Automated synthesis of norms in social networks

by

Iosu Mendizabal Borda

Master in Artificial Intelligence
April 20, 2015

Supervisor:
Dr. Juan A. Rodríguez-Aguilar
Artificial Intelligence Research Institute (IIIA)
Spanish Research Council (CSIC).

Co-Supervisor:
Dr. Maite López-Sánchez
Volume Visualization and Artificial Intelligence Research Group
Departament de Matemàtica Aplicada i Anàlisi (MAiA)
Universitat de Barcelona (UB)

Facultad de
Matemàtiques

Escola Tècnica
Superior D'Enginyeria

Facultad D'Informàtica
de Barcelona (FIB)

Abstract

On-line communities are virtual environments where users exchange contents. Occasionally, users' interactions lead to frictions, jeopardising the proper functioning of the community. Trying to avoid frictions, on-line communities typically incorporate a regulation mechanism based on: (i) norms set by the owner of the community; and (ii) human moderators. In this thesis we introduce a legislation mechanism, capable of automatically synthesise norms from a multi-agent domain, to synthesise norms in a deliberative and participatory manner, namely DESMON. Moreover, we have created an agent-based on-line community simulator to model the interactions within their users. We then empirically evaluate our norm synthesis approach against two state-of-the-art norm synthesis mechanisms, to regulate simulated on-line communities. As a result, DESMON is capable of synthesising normative systems according to the consensus of the users, capturing the opinion of the majority, hence avoiding frictions between them.

Acknowledgements

First and foremost I want to thank my advisors, whose guidance and dedication has made this work possible. My most sincere gratitude to Juan A. Rodriguez-Aguilar, for your unbreakable spirit of help, for being always optimistic about everything, for your endless knowledge and for betting on me. To Maite Lopez-Sanchez, for your cheerfulness and kindness, your perpetual aim of perfectionism and your good taste on colours ;). I sincerely thank you all the support and want to express gratitude because you trusted on me. You are the inspiration that everybody is looking for.

To Javier Morales because you have been like an advisor without being it, for letting me be part of your legacy and for helping me in any question I had and will have!

To the Artificial Intelligence Research Institute (IIIA-CSIC) who has adopted me and relied on me. Thanks to all the colleagues that I have encountered there. To all the doctoral students that I have met over the time. Thanks for making me feel as pleased as at home.

To the MAI Master fellas, that have made these last two years fly by.

And last but not least, to my Father Pedro, and my Mother Maite, for giving me the education and the opportunity of being what I am.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since the diffusion of the Internet, two decades ago, on-line communities have become an indispensable and necessary tool for communication, information/content exchange and social activities. Occasionally, users' interactions within these communities lead to frictions (conflicts) and this may jeopardise the proper functioning of such communities. These communities typically incorporate a pre-designed legislation, but it is far from being the ideal in terms of expressing users' opinion. In this thesis, we aim at creating a mechanism to generate a legislation which captures users' opinions. Hence, in the following we first introduce in section 1.1 the reasons that lead to this work. Thereafter in section 1.2 the approach we proposed to the problem. Section 1.3 presents the contributions we made, followed by Section 1.4 that offers a list of our published publications throughout this work.

1.1 Motivation

On-line communities are virtual environments that work over the Internet allowing their members to share content and information between users (e.g., pictures, opinions, information, videos, etc.). An on-line community can be either public or private. Public communities are those that accept any new user and let visualize the community without an account (as it is the case in most blogs). On the other hand,there are private communities, which are the ones that need an invitation to be part of and users only share their content with their "community friends". Furthermore, communities are dynamic, because its members may change their preferences and behaviours along time.

Within on-line communities users eventually interact with other users. Some of these users may eventually exhibit totally different opinions. As a consequence, some interactions may lead to frictions between users, which can be regarded as conflicting situations. These situations complicate the information exchange, hence causing discomfort to users, who may decide to abandon the community.

Community managers' goal is to avoid conflicting situations by establishing some hard-wired terms and conditions (i.e., norms), which specify what is acceptable and what is not. Users are expected to fulfil such norm. As an example, norms can avoid frictions between users, prohibiting them to share inappropriate contents, which may lead to arguments between users.

Typically, a regulation mechanism is based on:

- Some **pre-designed** *terms* and *conditions* (**norms**) set by the owner of the community that describe, in general terms, users' *obligations, permissions* and *prohibitions*. They aim at achieving healthy on-line communities in the sense proposed in [Hinds and Lee, 2011].
- Human **moderators** who actively participate in the community, performing corrective actions in order to avoid and solve conflicts following some pre-designed terms and conditions.

This regulatory approach suffers from lack of *participation*, *transparency* and *adaptability* among others.

1. It lacks of **participation** because the terms and conditions of an on-line community are fixed by the owner/designer of the community, hence not allowing users to participate in the regulation process.
2. It lacks of **transparency** when the norm is applied to someone. Because different moderators, as human beings they are, could apply different punishments to similar actions.
3. It lacks of **adaptability** since communities are dynamic (e.g. users may change or their opinions may change along time), the regulations initially set by the community owner/designer may may not perform well as the community changes.

Moreover, the use of human moderators to control all the conflictive contents of an on-line community is very costly. At this point, we can conclude that since norms are pre-designed and imposed by a community designer, they may not be aligned with the users' preferences. And thus, frictions will arise depending on what users consider inappropriate. Against this background, the purpose of this thesis is the creation of a mechanism to regulate on-line communities that takes into account users' preferences as to what they consider as appropriate and inappropriate.

1.2 Approach

From the discussion above follows that if a community designer aims at achieving a healthy on-line community, its regulation mechanism must allow users to participate in the regulation process. For this purpose, our approach will consider that users can *report* (i.e., complain about inappropriate content) about

conflicting situations. Based on such feedback we will engineer a regulation mechanism that generates *norms* aimed at avoiding the conflicting situations identified by users. Moreover, norm generation will only occur whenever there is enough evidence (i.e., users' complaints) supporting norms' creation, considering it creation necessary for the users of the community. In other words, we propose to build a regulation mechanism that cumulates users' complaints and generates norms only whenever enough complaints are gathered, and hence enough consensus exist to support norm generation.

We argue that our approach is deliberative because it will only generate norms when the feedback collected from users' opinions indicate that there is consensus about the need for a norm. This deliberative manner is bound to lead do larger and healthier on-line communities, which is the main aim of any community. On the one hand, as norms are, indirectly, created by users' participation, they will be aligned with users' preferences and may effectively avoid frictions. This will potentially attract new users and hence promote community growth. On the other hand, according to Ostrom's principles [Ostrom, 1990], individuals that participate in modifying the rules of a community are more likely to comply with them. Therefore, a community of users aware of their participation in the regulation process is more likely to follow the norms they contributed to create. Hence, this will promote a healthier behaviour from its members.

As a paradigmatic example, within this thesis we have collaborated with a real on-line community and they shared with us their moderation problems. This community is named *Fanscup* [Lanas and Garzón, 2005] and they manage soccer communities. Fanscup includes the main football tournaments of ten countries, with a web page for each team and their respective fans. Each web page is structured as a virtual community with typical sections: forum, photos, videos, news, and so on. The managers of Fanscup realised that as their web sites went larger and larger they needed support for the moderation of their web site. So they tried to tackle this problem by assigning moderator rules to the most active users of their community (a solution that most communities adopt). However, they have observed over time that this solution leads to two main attitudes on the behaviour of moderators. Firstly, the inactivity or the disassociation of the user to the given role, which creates moderators that do not moderate (false moderators). Secondly, the dictatorship or tyrannical attitude which corresponds to moderators that apply their own rules without following the web site regulations and taking into account their own opinions. Furthermore, the regulation of Fanscup was also pre-designed and remaind unmodified since its creation. They have seen two problems in their approach to regulation. On the one hand, likewise most virtual communities, the regulation was full of pre-designed general terms which users may not understand and may not agree with. On the other hand, most users never read the regulation of the community, because it is too dense and not easy to find.

The work in [Morales et al., 2013] introduced a participatory mechanism to

regulate a Multi-Agent Systems (MAS) scenario employing an architecture and computational model named IRON. There, IRON adopts the role of the regulation mechanism that synthesises norms based on MAS agents conflicts. A second approach of this work appears in [Morales et al., 2014a], where they introduce another norm synthesis approach, the so-called SIMON, to overcome IRON's drawbacks in terms of compactness. Although SIMON manages to synthesise more compact normative systems that IRON, likewise IRON it cannot be employed to synthesise norms that regulate virtual communities. Briefly, neither IRON nor SIMON are appropriate for this approach, because they were designed for a specific domain (i.e., traffic scenario where collisions between cars are conflicts) where norm generation was required to be reactive (i.e., for each collision between cars a conflict arises and this generates a norm). They cannot capture the consensus of the community, because a conflict in a community occurs whenever enough evidence has been gathered about a certain content. Moreover, both IRON and SIMON are very conservative approaches and take a long time to discard norms. Thus, they synthesises norms that may not be aligned with the overall preferences of the users.

Against these problems, our approach in this thesis is to adapt and improve SIMON to produce a *deliberative and participatory* approach to regulate on-line communities capturing their consensus. Moreover, we adapt SIMON because it was able to synthesise compact normative systems, and this is very useful when for human beings. Furthermore, this compactness will also help making the cost of reasoning smaller for the agents of the simulation.

1.3 Contributions

In this thesis we contribute to advancing the state-of-the-art on regulation mechanisms for domains where social interaction occurs, namely virtual communities. We regulate on-line communities in a deliberative manner capturing users' opinion with a norm synthesis mechanism, the so-called DESMON. DESMON extends from SIMON, detailed in [Morales et al., 2014a], which cannot capture users' consensus. For this reason, we aim at adapting the norm synthesis mechanism of SIMON to capture users' consensus. And hence, DESMON will allow to achieve healthier and larger on-line communities. This main contribution is composed of three sub-contributions:

- Our main contribution is a novel norm synthesis machine for virtual communities, the so-called DESMON. DESMON has been conceived to synthesise norms when there is enough consensus between users and to discard norms when they are no longer needed. Unlike SIMON, DESMON employs a more deliberative approach to generate norms. Furthermore, DESMON benefits from an alternative norm evaluation mechanism to detect when norms are no longer necessary, hence being less reluctant than SIMON to disregard norms that are not useful for regulatory purposes. Finally, DESMON implements a novel norm synthesis strategy that exploits a norm management

life-cycle richer than SIMON's. Nonetheless, DESMON borrows SIMON's powerful generalisation mechanism to guarantee the compactness of the normative systems that it synthesises. To the best of our knowledge, DESMON is the first deliberative norm synthesis machine.

- Second, we have created a novel multi-agent on-line community simulator, where users' behaviour is modelled by agents. To the best of our knowledge, this is the first multi-agent simulator for virtual communities. It is also part of the *NormLab framework*, which is a novel framework to support norm synthesis research, detailed in [Morales et al., 2014b] and [Morales et al., 2015a]. Furthermore, we have configured our simulator to allow IRON, SIMON and DESMON to operate on it, and hence synthesise norms.

- Third, we have empirically evaluated the norm synthesis approaches together with our virtual community simulator, namely we have compared IRON and SIMON with DESMON. Moreover, in each experiment we use three different populations, composed by: moderate (users that complain about inappropriate content) and spammer (users that upload inappropriate content) agents: (1) *majority of moderates*, (2) *balanced population*, and (3) *minority of moderates*. In our experiments we will like to observe: (i) with the first population (1), IRON, SIMON and DESMON converges to their compactest regulations, legislating the inappropriate contents. (ii) with the second population (2), as users are in equilibrium the NSM's will not converge to a normative system, because will be continuously activating and discarding norms. (iii) with the third population (3), as the majority is spammer we will like to observe a normative systems without regulations. On the one hand, as IRON and SIMON are reactive approaches they will hasten to the norm generation, continuously activating and discarding norms. On the other hand, DESMON will wait gathering evidences for norm generation to capture the majorities opinion. As the majority is spammer, and hence do not want to have norms, DESMON will be able to capture the opinion of the users and synthesise a normative system without regulation (without norms). So we will empirically demonstrate that DESMON is a deliberative approach, which is able to synthesise normative systems in consensus capturing the users' opinion.

1.4 Publications

In this section we will introduce the publications that we have published among the life period of this thesis.

- **Journal:**
  - Morales, J., Mendizabal, I., Sanchez-Pinsach, D., López-Sánchez, M., and Rodríguez-Aguilar, J. A. (2015b). Using iron to build frictionless on-line communities. *AI Communications*, 28:16

- **Demonstration:**
  - Morales, J., Mendizabal, I., Sánchez-Pinsach, D., Lopez-Sanchez, M., Wooldridge, M., and Vasconcelos, W. (2014b). Normlab: A framework to support research on norm synthesis (demonstration)
  - Morales, J., Mendizabal, I., Rodriguez-Aguilar, J. A., Sánchez-Pinsach, D., Lopez-Sanchez, M., Wooldridge, M., and Vasconcelos, W. (2015a). Extending normlab to spur research on norm synthesis (demonstration)

1.5 Organisation of the thesis

The remainder of this thesis is organised as follows.

Chapter 2. We first explain the necessary background for a better comprehension of this thesis. We introduce state-of-the-art moderation mechanisms used nowadays, and discuss different existing approaches for the moderation in virtual communities. We also introduce the Multi-Agent Systems and different coordination mechanisms for these systems. Thereafter, we explain the state-of-the-art norm synthesis mechanisms IRON, and afterwards its improved version SIMON.

Chapter 3. We introduce DESMON the improved norm synthesis approach, aim at synthesising norms in a deliberative and participatory manner capturing the consensus of the users' opinion.

