor has towards the debtor of an obligation to cancel or suspend an obligation or another power, or initiate new obligations or powers. The concept of obligation is a specialization of the concept of commitment in that obligations can be created, cancelled, etc. by someone who has the power to do so. In turn, powers constitute a specialization of obligations in that they can include in their antecedent the creation, cancellation, etc. of other powers or obligations.

Obligations and powers constitute the basic elements of *legal contracts*. Legal contracts form the foundation of all commerce world-wide and have been used since time unmemorable. Legal contracts can be thought as process specifications that describe the space of allowable executions that comply with legal terms and conditions. The presence of powers in legal contracts makes them a much more malleable concept than that of business processes in that they can be reshaped with the introduction/cancellation of obligations while contracts are being executed ("performed" in Law).

Symboleo is a formal specification language for legal contracts, intended to serve in formalizing requirements for smart contracts. The latter are software systems, possibly running on blockchain platforms, that partially automate, monitor and control the execution of legal contracts. Symboleo is founded on an ontology that is centered around the notions of *obligation* and *power*, and includes *role* and *party* (the actors playing roles in a contract), *asset*, *situation* and *event*. Situations occur over time, e.g., the situation of commuting to work. Events, on the other hand, happen instantaneously, e.g., *arrivedAtWork*.

Symboleo adopts many elements from Azzurra. Obligations and powers use the same format as commitments. Their antecedents, consequents, and triggers are expressed in terms of events happening in a certain order and satisfying constraints. For example, for a sale contract the consequent of a delivery obligation may be "Sale item delivered to delivery address by delivery date".

Table 2 presents a Symboleo specification for a contract (adapted from [10]). As shown, a Symboleo specification begins with the description of concepts in the domain. These are defined as classes that specialize concepts in the Symboleo ontology. For example, `Goods` specializes `Asset` and has an additional attribute `goodsID`. Instances of this class include sale items involved in sale transactions. The domain model for a contract is followed by declarations of variables that take as values instances of domain classes. Pre/post-conditions have the same semantics as in program specifications. The core of contract specifications consists of obligations and powers. In our example there are two obligations: the seller must deliver the sale item to the delivery address by the delivery date (O_1), while the buyer must pay on time the sale amount (O_2). The contract also includes one power: if the buyer violates the payment obligation, the seller has the power to terminate the contract (P_1). Note that the creditor of a power may choose to not exercise it.

Table 2: Abbreviated Symboleo specification for a goods sale contract

Domain salesD

Goods **isA** Asset **with** goodsID: Integer;
...
Delivered **isA** Event **with** delAddress: String, delDueDate: Date;

endDomain
Contract salesC(seller: Seller, buyer: Buyer, ID: Integer, amnt: Integer, curr: Currency, delAdd, delDd: String)

Declarations
/∗ *Values of parameters are passed on to the variables defined in the domain model.* ∗/
goods : Goods **with** goodsID := ID;
...
delivered : Delivered **with** delAddress := delAdd, delDueDate := delDd;
Preconditions
isOwner(seller, goods) AND NOT isOwner(buyer, goods);
Postconditions
isOwner(buyer, goods) AND NOT isOwner(seller, goods);
Obligations
O_1 : O(Seller, Buyer, true, **happensBefore**(delivered, delivered.delDueD));
O_2 : O(Buyer, Seller, true, **happensBefore**(paid, paid.payDueD));
Powers
P_1 : **violates**(O_2, _) → P(Seller, Buyer, true, **terminates**(salesC));
SurvivingObl
/∗ *Some obligations may remain active, e.g., confidentiality obligations.* ∗/
Constraints
not(isEqual(buyer, seller));

endContract

Legal contracts can be very complex constructs with many features that go well beyond those of business processes. For instance, some obligations may apply after the successful termination of a contract (and are accordingly called *surviving* obligations). A confidentiality obligation for a sale transaction for 6 months after a contract terminates is an example of such an obligation. A contract may spawn subcontracts that may be established while the contract is executing. For example, when a large project is undertaken in the construction industry, not all the subcontractors with their respective subcontracts might have been identified when the project starts. Symboleo specifications can be validated to ensure that they are consistent with the expectations of the contracting parties by a tool that enacts scenarios and determines the contract's final state. For example, for the scenario "Seller delivers on time, buyer does not pay on time, seller exercises power to terminate", the tool determines that the final state of the contract is 'cancelled'. The scenarios for validation are provided by the contracting parties,

along with their anticipated final state of the contract when each scenario is enacted.