Chapter 4. We present a novel multi-agent system simulator to simulate the behaviour of virtual communities.

Chapter 5. We explain the necessary adaptations made in our on-line communities simulator to allow all norm synthesis machines to operate on it.

Chapter 6. We empirically compare DESMON against IRON and SIMON when synthesising norms for a virtual community. In this chapter we study the deliberativeness of DESMON.

Chapter 7. We conclude this thesis by providing a summary of the presented contributions, drawing the most notable conclusions derived from this work, and outlining possible directions for future research.

Chapter 2

Background and related work

This chapter is focused on explaining the fundamental parts for a better comprehension of this thesis. We first introduce state-of-the-art moderation mechanisms in Section 2.1. Once we have described these mechanisms we study *how* is done the moderation and draw some conclusions of the state-of-the-art. As our purpose in this thesis is to create a moderation mechanism for social networks, we explain the Multi-Agent Systems, Section 2.2, that is the technique we use to simulate our virtual community. Afterwards, Section 2.3 presents some MAS coordination mechanisms, we focus on the coordination based on norms, a standard technique in MAS coordination, because is the one we use in order to legislate a virtual community. Moreover, now that we know *how* we will model our on-line community (MAS) and the mechanism to coordinate the users (norms) we present two state-of-the-art on-line norm synthesis mechanisms: IRON and SIMON. We are going to based on these mechanisms to generate our deliberative and participatory norm synthesis mechanism.

The next chapters will detail how we apply the explained technology: Chapter 3 describes the new deliberative norm synthesis mechanism, Chapter 4 details our Multi-Agent simulator of on-line communities. Chapter 5 explains the integration of the two previous chapters and finally, Chapter 6 empirically proves the different norm synthesis mechanisms with our novel social network domain.

2.1 State-of-the-art

Virtual communities and social networks are a very recent technology developed after the huge growth of the internet. Nowadays more and more people interact through these on-line communities and the moderation requirements of the contents of these sites has grown substantially. In the last decade some enterprises have started to focus on the problem of moderating them. In this section we

will show some of these enterprises that are the state-of-the-art in automatic moderation systems.

2.1.1 Keepcon

Keepcon [Guzmán, 2008] is an enterprise located in Buenos Aires (Argentina), founded in 2009. They started thinking about the automatic moderation of virtual communities and after some research on the area they realized that it was still a lot of work to do. So they created an automatic system to moderate virtual communities, which is called like the company, Keepcon.

The system is split in three main steps that are commented in the figure 2.1 and in the following paragraphs.

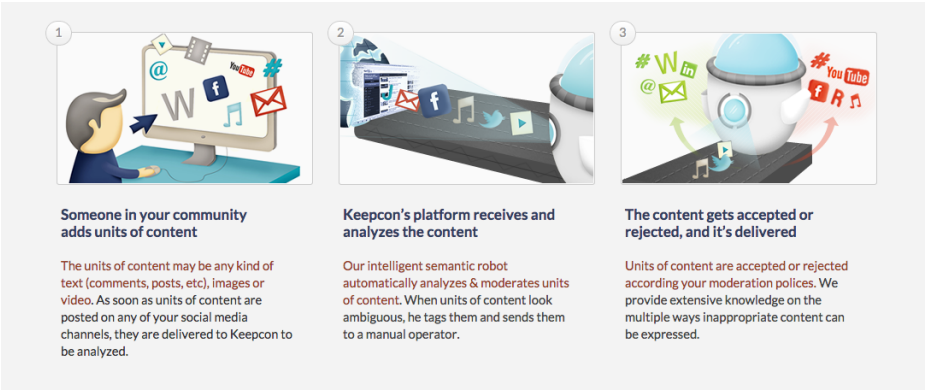

Figure 2.1: Main steps of the Keepcon.

1. **Content generation**: The first step is when the users generate and upload contents to the community. These contents go through Keepcon to be analysed and classified.

2. **Content analysis**: Once the content is in the Keepcon, it starts its analysis. This step is separated in two phases:

   2.1. *Comprehension phase*: The aim of this phase is to interpret and comprehend what is in the context. It is a tough task cause many of the contents are not written in the same way (for instance: to forgive someone = "2 4give some1"). Moreover, not only the way in which the text is written affect to the comprehension, there are ambiguous terms that can be taken as insult or offensive contents by users. For this task, Keepcon has different tools for: language detection, typos detection, url mining, image analysis, and so on.

2.2. *Classification phase*: This phase combines two techniques based on artificial intelligence: (i) *Symbolic technology* and (ii) *Learning machines*. The first one is based on rules, these rules are created by the clients and this way they avoid the undesirable contents for their community. The second one is based on machine learning techniques, they train some algorithms with a text corpus and then they test it with real cases to measure the precision and classify with it.

3. **Content verification**: Finally, after the classification stage the Keepcon machine sends the contents that it has not classified to the Manual Moderation Factory, where human moderators will manually classify the ambiguous contents that the machine can not classify.

2.1.2 Stilus Forum

Stilus Forum, [DAEDALUS, 2009], is a product of the enterprise *DAEDALUS - Data, decisions and language S.A.* The enterprise is specialised on searching technologies, natural language processing technologies and business intelligence.

This product is based on the automatic filtering of messages according to their content. Between the main characteristics of the product are worth noting the next ones:

- Content filtering: offensive, illegal, inappropriate, and so on.
- Levels of filtering: From less to more strict.
- Coverage of different types of Spanish: From the peninsular Spanish to the Latino american ones.
- Shortened text recognition, as SMS.
- Real time execution.

The enterprise does not give much more details about the process of moderation or text recognition.

2.1.3 Sourpanel

Sourpanel, [Gil Alonso and Roig, 2007], is a product of the Spanish enterprise *Sourtech*. They are developing their own moderation method since 2007 and are constantly improving it. Till 2012 they have moderated 60 millions of messages with this software. But the system is not automatized 100%, but is divided in an automatic moderation stage followed by a manual moderation, based on human moderators.

The system provides the following characteristics:

- **Automatic moderation**: Using natural language processing, regular expressions, spam filters, and so on.

- **Human moderators assistance**: Classifying the contents by subjects.
- **Alerts**: Predictions of inappropriate conversations, users analysis, and so on.
- **Reports**: Customized reports for the clients.

Moreover, there are two main actions which can be done by the Sourpanel to the contents. The first one is the automatic deletion of contents, based on a coefficient of confidence, and the second one is the assistance for the human moderators commented in its characteristics. This last aspect is very helpful for the human moderators, because the content is going to be classified by subjects by Sourpanel and then sent to the moderators for their review. The work-flow of the Sourpanel is described in figure 2.2.

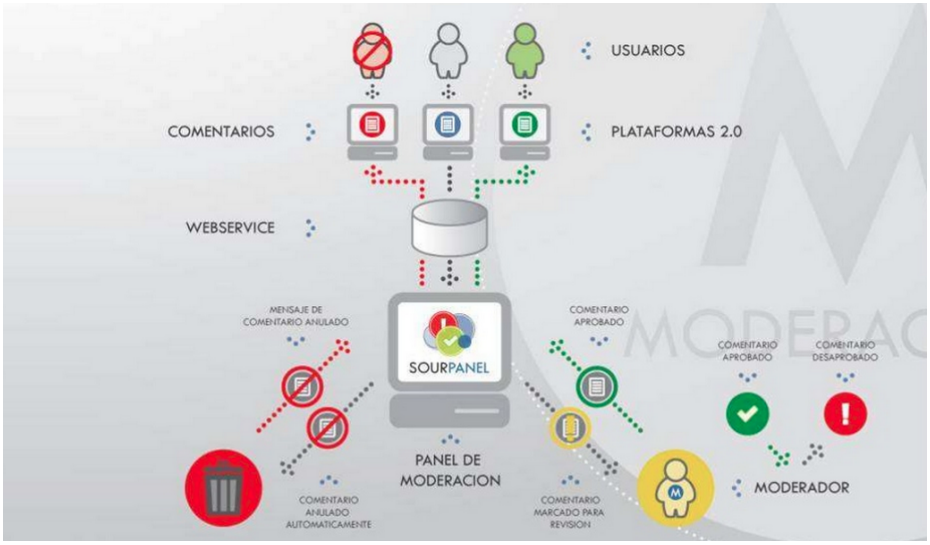

Figure 2.2: Work-flow of the Sourpanel software.

2.1.4 Definition of moderator and types of moderation

In virtual communities, where people interaction occurs, a moderator is a user or a employee that has been given a special authority to enforce the rules of the community with the aim of preserving the cordiality and good environment between the rest of the users.

Its main role is to detect or solve the conflicts that occur within the community applying the regulations that are ruling it. A moderator can edit, delete or move contents of a virtual community and also can punish the users that have uploaded inappropriate contents.

There are different types of moderation: *Pre-moderation* and *Post-moderation*,

these are generally divided by the moment in which the moderation has been done.

- **Pre-moderation**: Consists on moderating the content before it is published. Hence it is not visible by the other users of the virtual community. This dynamic is not very regular with human moderators, due to there will be too many contents to review before uploading them (impossible for big communities) and this will increase users impatience.
- **Post-moderation**: Consists on moderating the content after it is published. Once the user uploads a content it is visible for the rest of the community. So the moderation is done afterwards, when people has complaint about certain contents.

Finally, we can consider some advantages and disadvantages of these two types. First of all, time is a key issue that we have to take into account, post-moderation deals with it uploading all the contents without being reviewed, while pre-moderation waits to upload only the contents that have been reviewed. On the other hand, the user experience will be better in the pre-moderation because all the bad and inappropriate contents will be filtered out before other users can see them. Moreover, the post-moderation can create conflicts between users.

2.1.5 Conclusions of the state-of-the-art

After this research we can conclude that there are few automatic moderation mechanisms, and these are not autonomous at all. The combination of automatic moderation mechanisms and human moderators is the most common in the state-of-the-art mechanisms. Also, it is a very novel approach and the techniques of automatic moderation are improving with time. Nowadays the moderation has been automatised in the classification of the contents as an assistance for the human moderators. And only in the cases with high confidence of the automatic techniques is led to a content deletion without the supervision of a human.

2.2 Multi-Agent Systems

In this section we will explain the basic concepts of a multi-agent system extracted from Michael Wooldridge's book [Wooldridge, 2009].

2.2.1 Agents

We will start by defining what an agent is. An agent is a computational system that is located in an environment and is capable of making actions on an autonomous manner, with the aim of achieving some goals pre-designed on their creation.

Figure 2.3 provides an abstract vision of what an intelligent agent could look like. As depicted in the illustration, we can observe how the actions, realized by

the effectors or actuators, allow it to interact with the environment causing the changes on it. The agent also takes perceptions from the environment by means of sensors, and this is how it interacts with it.

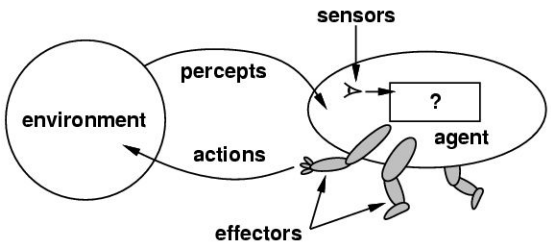

Figure 2.3: Abstract vision of an intelligent agent.

But, *When do we consider that an agent is intelligent?* One of the ways to answer this question will be listing the capabilities we think an intelligent agent must have. Therefore, for an agent to be intelligent we will mention the list of capabilities suggested in [Wooldridge and Jennings, 1995]:

- **Reactivity**: Intelligent agents are capable to perceive their environment and answer to any change that happened on it with the aim of satisfying the pre-designed goals.
- **Proactiveness**: Intelligent agents are able to exhibit goal-directed behaviour by taking the initiative in order to satisfy their design objectives
- **Social Ability**:Intelligent agents are capable of interacting with other agents (and possibly humans) in order to satisfy their design objectives.

Summing up, an agent is a computational system capable of acting in an autonomous manner. In other words, an agent can find out by itself what he needs to do to satisfy the pre-designed goals, instead of saying explicitly in each moment what do he had to do. Hence, a multi-agent system is the one that consist of a certain number of agents, which interact one with the other, generally communicating by message exchange.

In the most general case, the agents in a multi-agent system will be represented or will act in the name of several users with different objectives and motivations. With the aim of interacting successfully, these agents need the capabilities of cooperate, coordinate and negotiate between them, as the same way as human beings that collaborate, coordinate and negotiate with other human being daily.

2.2.2 MAS Simulation

There are too many variables and unknowns in human society, that only permit to forecast very general trends in a short future time space.

Nevertheless, Multi-agent systems offer an innovative and interesting tool for human societies simulation. The agents could be used to simulate the human behaviour and this would help in the solving of different situations. For example, agents could represent people of a society, electronic institutions or entities to represent social processes. Moreover, in [Gilbert and Conte, 1995] are detailed some of the benefits the MAS related to social processes could have:

- The computer simulation allows the observation of properties of a model, that in principle could be analytically derivable but still is not established.
- Different alternative solutions to a phenomenon observed in nature.
- Properties that are difficult to observe in nature can be observed in simulations in an isolated manner, save them and reproduce them as much as needed.
- The created society can be moulded explicitly, agents can be created with different representations and these properties can be studied afterwards.

To conclude, we are going to use a MAS to model our on-line social network simulator, the pertaining agents will represent human behaviours which will interact between each other. So a coordination mechanism will be needed in the agents interaction, in next section 2.3 we introduce some of them and focus on the used one in section 2.3.1.

2.3 MAS Coordination mechanisms

As is explained in [Wooldridge, 2009], *"Perhaps the defining problem in cooperative working is that of coordination"*. Some coordination mechanisms are essential if the activities of the agents that participate interact between them.

Coordination in multi-agent systems its assumed to occur on execution time, the agents must be able to recognise their relations and when it is necessary to handle them as part of their activities.

Multiple techniques are available in the literature having as objective the dynamic coordination of MAS as:

- Coordination through partial global planning.
- Coordination through joint intentions.
- Coordination by mutual modelling.
- Coordination by norms and social laws.

We are going to focus on the last one and explain it in next section 2.3.1, since it is the technique adopted by the approaches, IRON (section 2.4) and SIMON (section 2.5), used in this work.

2.3.1 Coordination via Norms and Social laws

In our every day lives, we humans use a range of techniques for coordinating activities. One of the most important is the use of *norms* and *social laws* as said in [Lewis David, 1969]. A norm is simply an expected pattern of behaviour; whereas the term social laws carry with them some extra authority the norms does not have.

In MAS simulations conventions play a key role in the social process, they provide agents a behavioural constraint balancing: the individual freedom on the one hand and the goal of the society on the other. They also simplify the decision making process, by dictating some actions to be followed in certain situations. For example, *language* is a convention accepted by the human beings to coordinate our activities with others.

Norms have been widely studied as a mechanism for coordinating MAS [Shoham and Leyton-Brown, 2009; Dignum, 1999; Boella et al., 2006]. Coordination in this sense is usually understood as achieving some system-level goal, such as ensuring that the system avoids certain undesirable states. However, the problem of actually *synthesising* norms that effectively coordinate a multi-agent system is challenging. Since the seminal work of [Shoham and Tennenholtz, 1995], the norm synthesis problem (namely, creating a set of norms which ensures that coordination is successful) has attracted considerable attention. Two approaches for norm synthesis have been considered in the literature: *off-line* and *on-line*.

Off-line approaches (e.g., [Shoham and Tennenholtz, 1995; Fitoussi and Tennenholtz, 1998]) are aimed at synthesising normative systems at design time. Unfortunately, the complexity of the off-line norm synthesis problem is high (NP-hard) [Shoham and Tennenholtz, 1995]. These complexity issues have prompted research into the problem of managing the size of the system state space [Christelis and Rovatsos, 2009]. Unfortunately, even if we ignore the problem of computational complexity, computing norms off-line is not appropriate if the state space of the system is not known in advance, or if it may change over time.

In contrast to off-line approaches, **on-line** approaches are aimed at synthesising norms at run-time rather than design time. The key conceptual advantage of on-line approaches compared to off-line approaches is that on-line approaches are not assumed to require complete knowledge of the system at design time.

Recently, *norm emergence* (or convention emergence) has become a popular technique for on-line norm synthesis (see, e.g., [Christelis and Rovatsos, 2009; Sen and Sen, 2010; Griffiths and Luck, 2010; Salazar et al., 2010; Villatoro et al., 2011; Yu et al., 2013]). Norm emergence does not require any global state representation or centralised control, and considers that agents collaboratively choose their own norms. Norm emergence therefore implies that agents are endowed with the computational capability to synthesise norms, and that they will choose to cooperate in the norm synthesis process. A norm is considered

to have emerged when a significant number of agents in an agent society adhere to a common behaviour (that is, they choose the same actions), which is not dictated by a central authority. Therefore, a key issue in norm emergence is the design of emergence mechanisms that help agents agree on-line (converge) on some norm(s) [Kittock, 1993; Walker and Wooldridge, 1995]. Typically, state-of-the-art norm emergence mechanisms: (i) require an initial set of pre-designed, alternative norms; (ii) are sensitive to such initial conditions; and (iii) mainly converge to a unique norm instead of to a set of norms (with the exception of [Sen and Sen, 2010; Salazar et al., 2010]).

A recent alternative on-line approach is described in [Morales et al., 2011]. There norms are synthesised by observing agent interactions, without requiring their active participation in the synthesis process, unlike state-of-the-art norm emergence mechanisms. These approach proved to be capable of synthesizing a set of norms, instead of a single norm, from *scratch* (namely, without requiring any initial, alternative norms).

On the next Section 2.4 we will present a state-of-the-art *on-line norm synthesis mechanism*: IRON. Afterwards Section 2.5 will introduce an improved version of it, SIMON, overcoming some encountered drawbacks.

2.4 IRON

We now survey IRON [Morales et al., 2013], an abstract and domain-independent on-line norm synthesis mechanism capable of synthesizing normative systems to avoid undesirables states for the given domain. In this work we have extended the mechanism to regulate on-line communities and also overcome some discovered drawbacks. With this aim, IRON synthesises norms that are used by the users of a community to avoid conflicts. Figure 2.4 illustrates the abstract architecture of IRON. In brief, it works by continuously monitoring agents' interactions in a distributed manner through its *sensors*, searching for conflicting situations.

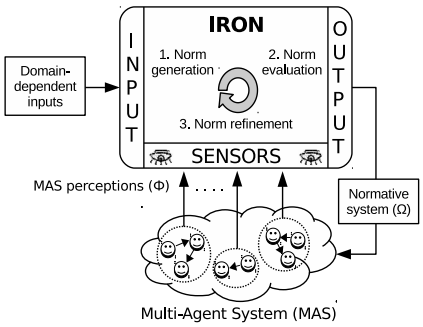

Figure 2.4: IRON 's abstract architecture.

At each system's time step, IRON carries out the overall norm synthesis process throughout three subsequent stages.

1. Whenever it detects new conflicts it performs a *norm generation* process that synthesises norms for the agents of the MAS. These new norms, which regulate agents' behaviour and are aimed at avoiding conflicts in the future, are then communicated to the agents of the MAS.

2. Since at each time step agents choose whether to comply or not with norms, IRON monitors the consequences of such decisions to evaluate norms. In other words, it performs a *norm evaluation* process to assess if norms manage to avoid conflicts or not.

3. It carries out a *norm refinement* process, which: (i) generalises norms when possible, joining several norms to a single parent that concisely represents all of them; and (ii) discards those norms that have not performed well for a period of time.

4. If IRON has made any changes to the normative system, either by adding or removing norms, it sends the new normative system to the agents within the MAS.

Notice then that IRON 's norm synthesis is a *conflict-driven* process. It generates norms whenever new conflicts arise, and it evaluates norms based on the conflicts that arise after agents fulfil or infringe norms. Moreover, it generalises and specialises (deactivates) norms based on their continuous evaluations. Therefore, conflicts also affect norm refinement.

2.4.1 Information model

Consider a Multi-Agent System (MAS) composed of a set of agents $Ag = \{ag_1, \ldots, ag_n\}$ and a set of actions $Ac = \{ac_1, \ldots, ac_c\}$ available to the agents. Agents in the system have their local, *individual context*, which is described in terms of their local point of view (that is, mimicking their perception).

In the norm synthesis process, norms are of the form $\langle \varphi, \theta(ac) \rangle$ where φ is the norm's precondition, θ is a deontic operator (e.g., a *prohibition*) and $ac \in Ac$ is an action available to the agents. The precondition of a norm is a set of first-order n-ary predicates $p(\tau_1, \ldots, \tau_n)$, where p is a predicate symbol and $(\tau_1, \ldots, \tau_n)$ is a set of terms. Moreover, norms are expressed in terms of agents' individual contexts so that they can be understood by them. Hence, whenever the individual context of an agent satisfies the precondition φ of a norm, then the norm applies to the agent and $\theta(ac)$ holds for it. We define a normative system Ω as the set of norms that are currently active in the multi-agent system.

As an example, consider a traffic scenario where agents are driving cars, and conflicting situations are *collisions* between cars. We consider three unary predicate

symbols $\{left, front, right\}$ representing the three road positions that an agent perceives. Each predicate has a single term from $\{police, ambulance, fire-brigade, emergency, nil\}$ representing different types of *emergency* vehicles, and symbol "*nil*" standing for no vehicle. The actions available to agents are $Ac = \{go, stop\}$. With these definitions in place we can create norms such as n_1, n_2, n_3 shown in figure 2.5. All three norms establish a *prohibition* to *go* to an agent that has different types of emergency vehicles to its left position, and nothing to its front and right positions.

$$
\begin{array}{ll}
n_1 : \langle \{ \textit{left}(\textit{police}), & \textit{front}(\textit{nil}),\ \textit{right}(\textit{nil}) \},\ \textit{prh}(\textit{go}) \rangle \\
n_2 : \langle \{ \textit{left}(\textit{ambulance}), & \textit{front}(\textit{nil}),\ \textit{right}(\textit{nil}) \},\ \textit{prh}(\textit{go}) \rangle \\
n_3 : \langle \{ \textit{left}(\textit{fire-brigade}), & \textit{front}(\textit{nil}),\ \textit{right}(\textit{nil}) \},\ \textit{prh}(\textit{go}) \rangle
\end{array}
$$

Figure 2.5: Example of norms synthesized by IRON.

The norm synthesis process is an abstract, domain independent mechanism. However, in order to configure it for different scenarios, it requires some domain-dependant inputs that must be implemented for each specific scenario. In what follows, we will describe IRON 's norm synthesis process (section 2.4.2) as well as the inputs (section 2.4.3) it requires to synthesise norms for a specific scenario.

2.4.2 IRON's norm synthesis process

IRON is based on three components to perform norm synthesis:

1. A *normative network* (NN), which is a graph-based data structure to represent explored norms,

2. A set of normative network *operators* that allow to apply changes to the normative network

3. A *strategy* to apply operators to the normative network.

The right-hand side of Figure 2.6 illustrates the components of IRON. The *control unit* contains the set of operators $\mathcal{O}$, as well as a strategy Π to apply operators. IRON 's strategy applies operators to the normative network in order to retrieve information about norms and to apply changes to the normative network (see *read* and *write* arrows in Figure 2.6). In what follows, we briefly describe these three main components.

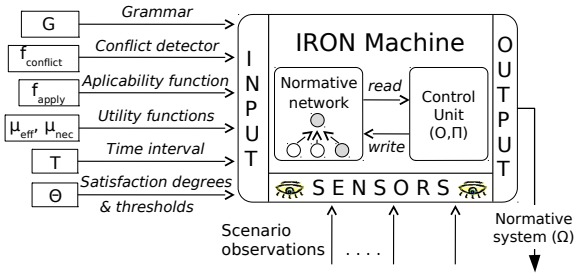

Figure 2.6: IRON 's components and inputs.

2.4.2.1 The Normative Network

IRON represents explored norms by means of the normative network (NN). Specifically, a normative network is a graph-based data structure whose nodes stand for norms and whose edges stand for (generalisation) relationships among norms. In a normative network, norms can be either *active* or *inactive*. The set of active norms in the normative network constitutes the *normative system* (Ω), that is provided to the agents in the scenario. Figure 2.7 illustrates the evolution of a normative network (and its corresponding normative system) over time period $t_i - t_{i+2}$. At time step t_i, the normative network NN_i contains one unique active norm n_1 (represented as a white circle), hence representing the normative system $\Omega_i = \{n_1\}$. At time t_i a new active norm n_2 is created and added to the normative network, hence leading to $NN_{i+1} = \{n_1, n_2\}$ and $\Omega_{i+1} = \{n_1, n_2\}$. Finally, at instant t_{i+2} norm n_2 is deactivated (represented as a gray circle), yielding $NN_{i+2} = \{n_1, n_2\}$ and $\Omega_{i+2} = \{n_1\}$. Notice that at time step t_{i+2} the normative system contains one unique norm n_1, even though the normative network contains two norms n_1, n_2. Recall that a normative network represents a normative system as its active norms, and at time t_{i+2} the only active norm in the normative network is n_1.

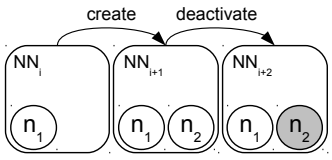

Figure 2.7: Evolution of a Normative Network along time.

We now offer a formal definition of the normative network employed by IRON:

Definition 1 (Normative Network)**.** A Normative Network (NN) is a tuple $\langle \mathcal{N}, R_G, \Delta, \delta \rangle$ where: *(i)* $\mathcal{N} \subseteq \mathbf{N}$ is a subset of our language of norms; *(ii)* $R_G \subseteq \mathcal{N} \times \mathcal{N}$ is a *generalisation* relationship between norms; *(iii)* $\Delta = \{active, inactive\}$ is the set of possible states of a norm; *(iv)* $\delta : \mathcal{N} \rightarrow \Delta$ is a function that returns the state of a norm $n \in \mathcal{N}$.

Since IRON considers that the current *normative system* is composed of the norms that are currently active in the normative network, we define $\Omega = \{n \mid n \in \mathcal{N}, \delta(n) = active\}$.

2.4.2.2 Operators for normative networks

IRON transforms the normative network over time, leading from one normative system into another, searching for a normative system that effectively coordinates the MAS. With this aim, it includes four different operators, more precisely, IRON implements operators to perform:

- The *creation* of a new norm, activating it and adding it then to the normative network.
- The *deactivation* of a norm in the normative network, hence removing it from the normative system.
- The *generalisation* of a group of norms in the normative network to a more general norm that concisely represents all of them.
- As a dual operation to generalisation, the *specialisation* of a general norm to more specific norms.