All commercial services are defined in terms of legal contracts and they do include both obligations and powers for the contracting parties. But note that not all services need to have associated explicit legal contracts, see for example Government or volunteer services.

5 Conclusions

Social models consisting of actors and social dependencies are useful for capturing requirements for social systems that include software, business processes and services. We trace the evolution of the concept of social dependence to commitments and then to obligations and powers, focusing on changes in semantics, the languages where these relationships inhere, and the application domains, i.e., the types of artifacts we are defining requirements for.

Epilogue

This chapter has been written for the occasion of Mike Papazoglou's retirement celebration, to hopefully happen within 2021. I have known Mike for thirty years as a colleague working on topics of mutual interest, collaborator on international projects, co-author and friend.

Throughout, Mike has been an exemplary researcher not only for his research contributions and their impact, but also for the leadership role he undertook in conceptualizing, shaping and promoting research areas. During the 90s, his passion was with Cooperative Information Systems, an area of research where he founded a conference series and a journal that are still going strong. In the following decade, his allegiance shifted to Web Services and Service-Oriented Computing where he was a key player in framing the area and the research venues that define and serve it.

It is a pleasure to contribute, along with my University of Ottawa colleagues working on smart contracts, a chapter on Service Requirements that spans his interests as well as mine.

John Mylopoulos
Toronto, June 8, 2020

References

1. Aiello, M., Giorgini, P.: Applying the Tropos methodology for analysing web services requirements and reasoning about qualities of services. Tech. Rep. DIT-04-034, University of Trento (2004)

[3] Nothing is certain in the days of the pandemic.

2. Amyot, D., Mussbacher, G.: User requirements notation: the first ten years, the next ten years. JSW **6**(5), 747–768 (2011)
3. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems **8**(3), 203–236 (2004)
4. Dalpiaz, F., Cardoso, E., Canobbio, G., Giorgini, P., Mylopoulos, J.: Social specifications of business processes with Azzurra. In: 9th RCIS. pp. 7–18. IEEE (2015)
5. Eric, S., Giorgini, P., Maiden, N., Mylopoulos, J.: Social modeling for requirements engineering. MIT Press (2011)
6. Lau, D., Mylopoulos, J.: Designing web services with Tropos. In: Proceedings. IEEE International Conference on Web Services, 2004. pp. 306–313. IEEE CS (2004)
7. Mylopoulos, J., Amyot, D., Logrippo, L., Parvizimosaed, A., Sharifi, S.: Social dependence relationships in requirements engineering. In: 13th International iStar Workshop, CEUR-WS 2642. pp. 55–60 (2020)
8. Parvizimosaed, A., Sharifi, S., Amyot, D., Logrippo, L., Mylopoulos, J.: Subcontracting, assignment, and substitution for legal contracts in Symboleo. In: 39th International Conference on Conceptual Modeling (ER'20). Springer (2020)
9. Penserini, L., Perini, A., Susi, A., Mylopoulos, J.: From stakeholder needs to service requirements. In: 2006 Service-Oriented Computing: Consequences for Engineering Requirements (SOCCER'06-RE'06 Workshop). IEEE CS (2006)
10. Sharifi, S., Parvizimosaed, A., Amyot, D., Logrippo, L., Mylopoulos, J.: Symboleo: A specification language for smart contracts. In: 28th IEEE International Requirements Engineering Conference (RE'20). pp. 384–389. IEEE CS (2020)
11. Singh, M.P.: An ontology for commitments in multiagent systems. Artificial intelligence and law **7**(1), 97–113 (1999)