Next we detail each operator:

1. **Create**. The *create* operator synthesises a new norm from each new detected *conflict*. Next, it employs operators *add* and *activate* to activate the norm and add it to the normative network. Thus, the created norm will be included in the normative system. IRON assumes that a conflict can be avoided if some of the agents' previous actions are not performed. Therefore, it generates norms that prohibit agents to perform such actions in the same conflictive context. Specifically, the *create* operator receives a set of detected $conflicts$ and generates a new norm for each conflict. This generation process is based on an unsupervised version of classical Case-Based Reasoning (CBR) [Aamodt and Plaza, 1994] (more details can be found at [Morales et al., 2013]). Figure 2.7 illustrates the creation of a norm. At time t_i, the normative network NN_i contains one active norm n_1. By applying operator *create*, a new active norm n_2 is added to the normative network, yielding $NN_{i+1} = \{n_1, n_2\}$ and $\Omega_{i+1} = \{n_1, n_2\}$.
   (a) **Add**. The *add* operator adds a norm to the normative network so that IRON can keep track of its ability to avoid conflicts.
   (b) **Activate**. As described above, norms in a normative network may be either *active* or *inactive*. Operator *activate* sets the state of a norm in the normative network to *active* so that it belongs to the normative system.

2. **Deactivate**. Consider that at a given time IRON detects that a norm does not succeed in avoiding conflicts, and hence must be removed from the normative system. IRON will not remove the norm from the normative network, but it will use operator *deactivate* to set the state of the norm to *inactive*. Therefore, the norm will no longer belong to the normative system but the normative network will still have it to keep track of its exploration. Again, Figure 2.7 depicts the normative network NN_{i+1}, which at time t_{i+1} contains two active norms n_1, n_2. Then, operator *deactivate* deactivates norm n_2, yielding $NN_{i+2} = \{n_1, n_2\}$ and $\Omega_{i+2} = \{n_1\}$.

3. **Generalise**. As part of the norm refinement process, IRON uses operator generalise to represent several norms as a more general, single norm that implicitly includes all of them. Specifically, norm generalisations can be performed for any norm in the normative network. Thus, by means of norm generalisations IRON reduces the cardinality of synthesised normative systems. As an example, consider norms n_1, n_2, n_3 described in figure 2.5. All these three norms can be generalised to the following norm:

$$n_4 : \langle \{left(emergency), front(nil), right(nil)\},\ prh(go) \rangle$$

Figure 2.8: Generalisation of example norms.

Norm n_4 prohibits an agent to go whenever it perceives any type of *emergency* vehicle (either police, ambulance or fire-brigade) to its left position, and nothing to its front and right positions. Figure 2.9 illustrates the generalisation of n_1, n_2, n_3 to n_4. At time t_i, the normative network NN_i contains three active norms n_1, n_2, n_3. Then, IRON generalises them to n_4 by performing the following steps: (i) it generates norm n_4, (ii) it activates n_4 and adds it to the normative network, (iii) it establishes generalisation relationships from n_1, n_2, n_3 to n_4, and (iv) it deactivates norms n_1, n_2, n_3, removing them from the normative system.

4. **Specialise**. Operator *specialise* undoes a norm generalisation, specialising a general norm into a set of more specific norms. As an example, consider the normative network NN_i depicted in Figure 2.10. It represents normative system $\Omega_i = \{n_4\}$, containing one single norm which is applicable in the specific situations described by norms n_1, n_2, n_3. Consider now norm n_2 does not succeed in avoiding conflicts, and hence it must be removed from the normative system. Even though n_2 does not belong to the normative system, it is implicitly represented by n_4. Therefore, operator *specialise* specialises n_4, deactivating it and activating n_1, n_3 in the normative network. Thus, after the norm specialisation, the normative system becomes $NN_{i+1} = \{n_1, n_3\}$, which no longer contains the situation described by n_2.

Algorithm 1 IRON's norm synthesis strategy

```
1: function IRONSTRATEGY(⟨s_{t-1}, s_t⟩, NN, G, f_apply, f_conflict, μ_eff, μ_nec, Θ, T_w)
2:     conflictDescription ← conflictDetection(⟨s_{t-1}, s_t⟩, f_conflict)
3:     NN ← normGeneration(NN, conflictDescription, G)
4:     P ← normEvaluation(NN, ⟨s_{t-1}, s_t⟩, f_apply, f_conflict, μ_eff, μ_nec, T_w)
5:     NN ← normRefinement(NN, P, G, Θ)
6:     Ω ← {n ∈ NN | δ'(n) = active}
7:     return Ω
8: end function
```

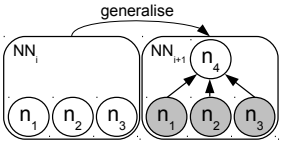

Figure 2.9: Generalisation of norms n_1, n_2, n_3 to a new norm n_4.

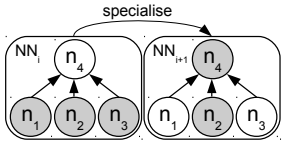

Figure 2.10: Specialisation of norm n_4 to its child norms n_1, n_3.

2.4.2.3 IRON's strategy

IRON invokes previous operators by following a specific *strategy* to perform the norm synthesis process. Specifically, the proposal of [Morales et al., 2013] is to monitor the evolution of the system at regular time intervals and apply operators under certain conditions. Algorithm 1 describes in outline IRON's overall norm synthesis strategy. Since IRON is an on-line mechanism, at every tick it runs its strategy to perform three main tasks, namely: *norm generation, norm evaluation* and *norm refinement*. Algorithm 1 specifies IRON's general strategy. It is domain-independent and takes as input:

- A pair $\langle s_{t-1}, s_t \rangle$ containing descriptions of the system state at time $t-1$ and time t, respectively. This pair stands for a transition between the system state at consecutive times. In fact, the differences between s_{t-1} and s_t reflect the local changes that occurred when the system evolved from $t-1$ to t.
- A normative network *NN*, which includes the current normative system Ω.
- A grammar $\mathcal{G}$, including the subsumption relationships between its terms;
- A function f_{apply} to check norm applicability in the current system state s.
- A function $f_{conflict}$ to detect if a given system state s is undesired.
- Two evaluation functions μ_{eff}, μ_{nec} to assess the effectiveness and necessity of norms in *NN*.
- Θ, *a set of satisfaction degree thresholds* ($\Theta = \{\alpha^{gen}_{eff}, \alpha^{gen}_{nec}, \alpha^{esp}_{eff}, \alpha^{esp}_{nec}\}$).

- *A time period* T_w.

The strategy is on-line and conflict-driven (since it is aimed to avoid undesired, conflicting states), and thus, at every tick, it starts by searching for conflicts in the current system state. Function *conflictDetection* (line 2) uses the domain-dependent function $f_{conflict}$ to assess if the current system state s_t is undesired (i.e., it detects undesired states $C \subset S$). In case that $s_t \in C$, it returns a $conflictDescription$ that incorporates descriptions s_{t-1} and s_t, together with the identifiers of those agents whose actions lead to the undesired state s_t (for instance, in a traffic scenario, those cars that went forward before colliding). Next, the *normGeneration* function (line 3) synthesises a norm to avoid the transition from s_{t-1} to s_t (though disregarding the generation of general norms) in order to avoid it in the future. Subsequently, *normEvaluation* in line 4 evaluates norms in terms of their effectiveness and necessity. Finally, the norm refinement function in line 5 generalises and/or specialises norms according to their effectiveness and necessity ranges during the time period T_w. The algorithm outputs a normative system (line 7), which is the aggregation of all the active norms of the Normative Network (line 6), for the agents in the domain that IRON is aiming at regulating.

We now explain in detail the three main tasks of IRON, namely: *norm generation, norm evaluation* and *norm refinement*.

1. **Norm generation**. During this phase, IRON first monitors the multi-agent system operation through a set of distributed *sensors*, searching for conflicts. As depicted in Figure 2.6, IRON represents agents' perceived interactions in the form of *observations*, which are partial descriptions of the scenario from a global, external observer's perspective. Then, it performs conflict detection within perceived *observations*. Whenever conflicts arise, it invokes the previously described operator *create* to generate norms that regulate agents' behaviour in order to avoid detected conflicts in the future. Recall that, since agents must be able to understand norms, IRON describes norms from an agent's local perspective.

2. **Norm evaluation**. At each time step, some norms may apply to the agents. In this case, agents decide whether to fulfil norms or infringe them. During the norm evaluation stage, IRON monitors the effects of such decisions in order to assess if norms succeed in avoiding conflicts. With this aim, it determines if agents have fulfilled or infringed norms, and which of these fulfilments or infringements have led to conflicts. Then, norms are evaluated in terms of their *effectiveness* and *necessity*, represented as μ_{eff} and μ_{nec}. On the one hand, IRON measures the effectiveness μ_{eff} of a norm from the outcomes of its *fulfilments*: the higher the ratio of *successful* fulfilments (fulfilments that did not end up with conflicts), the more effective the norm. On the other hand, it measures the necessity μ_{nec} of a norm according to the following principle: the higher the ratio of *harmful infringements* (infringements leading to conflicts), the more necessary the norm. Finally, IRON computes the effectiveness and necessity

ranges of norms during a period of time T. Specifically, a range of a norm contains a lower and upper bound for the range of values of effectiveness or necessity (μ_{eff}, μ_{nec}) of the norm during T. These ranges will be used to refine and evaluate norms, in Section 2.4.3.4 we detail how we evaluate the norms.

3. **Norm refinement**. The norm refinement process yields a new normative system transforming the normative network, deactivating ineffective or unnecessary norms, and performing norm generalisations and specialisations. On the one hand, it generalises a norm whenever the lower bound of its effectiveness *and* necessity ranges are over a generalisation threshold α_{gen}. On the other hand it specialises (deactivates) a norm whenever the upper bound of its effectiveness *or* necessity ranges are under a specialisation threshold α_{spec}.

2.4.3 IRON's inputs

As detailed above, IRON 's norm synthesis is an abstract, domain-independent mechanism. However, during the norm synthesis process it requires some domain information:

- A grammar $\mathcal{G}$ to define norms for the given scenario.
- A conflict detection function $f_{conflict}$ that allows to detect conflicts in perceived *observations*.
- A norm applicability function f_{apply} to determine whether a norm applies to the agents in a perceived *observation*.
- Norm evaluation functions to compute the effectiveness and necessity of norms in the normative network.
- A set of configuration parameters composed of a time interval (T) to compute effectiveness and necessity ranges of norms during a period of time T, as well as a set of thresholds to define the acceptable range of effectiveness and necessity of norms, to consider to generalise and specialise norms.

In the next section we are going to detail these domain information for the norm synthesis machine.

2.4.3.1 A Grammar for norm synthesis

The first input IRON requires is a grammar to synthesise norms of the form $\langle \varphi, \theta(Ac) \rangle$ for the given scenario. IRON adapts its grammar from [García-Camino et al., 2009], using as building blocks *atomic formulae* of the form $p^n(\tau_1, \ldots, \tau_n)$, p being an n-ary predicate symbol and $\tau_1, \ldots, \tau_n$ terms of an agents' language that describes agents' individual contexts.

$$
\begin{array}{lll}
\text{Norm} & ::= & \langle \varphi, \theta(Ac) \rangle \\
\varphi & ::= & \varphi \,\&\, \varphi \mid \alpha \\
\theta & ::= & obl \mid prh \\
Ac & ::= & ac_1 \mid ac_2 \mid \ldots \mid ac_n \\
\alpha & ::= & p^n(\tau_1, \ldots, \tau_n)
\end{array}
$$

Figure 2.11: Abstract grammar for norm synthesis.

In order to configure IRON to synthesise norms for a given scenario, this abstract grammar must be instantiated, specifying the predicates and terms that are considered for that particular scenario.

2.4.3.2 A function for conflict detection

The definition of conflict is domain-dependant. For instance, in a traffic scenario a conflicting situation may be a collision among cars, while in an on-line community scenario a conflict may be defined as a user that uploads inappropriate contents (e.g., uploading a `spam` content), hence leading other users to complain about it. Therefore, IRON requires as an input a function $f_{conflict}$ to detect conflicts for a given scenario. Specifically, function $f_{conflict}$ receives as an input a set of perceived *observations*, and returns a set of $conflicts$ that it has detected within these *observations*.

2.4.3.3 A function to detect norm applicability

As described in the strategy in Section 2.4.2.3, IRON evaluates norms based on the consequences of agents' norm fulfilments and infringements. With this aim, function f_{apply} receives as an input a set of perceived *observations*, and returns a set of norms that apply in the situations described by the *observations*. Specifically, this function operates as follows. First, it computes the individual context of each agent in the *observation*. Second, for each agent's individual context, it retrieves which norms apply to that context. As an example, consider an agent which perceives a police car to its left, and nothing to its front and right positions. In this specific individual context, norm n_1 applies to the agent and hence the agent is forbidden to go.

2.4.3.4 Functions to evaluate norms

During norm evaluation, IRON requires as an input two functions μ_{eff} and μ_{nec} to evaluate the effectiveness and necessity of norms. In fact, these functions are not really scenario-dependant. They evaluate norms based on the conflicts (in general) that arise after agents fulfil or infringe norms. However, IRON provides two default utility functions μ_{eff} and μ_{nec} that may be replaced by other functions explicitly implemented for a specific scenario. IRON's default utility functions evaluate norms along the lines of the explanation of norm evaluation

in section 2.4.2.3. Function μ_{eff} computes the effectiveness of a norm as its ratio of successful fulfilments. Function μ_{nec} computes the necessity of a norm as its ratio of harmful infringements. The definition of default formulas μ_{eff}, μ_{nec} correspond to the formulas 1, 2, 3, 4 in [Morales et al., 2013]. These are also explain in next Chapter 3 on Section 3.2.2.2.

2.4.3.5 Configuration parameters

Recall from section 2.4.2.3 that IRON refines the normative system deactivating, generalising and specialising norms, based on their effectiveness and necessity ranges and a set of generalisation and specialisation thresholds. Therefore, IRON requires as an input parameter T to compute ranges, as well as thresholds $\alpha_{gen}, \alpha_{spec}$ to generalise and specialise norms. A low time interval T, will make IRON to be more *reactive* to changes in effectiveness and necessity ranges, refining the normative system in consequence. By contrast, large time intervals T will make IRON to be more *conservative* to refine the normative system, since effectiveness and necessity ranges will be less reactive to changes. Moreover, the greater the generalisation threshold is, the more conservative IRON is about generalising norms. Finally, the lower the specialisation threshold is, the more conservative IRON will be about deactivating norms.

2.5 SIMON

As argued in [Morales et al., 2014a], IRON has some drawbacks. First, IRON misses out on *compactness* issues: there are no metrics on *minimality* and *simplicity* of the synthesized normative system. Indeed, there is a lack of literature on experimental analysis of compactness, and work so far has only addressed compactness issues from a theoretical perspective [Fitoussi and Tennenholtz, 2000]. Additionally, IRON 's generalisation is highly conservative. IRON requires that *all* the children of a potential generalisation (i) **exist**, (ii) **are active** and (iii) **perform well** in order to generalise norms.

In this section is introduce SIMON (*SI*mple *M*inimal *O*n-line *N*orm Synthesis), the approach of norm synthesis presented by [Morales et al., 2014a], which incorporates an alternative technique for norm generalisation that increases the compactness of synthesised normative systems. SIMON is based on IRON, but it tries to avoid the above mentioned drawbacks. The following sections will talk about the improvements of SIMON (2.5.1) with respect to IRON and the extra inputs these improvements require in order to operate (2.5.2).

2.5.1 Improvements of SIMON

SIMON tries to tackle the drawbacks of IRON by using (i) an ontology, and by improving the (ii) norm generalisation, (iii) strategy and (iv) norm evaluation

processes in IRON. We now focus on these improvements.

2.5.1.1 Ontology

The ontology introduced by SIMON is a directed tree rooted at a most general term and whose edges capture generalisation relationships. It uses the same language used by IRON (explained in Section 2.4.1). Moreover, SIMON denotes the set of terms in the language by $\mathcal{T}$ and defines a relationship between the terms in $\mathcal{T}$ such that if $\tau, \tau' \in \mathcal{T}$ and $\tau' \leq \tau$, we say that τ is more general than τ'. There is also a single term $\tau_0 \in \mathcal{T}$, called the *most general term*, which is not generalised by any other term.

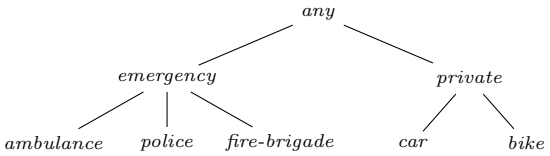

Figure 2.12: Relationships between terms, ontology.

Figure 2.12 illustrates an example ontology rooted at the "*any*" term (most general term, τ_0); term "*emergency*" is more general than "*ambulance*", and term "*any*" is more general than "*emergency*", that is, *ambulance* $\sqsubseteq$ *emergency* $\sqsubseteq$ *any*.

2.5.1.2 Preliminary definitions

To understand the generalisation mechanism of SIMON as described in [Morales et al., 2014a], some preliminary definitions are required:

1. Subsumption of terms

We say that τ subsumes τ', denoted as $\tau' \sqsubseteq \tau$, if the term τ is more general than τ'. For example, "*any*" subsumes terms "*emergency*" and "*ambulance*", that is, *emergency* $\sqsubseteq$ *any* and *ambulance* $\sqsubseteq$ *emergency*. Formally:

Definition 2 (Terms subsumption)**.** Let $\tau, \tau' \in \mathsf{T}$ be two terms. We say that τ subsumes τ', denoted as $\tau' \sqsubseteq \tau$, iff there is a (possibly empty) sequence of terms $\tau'_0, \ldots, \tau'_m$ such that $\tau' \leq \tau'_0 \leq \cdots \leq \tau'_m \leq \tau$.

2. Intersection of terms

Considering that the ontology of terms has a tree structure, the *intersection* between two terms is the most specific term subsuming these two terms. For instance, the intersection between "*ambulance*" and "*emergency*" is the term "*ambulance*", that is the most specific term. Formally:

Definition 3 (Terms intersection)**.** For $\tau, \tau' \in \mathcal{T}$, their intersection $\tau \sqcap_t \tau'$ is:

$$\tau \sqcap_t \tau' = \begin{cases} \tau & \text{if } \tau \sqsubseteq \tau' \\ \tau' & \text{if } \tau' \sqsubseteq \tau \\ \emptyset & \text{otherwise} \end{cases}$$

3. Intersection of predicates

The intersection between two predicates $p(\bar{\tau}), p(\bar{\tau}') \in \mathcal{L}_{Ag}$ is another predicate with the intersection of each corresponding pair of terms in $\bar{\tau}, \bar{\tau}'$, whenever such intersection exists for all of them and the predicates are equal. Formally:

Definition 4 (Predicates intersection)**.** For $p(\bar{\boldsymbol{\tau}}), p(\bar{\boldsymbol{\tau}}') \in \mathcal{L}_{Ag}$, if $\tau_i \sqcap_t \tau_i' \neq \emptyset, 1 \leq i \leq n$, then their intersection $p(\bar{\boldsymbol{\tau}}) \sqcap_\pi p(\bar{\boldsymbol{\tau}}')$ is $p(\bar{\boldsymbol{\tau}}'')$ such that $\tau_i'' = \tau_i \sqcap_t \tau_i'$ for all $1 \leq i \leq n$.

Following the examples of figure 2.12, the intersection between *left(ambulance)* and *left(emergency)* is *left(ambulance)*.

4. Generalisation of terms

Inspired by the anti-unification of terms proposed in [Armengol and Plaza, 2000], the most specific generalisation of terms, $\tau, \tau' \in \mathcal{T}, \tau \neq \tau'$, is the most specific term (τ_s) that *strictly* subsumes both of them, $\tau \sqsubset \tau_s$ and $\tau' \sqsubset \tau_s$. Formally:

Definition 5 (Generalisation of terms)**.** For $\tau, \tau' \in \mathcal{T}, \tau \neq \tau'$, their *most specific generalisation*, denoted as $\tau \sqcup_t \tau'$, is a term $\tau_s \in \mathcal{T}$ such that $\tau \sqsubset \tau_s$ and $\tau' \sqsubset \tau_s$, and $\nexists \tau'' \in \mathcal{T}$ such that $\tau \sqsubset \tau'', \tau' \sqsubset \tau''$ and $\tau'' \sqsubset \tau_s$.

For instance, the most specific generalisation of "*ambulance*" and "*car*" is "*any*", since there is no other term which is more specific and strictly subsumes both of them. On the other hand, the most specific generalisation of "*car*" and "*bike*" is "*private*", as we can observe in Figure 2.12.

5. Generalisation of predicates

The most specific generalisation of predicates is followed by the generalisation of terms. If the predicates are equal and the terms generalisable, as seen in the previous definition, the most specific generalisation of predicates will be the same predicate with the specific generalised term. Formally:

Definition 6 (Generalisation of predicates)**.** Predicates $p(\bar{\boldsymbol{\tau}}), p(\bar{\boldsymbol{\tau}}') \in \mathcal{L}_{Ag}$ have a *most specific generalisation* iff $\tau_i \sqcup_t \tau_i' \neq \emptyset, 1 \leq i \leq m$. Their most specific generalisation, denoted as $p(\bar{\boldsymbol{\tau}}) \sqcup_p p(\bar{\boldsymbol{\tau}}')$, is another predicate $p(\bar{\boldsymbol{\tau}}'')$ such that $\tau_i'' = \tau_i \sqcup_t \tau_i', 1 \leq i \leq m$.

For example, the most specific predicate generalisation for "*left(ambulance)*" and "*left(police)*" is "*left(emergency)*".

6. Generalisation between norms

Finally, there is a generalisation relationship between norms $\mathbf{n}$ and $\mathbf{n}'$ when, for each predicate there is an equivalent predicate, and the term is equal or subsumed. Formally:

Definition 7 (Norms generalisation relationship)**.** $\mathbf{n} = \langle \varphi, \theta(ac) \rangle$ is more general than $\mathbf{n}' = \langle \varphi', \theta(ac) \rangle$, denoted as $n' \subseteq n$, iff $|\varphi| = |\varphi'|$, and for each predicate $p(\bar{\boldsymbol{\tau}}') \in \varphi'$, there is a predicate $p(\bar{\boldsymbol{\tau}}) \in \varphi, \tau_i' \sqsubseteq \tau_i, 1 \leq i \leq m$.

$$
\begin{array}{ll}
n : \langle \{ \mathit{left}(\mathit{emergency}), & \mathit{front}(\mathit{nil}),\ \mathit{right}(\mathit{nil}) \},\ \mathit{prh}(\mathit{go}) \rangle \\
n' : \langle \{ \mathit{left}(\mathit{ambulance}), & \mathit{front}(\mathit{nil}),\ \mathit{right}(\mathit{nil}) \},\ \mathit{prh}(\mathit{go}) \rangle
\end{array}
$$

Figure 2.13: Example of relationship between norms.

For instance, in figure 2.13 we show norms $\mathbf{n}$ and $\mathbf{n}'$, where $\mathbf{n}$ is more general than $\mathbf{n}'$ because for all the predicates there is an equivalent predicate and, in the case of the first predicate, a more general one, namely:

$$
\begin{array}{c}
\text{“}\mathit{left(emergency)}\text{”} \sqsubseteq \text{“}\mathit{left(ambulance)}\text{”} \\
\text{“}\mathit{front(nil)}\text{”} = \text{“}\mathit{front(nil)}\text{”} \\
\text{“}\mathit{right(nil)}\text{”} = \text{“}\mathit{right(nil)}\text{”}
\end{array}
$$

2.5.1.3 Norm Generalisation

SIMON 's norm generalisation consists of three phases:

i) **Monitoring** when the norms of the NS start performing well.

ii) **Checking** if the identified norms are *generalisable* with the rest of the norms.

iii) **Generalising** norms if possible.

Specifically, SIMON first monitors if the effectiveness and necessity performance values, explained in section 2.4.3.4, surpass the generalisation thresholds during the current time period. Second, for each norm that starts performing well, it checks if it is generalisable with another norm in the NS, the active ones (notice that IRON considers all the norms in the NN). Finally, in case two norms are generalisable, SIMON generalises them to the *most specific general norm* (their parent norm).

So, two norms, $\mathbf{n}$ and $\mathbf{n}'$, will be generalisable for SIMON if for each predicate of $\mathbf{n}$ exists a predicate in $\mathbf{n}'$ such that: it (i) intersects or (ii) subsumes both predicates. Formally:

Definition 8 (Generalisable norms)**.** We say that two norms $n = \langle \varphi, \theta(ac) \rangle$ and $n' = \langle \varphi', \theta(ac) \rangle$, $n \neq n'$, are generalisable iff (i) there is at least one predicate $p(\bar{\boldsymbol{\tau}}') \in \varphi', p(\bar{\boldsymbol{\tau}}) \sqcup_p p(\bar{\boldsymbol{\tau}}') \neq \emptyset$; and (ii) for each remaining predicate $p(\bar{\boldsymbol{\tau}}) \in \varphi$, there is an equal predicate $p(\bar{\boldsymbol{\tau}}) \in \varphi'$.

$$
\begin{aligned}
n_1 &: \langle \{ \textit{left}(\textit{police}), && \textit{front}(\textit{police}), && \textit{right}(\textit{car}) \}, && \textit{prh}(\textit{go}) \rangle \\
n_2 &: \langle \{ \textit{left}(\textit{ambulance}), && \textit{front}(\textit{police}), && \textit{right}(\textit{car}) \}, && \textit{prh}(\textit{go}) \rangle \\
n_3 &: \langle \{ \textit{left}(\textit{emergency}), && \textit{front}(\textit{police}), && \textit{right}(\textit{car}) \}, && \textit{prh}(\textit{go}) \rangle \\
n_4 &: \langle \{ \textit{left}(\textit{fire-brigade}), && \textit{front}(\textit{police}), && \textit{right}(\textit{car}) \}, && \textit{prh}(\textit{go}) \rangle \\
n_5 &: \langle \{ \textit{left}(\textit{fire-brigade}), && \textit{front}(\textit{police}), && \textit{right}(\textit{police}) \}, && \textit{prh}(\textit{go}) \rangle \\
n_6 &: \langle \{ \textit{left}(\textit{fire-brigade}), && \textit{front}(\textit{police}), && \textit{right}(\textit{any}) \}, && \textit{prh}(\textit{go}) \rangle
\end{aligned}
$$

Figure 2.14: Example of generalisation between norms.

For instance, norms $\mathbf{n}_1$ and $\mathbf{n}_2$, shown in figure 2.14, satisfy the conditions of generalisation because:

1. Predicate $\textit{left}(\textit{police}) \in \varphi_1$ has a corresponding predicate $\textit{left}(\textit{ambulance}) \in \varphi_2$, these two predicates are subsumed by the term *emergency*:

$$\textit{left}(\textit{police}) \sqcup_p \textit{left}(\textit{ambulance}) = \textit{left}(\textit{emergency}).$$

2. Predicate $\textit{front}(\textit{police}) \in \varphi_1$ has $\textit{front}(\textit{police}) \in \varphi_2$.

3. Predicate $\textit{right}(\textit{car}) \in \varphi_1$ has $\textit{right}(\textit{car}) \in \varphi_2$.

Hence, $\mathbf{n}_1$ and $\mathbf{n}_2$ are generalisable to a new norm that can be $\mathbf{n}_3$, which it is assessed as follows:

Definition 9 (Norm generalisation)**.** Two generalisable norms $n = \langle \varphi, \theta(ac) \rangle$,-$n' = \langle \varphi', \theta(ac) \rangle$ can be generalised to a norm $n'' = \langle \varphi'', \theta(ac) \rangle$ such that, for each predicate $p(\bar{\boldsymbol{\tau}}) \in \varphi$ and $p(\bar{\boldsymbol{\tau}}') \in \varphi'$, there is a predicate $p(\bar{\boldsymbol{\tau}}'') \in \varphi''$ obtained as:

$$p(\bar{\boldsymbol{\tau}}'') = \begin{cases} p(\bar{\boldsymbol{\tau}}) & \text{if } \tau_i = \tau_i', \forall i \in [1, m] \\ p(\bar{\boldsymbol{\tau}}) \sqcup_p p(\bar{\boldsymbol{\tau}}') & \text{otherwise} \end{cases}$$

Following the previous example:

$$
\begin{aligned}
\textit{left}(\textit{police}) \sqcup_p \textit{left}(\textit{ambulance}) &= \textit{left}(\textit{emergency}) \\
\textit{front}(\textit{police}) \in \varphi_1 &= \textit{front}(\textit{police}) \in \varphi_2. \\
\textit{right}(\textit{car}) \in \varphi_1 &= \textit{right}(\textit{car}) \in \varphi_2.
\end{aligned}
$$

Moreover, SIMON generalises norms with *partial evidence*, while IRON requires *full evidence*. As an example SIMON can synthesize norm $\mathbf{n}_3$ from $\mathbf{n}_1$ and $\mathbf{n}_2$, even though $\mathbf{n}_4$ has never been synthesized. In contrast, IRON will synthesize norm $\mathbf{n}_3$ only whenever $\mathbf{n}_1$, $\mathbf{n}_2$ and $\mathbf{n}_4$ have been synthesized. Therefore, it is said that SIMON takes an *optimistic approach* to generalisation that requires less

information than IRON, and synthesizes more compact normative systems.

SIMON was also created with two generalising operation modes, namely *Shallow* and *Deep*. On the one hand, using *shallow* mode the generalisation will occur in a *direct* manner, whenever the terms of the predicates are equal except the term to generalise, which is the subsumed one. As we can observe in the previous example $\mathbf{n}_1$ and $\mathbf{n}_2$, using the *shallow* generalisation mode, will generalise to $\mathbf{n}_3$.

On the other hand, *deep* mode will generalise norms in an *indirect* way, whenever the terms of the predicate are equal or intersect between them, except the generalized term that will be the subsumed one.

As an example of a *deep* generalisation, we can take $\mathbf{n}_5$ and $\mathbf{n}_3$, depicted in figure 2.14, and see how the *deep* mode can generalise while *shallow* mode can not. In this occasion:

1. For the first predicate $\mathit{left}(\mathit{fire\text{-}brigade}) \in \varphi_5$ there is a predicate $\mathit{left}(\mathit{emergency}) \in \varphi_3$, so $\mathit{left}(\mathit{fire\text{-}brigade}) \sqcap_p \mathit{left}(\mathit{emergency}) = \mathit{left}(\mathit{fire\text{-}brigade})$.
2. For the second predicate $\mathit{front}(\mathit{police}) \in \varphi_5$, there is a predicate $\mathit{front}(\mathit{police}) \in \varphi_3$, that yields to $\mathit{front}(\mathit{police}) \sqcap_p \mathit{front}(\mathit{police}) = \mathit{front}(\mathit{police})$.
3. For predicate $\mathit{right}(\mathit{police}) \in \varphi_5$ there is a predicate $\mathit{right}(\mathit{car}) \in \varphi_3$, $\mathit{right}(\mathit{car}) \sqcap_p \mathit{right}(\mathit{police}) = \emptyset$ and $\mathit{right}(\mathit{car}) \sqcup_p \mathit{right}(\mathit{police}) = \mathit{right}(\mathit{any})$.

Therefore, norms $\mathbf{n}_3$ and $\mathbf{n}_5$ are generalised, in *deep* mode, to $\mathbf{n}_6$.

2.5.1.4 SIMON's strategy

We have so far described a new approach to perform and to revise/backtrack norm generalisations. Next, we introduce a novel strategy for norm synthesis, using optimistic norm generalisation. Since SIMON is an on-line method, its norm synthesis strategy is continuously executed. SIMON performs conflict detection and synthesises new norms as described in Section 2.4 (IRON). Crucially, the norm evaluation and norm refinement phases are novel. In addition to evaluating norms, the norm evaluation phase synthesises under-performing norms that have never been synthesised but are implicitly represented by general norms. Finally, norm refinement generalises norms taking the optimistic generalisation approach described in the previous section Norm Generalisation 2.5.1.3, and specialises norms as described in IRON 's Section 2.4.2.

Algorithm 2 describes the new strategy, where, for each detected conflict (line 2) it creates new norms (line 3) aimed at avoiding conflicts in the future. Then, norm evaluation evaluates applicable norms (lines 4–5) and returns norm performances P and a set of *negatively rewarded norms* (NRN), namely those underperforming norms that do not exist but are implicitly represented by general norms.

Next, it carries out norm refinement. First, it adds to the normative network

Algorithm 2 SIMON's norm synthesis strategy

```
1:  function simonStrategy(views, NN, O, mode, step)
2:      conflicts ← conflictDetection(views)
3:      (createdNorms, NN) = normCreation(conflicts, NN, O)
4:      applicableNorms ← normApplicability(views, NN)
5:      (P, NRN) ← normEvaluation(applicableNorms)
6:      for each n ∈ NRN do
7:          NN ← add(NN, n)
8:          NS ← getNormativeSystem(NN)
9:          for each n' ∈ NS do
10:             NN ← searchRelationships(NN, O, n, n', null, ∅)
11:         end for
12:     end for
13:     for each n ∈ applicableNorms do
14:         if crossedGeneralisationThreshold(n, P) then
15:             if mode = s-simon then
16:                 NN ← ShallowNormGeneralisation(NN, O, n)
17:             else
18:                 if mode = d-simon then
19:                     NN ← DeepNormGeneralisation(NN, O, n)
20:                 end if
21:             end if
22:         end if
23:         if crossedSpecialisationThreshold(n, P) then
24:             NN ← normSpecialisation(NN, n, P)
25:         end if
26:     end for
27:     return NN
28: end function
```

each norm in *NRN* (lines 7–8), and searches for their possible relationships with other norms in the *NN* (lines 9–10). Second, it performs the optimistic generalisation of norms described in the previous Section 2.5.1.3 whenever it detects that they start performing well (lines 15-21). The *mode* parameter determines whether to invoke our *Shallow* or *Deep* generalisation methods. Third, it specialises norms whenever it detects they have just become ineffective or unnecessary (lines 23–24).

2.5.1.5 Normative system evaluation

In addition to the effectiveness and necessity measures introduced in IRON, SIMON provides two further metrics introduced by [Fitoussi and Tennenholtz, 2000], namely minimality and simplicity. *Minimality* is concerned with minimising the amount of constraints (in a normative system) imposed on agents. The more minimal a normative system, the greater the individual agent freedom.

On the other hand, *simplicity* refers to norms that are easy to reason about by agents. The simpler the norms, the less computational resources required to reason about them. We note, therefore, that minimality and simplicity are local (agent-level) synthesis criteria, aimed at simplifying the reasoning of individual agents. Both concepts are naturally captured by measuring the size of a normative system (minimality) and its number of clauses (simplicity). So the minimality of a *NS* is $\mathcal{M}(NS) = |NS|$ and the simplicity of a *NS* is $\mathcal{S}(NS) = \sum_{\langle \varphi, \theta(ac) \rangle \in NS} |\varphi|$.

These two measures are key to the problem at hand (i.e., the on-line synthesis of normative systems) since the smaller the minimality and simplicity of normative systems, the better it is so as to give agents flexibility, to save the agents' computational resources (when processing norms), and to avoid over-regulation.

2.5.2 SIMON's extra inputs

Once explained the improvements of SIMON covering the drawbacks of IRON, we also have to comment the extra inputs, apart from the ones explained for IRON, that we have to apply in order to execute SIMON.

2.5.2.1 Operation modes

We recall from section 2.5.1.3 that SIMON offers two operation modes namely: *Shallow* (S-SIMON) and *Deep* (D-SIMON). These two modes affect the way in which SIMON generalise norms. On the one hand, having a direct generalization with *Shallow* mode which will generalise norms if all the predicates are equal except the generalised one. On the other hand, the *Deep* mode will make an indirect generalization which will be able to generalise norms with predicates intersecting between them apart from the one to generalise.

2.5.2.2 Generalisation steps

Another extra input of SIMON is the one called generalization steps. The aim of this parameter is to select the total number of clauses a generalisation can involve. A generalisation can involve from 1 clause up to the total number of clauses in the precondition of a norm.

According to the examples that we have used so far, the generalisation step, k, can vary between [1..3]. For instance, setting $k = 2$ means that a generalisation can involve both *left* and *front* clauses. Moreover, we have to observe that IRON can perform only generalisations of a single predicate, which will be the equivalent of fixing the generalisation step to 1.

2.6 Summary

This chapter explains the necessary background to the comprehension of this thesis. To sum up, we explain different state-of-the-art moderation mechanisms, from them we observed that no one is independent to human moderation. So being our approach a state-of-the-art improvement. Furthermore, since our on-line community simulator is based on MAS and the used coordination mechanism is based on norms, a standard technique in MAS coordination, in this background we describe both of the concepts. Moreover, we introduce two state-of-the-art on-line norm synthesis approaches: namely IRON and SIMON. IRON is an abstract and domain independent on-line norm synthesis approach capable of

synthesising norms to avoid undesirable states for a given domain. Recall the abstract computational model of it:

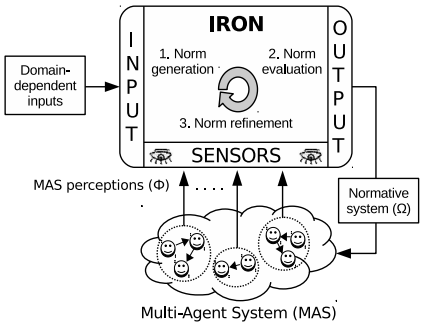

Figure 2.15: Abstract computational model.

As we can observe in Figure 2.15 this model is constantly perceiving observations from a MAS domain, creating norms from those perceptions and sending them to the MAS to be aware of the current legislation. SIMON is a second version of the IRON norm synthesis machine that overcomes it in terms of compactness. The computational model followed by IRON is also preserved in SIMON.

In the next Chapter 3 we present our novel deliberative norm synthesis machine constructed to capture the consensus of the community users. The computational model is preserved as it is also based on SIMON. Chapter 4 introduces the created MAS simulator to connect with the norm synthesis machine, as depicted in Figure 2.15. Moreover, the connection between the two modules is explained in Chapter 5 and, finally, Chapter 6 concludes evaluating the different norm synthesis approaches with our multi-agent social network simulator.

Chapter 3

Improving Norm Synthesis

This chapter introduces our first contribution of the thesis. As commented in Chapter 2, IRON has some drawbacks that SIMON overcomes. These two state-of-the-art approaches were used with a traffic domain, which needs reactive approaches for norm generation. But as we are studying the implementation of a novel on-line community simulator that needs a deliberative approach for norm generation, we have seen necessary the change of some aspects of SIMON's algorithm.

The chapter follows the next structure. We first analyse the norm synthesis of SIMON in 3.1. SIMON was created aiming at avoid conflicts in a traffic domain scenario, so the created norm synthesis approach may not capture the consensus of the majority. And thus, may not be appropriate to synthesise norms in virtual communities. Thereafter, we present DESMON 3.2, which is a deliberative norm synthesis approach based on SIMON, capable of capturing the consensus of the community. For this last reason, we will use DESMON for our purpose of regulating on-line communities. Finally, we conclude with a brief summary of the chapter in Section 3.3.

In the next Chapter 4 we present our novel multi-agent based on-line community simulator, which we connect with the here detailed norm synthesis machine DESMON, in Chapter 5. Finally, all of the norm synthesis machines are proven together with the on-line community simulator in Chapter 6.

3.1 Analysing SIMON's synthesis

As argued in the previous chapter, IRON has some drawbacks in compactness. It requires full evidence to generalise norms while SIMON is able to generalise norms with partial evidence, thanks to its optimistic generalisation approach. So SIMON overcomes it synthesising compact normative systems. However, SIMON and IRON have some characteristics that makes them inappropriate to synthesise

norms in a deliberative manner. After analysing SIMON's norm synthesis we have discover that it suffers from two major problems for our purpose. Specifically:

1. SIMON *hastens to add new norms.* The norm generation mechanism is highly *reactive* to conflicts, rushing to add norms to the normative system. Once created a norm, it is immediately activated, instead of accumulating evidences to assess whether it is really necessary or not.

2. SIMON *is reluctant to discarging norms.* It decides whether to discard a norm based on its effectiveness and necessity performance, which are computed ranges by cumulating its effectiveness and necessity along time. However, a norm's effectiveness and necessity are computed by means of Reinforcement Learning, which accumulates punctual evidences. Therefore, SIMON results in a slow deactivation process that preserves norms even if they show signs of being ineffective or unnecessary.

3.2 DESMON

We now introduce DESMON (*DE*liberative *S*imple *M*inimal *O*n-line *N*orm synthesis), the general approach to norm synthesis aimed at extending SIMON- adapting it to overcome the norm synthesis problems introduced in Section 3.1. DESMON is created to be a *deliberative* approach instead of *reactive* as IRON and SIMON. We call reactive approaches to the norm synthesis mechanisms in which each single conflict triggers the addition of norms. This approaches are intended to synthesise norms in scenarios that require to be highly reactive to conflicts, for instance a traffic domain where car collisions are conflicts. On the other hand, our approach, DESMON, is deliberative because it is intended to perform norm synthesis in scenarios like the on-line community scenario, in which single conflicts (i.e., users' complaints) can be accumulated until there is enough evidence to decide the addition of norms.

DESMON's operation follows the on-line approach of IRON and SIMON, and iteratively executes the same steps of the norm synthesis: (i) norm generation, (ii) norm evaluation and (iii) norm refinement. However, it provides some changes in each stage that are aimed at overcoming the problems found in 3.1. Specifically, it provides:

- A *norm generation* process which does not hastens the addition of the norms to the normative system.

- A novel approach to *norm evaluation* that allows to rapidly identify those norms that should be removed from the normative system.

- A *norm refinement* process that adds to the normative system those norms that are proven to be necessary and discards those norms that underperforms.

In what follows we describe DESMON's information model in Section 3.2.1. Then, we detail each one of its norm synthesis process stages (generation, evaluation and refinement) in Sections 3.2.2.1, 3.2.2.2, and 3.2.2.3 respectively. Finally, in Section 3.2.3 we will introduce the strategy of DESMON.

3.2.1 Information model

We consider a MAS composed of a set of agents $Ag = \{ag_1, \ldots, ag_n\}$, a set of actions $Ac = \{ac_1, \ldots, ac_m\}$ available to the agents, a set S of MAS states, and a set $C \subseteq S$ of undesirable MAS states. Hereafter, we will refer to agents and users interchangeably. An agent describes the MAS it is part of from its own, local point of view, namely its *context*. An agent's context is an expression of a language $\mathcal{L}_{Ag}$, composed of first-order predicates $p(\tau_1, \ldots, \tau_m)$, where p is a predicate symbol and $\tau_1, \ldots, \tau_n$ are terms of language $\mathcal{L}_{Ag}$. It also considers a *state transition function* $\mathbb{T} : S \times Ac^{|Ag|} \to S$ that leads the MAS to a state s' from a state s after the agents perform a set of actions $A \subseteq Ac^{|Ag|}$. For convenience, Ac includes a special action *nil* that stands for not performing any action. Given an agent, and a state transition $\langle s, A, s' \rangle$, function $action : Ag \times S \times S \to Ac$ returns the action the agent performed in the transition. An agent's context $c : Ag \times S \to \mathcal{P}(\mathcal{L}_{Ag})$ is an expression of an agent language $\mathcal{L}_{Ag}$ that describes the perception of an agent ag at a given state s. We assume that an agent's context is a set of first-order predicates of the form $p(\tau_1, \ldots, \tau_m)$, where p is a predicate symbol and $\tau_1, \ldots, \tau_n$ are terms of language $\mathcal{L}_{Ag}$. The set of terms in language $\mathcal{L}_{Ag}$ is denoted by $\mathcal{T}$.

Norms establish prohibitions and obligations to an agent whenever some preconditions are fulfilled. A norm is a pair $\langle \varphi, \theta(ac) \rangle$, where φ is the norm's precondition, $\theta \in \{prh, obl\}$ is a modality, where prh stands for a prohibition, and obl stands for an obligation, and ac is an action prohibited or obligated by the modality. A norm's precondition is a set of first-order predicates of the form $p(\tau_1, \ldots, \tau_m)$, where p is a predicate symbol and $\tau_1, \ldots, \tau_n$ are terms of language $\mathcal{L}_{Ag}$. Then, a norm $\langle \varphi, \theta(ac) \rangle$ *applies to* an agent ag in a state s if $c(ag, s) \models \varphi$, and hence $\theta(ac)$ holds for it.

DESMON's norm synthesis operates over a normative network (NN), which stands for a directed-graph whose nodes stand for synthesised norms and whose edges stand for generalisation relationships between norms. A norm in a network may be "created", "active", "discarded" or "represented". Likewise IRON and SIMON, DESMON computes the normative system as the norms that are "active" in the normative network. Figure 3.1 depicts the life cycle of a norm as implemented by DESMON. It shows the possible states that a norm may have in the normative network, along with the transitions between states. Once DESMON creates a new norm, it sets its state to "created" so that it is not yet included in the normative system. Figure 3.1a, namely the transition labelled with "a" in Figure 3.1, illustrates this operation. Eventually, it may consider the norm's activation (Figure 3.1b), setting its state to "active". Otherwise, it may consider to discard it (Fig-

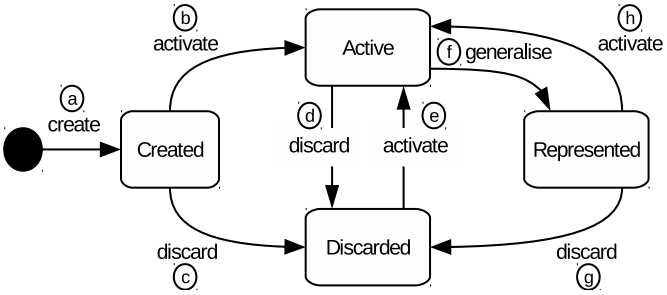

Figure 3.1: A norm's life cycle in DESMON.

ure 3.1c), setting its state to "discarded". At some point, DESMON may consider that an active norm under-performs, discarding it (Figure 3.1d). However, it may activate the norm again if it considers that it is necessary (Figure 3.1e).

Additionally, DESMON performs norm generalisations to compact the normative system. After generalising a norm (Figure 3.1f), DESMON sets its state to "represented", hence removing it from the normative system. However, the norm is not discarded at all, since it is represented by an active norm in the normative system. At some point, the represented norm may under-perform. DESMON then proceeds by first discarding the represented norm (Figure 3.1g), along with all those general norms that represent it. Finally, a represented norm may be activated again (Figure 3.1h) in case that one of the parent norms that represent it is discarded because one of its represented norms under-performs.

3.2.2 Norm synthesis process

Take into account that DESMON is based on SIMON's computational model, which is also based on IRON's. So recall from Section 2.4.2 that IRON uses a norm synthesis process based on three main stages:

1. *Norm generation*, in which it exploits detected conflicts to decide whether to add new norms.

2. *Norm evaluation*, in which it evaluates how norms perform to avoid conflicts.

3. *Norm refinement*, in which it discards norms that do not perform well to avoid conflicts, trying to generalise norms to compact the normative system.

These three norm synthesis stages operate, with the same operators of IRON detailed in 2.4.2.2, over the NN. Moreover, recall that in DESMON a norm may be "created", "active", "discarded" or "represented" (as shown in Figure 3.1).

In the next sections we will explain the changes that DESMON has with respect to IRON and SIMON, and the benefits provided by them.

3.2.2.1 Norm generation

During the norm generation process DESMON, decides whether to create new norms. Briefly, it monitors a MAS, searching for undesirable states (i.e., conflicts). Once it detects a new, non-regulated conflict, it creates a new norm aimed at avoiding the detected conflict in the future.

DESMON perceives the MAS as state transitions of the form $\langle s, A, s' \rangle$, being s' the current MAS state, s the previous MAS state, and A the set of agent actions that lead from s to s'. DESMON assumes that an undesirable state may be avoided if one of the actions that lead to that state was prohibited. Therefore, once DESMON detects an undesirable state s', it first identifies one of the agents ag involved in s'. Then, it retrieves its local context $c(ag, s)$ in the previous state s. Next, it creates a new norm $\langle \varphi, \theta(ac) \rangle$, where φ stands for the agent's context at the previous state, and $\theta(ac)$ is a prohibition of the action the agent performed during the transition from s to s'. Formally, $\varphi = c(ag, s)$ and $\theta(ac) = prh(action(ag, s, s'))$. Then, it adds the created norm to the normative network, and sets its state to "created". In this way, it does not immediately activate the norm. Instead, it decides whether activating or discarding in subsequent synthesis stages.

Function *normGeneration* (Algorithm 3) illustrates DESMON's norm generation. It receives a transition between the previous system state s, and current state s after the application of actions in A, a set of conflicting states C of the MAS, and a normative network NN. If the current system state s' is undesirable (line 2), then it creates a new norm aimed at avoiding s' in the future (line 3). Finally, it adds the norm to the normative network (line 4), and sets its state to "created" (line 5).

Algorithm 3 Function *normGeneration*

```
1: function normGeneration(⟨s, A, s'⟩, C, NN)
2:     if s' ∈ C and not(regulated(s', NN) then
3:         n ← create(⟨s, A, s'⟩, NN);
4:         NN ← add(n, NN);
5:         NN ← setCreated(n, NN);
6:     end if
7:     return NN
8: end function
```

3.2.2.2 Norm evaluation

We now detail how DESMON evaluates norms to assess whether they are good enough to regulate a MAS. It computes a norm's effectiveness and necessity based on the conflicts that arise after agents fulfil or infringe the norm. On the one hand, it computes the *cumulative* effectiveness of a norm as its ratio

of *successful fulfilments*, namely those fulfilments that did not lead to conflicts. On the other hand, it computes its *cumulative* necessity as its ratio of *harmful infringements*, namely those infringements that led to conflicts. We describe below how DESMON gathers and cumulates evidences of a norm's fulfilments and infringements, and how it computes its effectiveness and necessity according to these evidences.

3.2.2.2.1 Gathering norm compliance evidences We first formalise the concepts of norm fulfilment and norm infringement. Given a state transition $\langle s, A, s' \rangle$, and a norm n that applies to an agent ag in s, we say that ag fulfilled n in the transition from s to s' if:

- It performed an action that is obliged by n, or
- It did not perform an action that is prohibited by n.

Definition 10 (Norm fulfilment)**.** Given a state transition $\langle s, A, s' \rangle$, an agent ag, and a norm $\langle \varphi, \theta(ac) \rangle$ applicable to ag in s, we say that ag fulfilled the norm during the transition from s to s' iff (i) $\theta = prh$ and $action(ag, s, s') \neq ac$; or (ii) $\theta = obl$ and $action(ag, s, s') = ac$.

Analogously, we say that the agent infringed the norm if it performed an action that is prohibited by the norm, or did not perform an action that the norm obliges.

Definition 11 (Norm infringement)**.** Given a state transition $\langle s, A, s' \rangle$, an agent ag, and a norm $\langle \varphi, \theta(ac) \rangle$ applicable to ag in s, we say that ag infringed the norm during the transition from s to s' iff (i) $\theta = prh$ and $action(ag, s, s') = ac$; or (ii) $\theta = obl$ and $action(ag, s, s') \neq ac$.

Formally, the fulfilment (or infringement) of a norm n during a transition $\langle s, A, s' \rangle$ is considered as successful iff s' is not an undesirable state, namely if $s' \notin C$. Analogously, a norm fulfilment or infringement is harmful iff $s' \in C$. For each created norm n, DESMON creates a tuple of finite series

$$\langle \mathcal{SF}^n, \mathcal{HF}^n, \mathcal{SI}^n, \mathcal{HI}^n \rangle$$

that accumulate, respectively, its:

- *successful fulfilments* ($\mathcal{SF}^n$), namely fulfilments that did not lead to conflicts.
- *harmful fulfilments* ($\mathcal{HF}^n$), namely fulfilments that lead to conflicts
- *successful infringements* ($\mathcal{SI}^n$), namely infringements that did not lead to conflicts
- *harmful infringements* ($\mathcal{HI}^n$), namely infringements that lead to conflicts.

In particular, $\mathcal{SF}^n$ is a series $\langle sf_1^n, \ldots, sf_m^n \rangle$, where sf_i^n is the number of successful fulfilments of n the i-th time it was fulfilled, and $\mathcal{HF}^n$, $\mathcal{SI}^n$, $\mathcal{HI}^n$ are computed analogously.

3.2.2.2.2 Evaluating a norm's effectiveness and necessity DESMON continuously evaluates a norm in terms of its effectiveness and necessity to avoid conflicts. With this aim, for each created norm n, it creates a pair of series

$$\langle \mathcal{U}^n_{eff}, \mathcal{U}^n_{nec} \rangle$$

that accumulate, respectively, its (i) effectiveness ($\mathcal{U}^{eff}_n$), namely the frequency in avoiding conflicts along time whenever n is fulfilled; and (ii) necessity ($\mathcal{U}^{nec}_n$), namely the arisen conflicts whenever n is infringed. In particular, $\mathcal{U}^n_{eff}$ is a series $\langle eff^n_1, \ldots, eff^n_m \rangle$, where eff^n_i is the ratio of absence of conflicts of n the i-th time it was fulfilled. This is computed according to the following formula:

$$eff^n_i = \frac{sf^n_i}{sf^n_i + hf^n_i} \tag{3.1}$$

where sf^n_i is the number of successful fulfilments the i-th time n was fulfilled, and hf^n_i is the number of harmful fulfilments of n the i-th time it was fulfilled.

Analogously, $\mathcal{U}^n_{nec}$ gathers evidences about a norm's necessity along time. It is a series $\langle nec^n_i, \ldots, nec^n_m \rangle$, where nec^n_m is the ratio of arisen conflicts of n the i-th time it was infringed, and is computed according to the following formula:

$$nec^n_i = \frac{hi^n_i}{hi^n_i + si^n_i} \tag{3.2}$$

where hi^n_i is the number of harmful infringements of n the i-th time it was infringed, and si^n_i is the number of successful infringements of n the i-th time it was infringed.

DESMON exploits norms performances during norm refinement (described in the next Section 3.2.2.3). However, series $\mathcal{U}^n_{eff}, \mathcal{U}^n_{nec}$ may not be appropriate to decide whether to activate or discard a norm. Specifically, both series may have short-term fluctuations, which may lead DESMON to continuously activate and discard the norm. Therefore, as it is the case for data streams, DESMON computes the *cumulative moving average* [Chou, 1969] of series $\mathcal{U}^n_{eff}, \mathcal{U}^n_{nec}$ in order to highlight their long-term trends. It computes cumulative series $\hat{\mathcal{U}}^n_{eff}, \hat{\mathcal{U}}^n_{nec}$. In particular, $\hat{\mathcal{U}}^n_{eff}$ is a series $\langle \hat{eff}^n_1, \ldots, \hat{eff}^n_m \rangle$, where $\hat{eff}^n_i$ represents the cumulative effectiveness of n to the i-th time it has been fulfilled, and is computed as follows:

$$\hat{eff}^n_i = \frac{\sum_{eff^n_j \in \mathcal{U}^n_{eff}, 0 \leq j \leq i} eff^n_j}{i} \tag{3.3}$$

and $\hat{\mathcal{U}}^{nec}_n$ is computed analogously, by means of the following formula:

$$\hat{nec}^n_i = \frac{\sum_{nec^n_j \in \mathcal{U}^n_{nec}, 0 \leq j \leq i} nec^n_j}{i} \tag{3.4}$$

3.2.2.3 Norm refinement

During norm refinement, DESMON decides upon the activation, discard and generalisation of those norms that have been created during norm generation. With this aim, it employs the norms' performance evidences accumulated during norm evaluation. In what follows, we detail how DESMON decides and executes:

1. The **activation** of those norms that are proven to be necessary (in Section 3.2.2.3.1).
2. The **discard** of ineffective or unnecessary norms (in Section 3.2.2.3.2).
3. The **generalisation** of norms in order to compact the normative system, which is the same as DESMON's, explained in Section 2.5.1.3.

3.2.2.3.1 Activating necessary norms: DESMON decides the activation (and subsequent addition to the normative system) of norms based on their cumulative necessity, which it gathers iteratively during norm evaluation. Eventually, it activates a norm n provided that:

- It accumulates enough number of evidences regarding the necessity of the norm:
$$|\hat{\mathcal{U}}^n_{nec}| > evid_{min} \tag{3.5}$$
where $evid_{min}$ stands for the minimum number of evidences required to decide a norm's activation.
- Its cumulative necessity is over a *necessity threshold band* $\langle \alpha^-_{nec}, \alpha^+_{nec} \rangle$ that states the minimum necessity required for a norm to activate it (α^+_{nec}); and the necessity under which a norm should be discarded (α^-_{nec}).
$$\hat{nec}^n_m > \alpha^+_{nec} \tag{3.6}$$
where $\hat{nec}^n_m$ is the last value of series $\hat{\mathcal{U}}^n_{nec}$, and α^+_{nec} is the *upper bound* of the necessity threshold band $\langle \alpha^-_{nec}, \alpha^+_{nec} \rangle$.

Here, the difference between the reactive approaches as IRON and SIMON from DESMON relies on the minimum amount of evidences ($evid_{min}$) they require to activate a norm. In particular, IRON and SIMON were capable of activating a norm from a unique evidence (that is, $evid_{min} = 1$). By contrast, DESMON requires to cumulate a minimum of evidences $evid_{min} > 1$ to decide a norm's activation. Thus, the more minimum evidences DESMON requires, namely the higher $evid_{min}$ is, the more deliberative we consider it to be.

DESMON proceeds to activate a norm by first setting its state to "*active*" in the normative network. As described in Section 3.2.1, DESMON may activate a "*created*" norm (Figure 3.1b), or a "*discarded*" norm (Figure 3.1e), namely a norm that was discarded at some point because it was proven to be whether ineffective or unnecessary. Recall from Section 3.2.1 that, once DESMON discards

a norm, it discards all the general norms it represents, which implies the loss of compactness of the normative system. Therefore, after DESMON activates a discarded norm, it tries to regain the compactness of the normative system again by checking if any of the norm's parents may be re-activated.

Algorithm 4 Function *activateUp*

```
 1: function ACTIVATEUP(n, NN)
 2:     if not isRepresented(n) then
 3:         NN ← activate(n, NN)
 4:         children ← getChildren(n, NN)
 5:         parents ← getParents(n, NN)
 6:         for each child ∈ children do
 7:             if represented(child) then
 8:                 NN ← setRepresented(n, NN)
 9:             end if
10:         end for
11:         for each parent ∈ parents do
12:             if not representDiscardedNorms(parent) then
13:                 activateUp(n, NN)
14:             end if
15:         end for
16:     end if
17:     return NN
18: end function
```

Algorithm 4 illustrates recursive function *activateUp*, which activates a norm along with those norms that represent it. It receives a normative network *NN* and a norm n to activate. If the norm is not already represented by another norm (line 2) it activates it (line 3). Next, it removes from the normative system all its children (since they are now represented by n) by setting their state to "represented" (lines 6–10). Finally, for each parent of n, it checks if it represents any discarded norm (i.e., with state "discarded"). If a parent does no longer represent discarded norms, then it is activated up along the lines of n (lines 11–15). Eventually, the re-activation of its parent norm will imply that n will be set to state "represented".

3.2.2.3.2 Discarding under-performing norms: DESMON discards a norm whenever it gathers enough evidence to consider it as either ineffective or unnecessary. As detailed in Section 3.2.2.2.2, a norm's effectiveness is computed based on to the outcomes of its fulfilments, and its necessity is computed based on the outcomes of its infringements. As described in Section 3.2.2.2.1, those norms in the normative system (i.e., norms "active" and 'represented" in the normative network) are available to the agents, and hence can be fulfilled and infringed. Therefore, active and represented norms can be evaluated in terms of their effectiveness and necessity. By contrast, norms that are not represented in the normative system (i.e., norms that are "created" and "discarded") are not available to the agents, and hence they cannot fulfil them. Therefore, created and discarded norms can be evaluated only in terms of their necessity. DESMON decides to discard a norm as follows:

- If the state of a norm is "*created*", discard it if satisfies both conditions 7 and 8:

– There are enough accumulated evidences regarding its necessity.

$$|\hat{\mathcal{U}}^n_{nec}| > evid_{min} \tag{3.7}$$

where $evid_{min}$ stands for the minimum number of evidences required to decide a norm's discard.

– And its cumulative necessity is under a necessity threshold band.

$$\hat{nec}^n_m < \alpha^-_{nec} \tag{3.8}$$

where $\hat{nec}^n_m$ is the last value of series $\hat{\mathcal{U}}^n_{nec}$, and α^-_{nec} is the lower boundary of the necessity threshold band.

- If the norm is either "*active*" or "*represented*", discard it if, satisfies conditions 9 and 10 and at least one of conditions 11 or 12:

– There are enough accumulated evidences regarding its *effectiveness* and *necessity*.

$$|\hat{\mathcal{U}}^n_{eff}| > evid_{min} \tag{3.9}$$

$$|\hat{\mathcal{U}}^n_{nec}| > evid_{min} \tag{3.10}$$

– And, its *effectiveness* or *necessity* is under respective threshold bands.

$$\hat{eff}^n_m < \alpha^-_{eff} \tag{3.11}$$

$$\hat{nec}^n_m < \alpha^-_{nec} \tag{3.12}$$

where $\hat{eff}^n_m$ is the last value of series $\hat{\mathcal{U}}^n_{eff}$, $\hat{nec}^n_m$ is the last value of the necessity moving average $\hat{\mathcal{U}}^n_{nec}$, α^-_{eff} is the lower bound of an effectiveness threshold band $\langle \alpha^-_{eff}, \alpha^+_{eff} \rangle$, and α^-_{nec} is the lower bound of the necessity threshold band $\langle \alpha^-_{nec}, \alpha^+_{nec} \rangle$.

DESMON proceeds to discard a norm by first setting its state to "*discarded*" in the normative network. Then, it recursively discards up those parent norms that represent it so that it is no longer represented in the normative system.

Algorithm 5 Function *discardUp*

```
1: function DISCARDUP(n, NN)
2:     parents ← getParents(n, NN)
3:     for each parent ∈ parents do
4:         discardUp(n, NN)
5:     end for
6:     if represented(n) then
7:         specialise(n, NN)
8:     end if
9:     return NN
10: end function
```

Algorithm 5 depicts how DESMON discards a norm up. It receives a norm n to discard, and a normative network NN. First, it recursively discards up those

parent norms that represent it (lines 2–5). Finally, if the norm is not represented by any active parent, it discards it, setting its state to "discarded" and activates all the children it represented when it was active. Eventually, the algorithm will discard its under-performing children in recursive executions.

3.2.3 Simon's Norm synthesis strategy

We have so far described how DESMON manages to *generate norms* based on detected conflicts, *evaluate norms* performances, and *refine the normative system* with the same optimistic approach as SIMON (2.5.1.3). Now we are ready to introduce DESMON's strategy, which subsequently executes the different norm synthesis stages.

Algorithm 6 DESMON's general-purpose norm synthesis strategy

```
1:  function DesmonStrategy(evid_min, ⟨s, A, s'⟩, C, NN, T, G_M, G_S)
2:      if s' ∈ C and not regulated(s', NN) then
3:          createdNorms ← normCreation(⟨s, A, s'⟩, NN)
4:          NN ← add(createdNorms, NN)
5:      end if
6:      applicableNorms ← normApplicability(s, NN)
7:      normsPerformances ← normEvaluation(s', applicableNorms)
8:      normsToActivate ← ∅
9:      normsToDiscard ← ∅
10:     normsToGeneralise ← ∅
11:     for each n ∈ createdNorms ∪ applicableNorms do
12:         if shouldBeActivated(n, normPerformances, evid_min) then
13:             normsToActivate ← normsToActivate ∪ {n}
14:             if isCreated(n, NN) then
15:                 normsToGeneralise ← normsToGeneralise ∪ {n}
16:             end if
17:         else if shouldBeDiscarded(n, normPerformances, evid_min) then
18:             normsToDiscard ← normsToDiscard ∪ {n}
19:         end if
20:     end for
21:     for each n ∈ normsToActivate do
22:         NN ← activateUp(n, NN)
23:     end for
24:     for each n ∈ normsToDiscard do
25:         NN ← discardUp(n, NN)
26:     end for
27:     NN ← generaliseUp(normsToGeneralise, NN, T, G_mode, G_step, normsPerformances)
28:     return NN
29: end function
```

Algorithm 6 describes DESMON's norm synthesis strategy, which receives as input: the minimum number of evidences $evid_{min}$ it requires to decide a norm's activation/discard; a transition $\langle s, A, s' \rangle$ with the system states before (s) and after (s') the performance of the agent actions in A; the set $C \subset S$ of the conflicting states of the system; a normative network NN, a taxonomy of terms T, norm generalisation mode G_{mode} (*Shallow* or *Deep*) and the generalisation step G_{step} (these last two).

Initially, the strategy carries out norm generation. It first checks if the current state s' is undesired (i.e., conflictive) and it has not generated norms that can regulate (avoid) it (line 2). In that case, it creates (and adds to the normative

network) a new norm with state "created", which is aimed at avoiding state s' in the future (lines 3–4). Next, it performs norm evaluation. With this aim, it first retrieves those norms that were applicable to the agents in previous state s (line 6). Then, it evaluates how norms applicable in s have performed in avoiding conflicts in s' (line 7).

Afterwards, it executes norm refinement. First, it employs cumulative norms' performances to detect if each applicable norm (1) is neither active nor represented, but it is necessary enough to be activated up as described in Section 3.2.2.3.1 (lines 12–13); (2) can be taken to the generalisation process (lines 14-16); or (3) under-performs and should be discarded up as described in Section3.2.2.3.2 (lines 17–19).

Specifically, DESMON tries to generalise a norm whenever it is going to be activated for the first time, namely it should be activated (line 12) and its current state is *Created* (line 14). Finally, it activates up norms that should be activated (lines 21–23) along the lines of Section 3.2.2.3.1, discards up norms that should be discarded (lines 24–26) as described in Section 3.2.2.3.2, and generalises norms (lines 27) as done in SIMON [Morales et al., 2014a] and explained in Section 2.5.1.3.

3.3 Summary

We have introduced DESMON, the general approach to norm synthesis in a deliberative and participatory manner, based on SIMON, that overcomes it's problems of synthesising normative systems for non-reactive domains by applying some changes in the iteratively executed strategy stages.

First of all, we improved the norm life cycle by adding more states to the life of a norm, instead of being only "active" and "discarded". Secondly, we evaluate norms in term of their cumulative effectiveness and necessity and use these ranges to activate or discard norms. Thirdly, the norm generation step is modified to avoid hastening the addition of norms. So a norm is incubated ("created" state) until after a minimum of evidences it proofs to be necessary to avoid conflicts, hence it is activated. Finally, the discard of ineffective or unnecessary norms it is also novel. We discard a norm if a minimum number of evidences has been gathered and at least its cumulative necessity or effectiveness ranges are below a boundary.

Moreover, thanks to the mentioned changes, DESMON is a deliberative and participatory approach, and hence able to synthesise normative systems according to the general users' opinions. It is capable of synthesising norms on scenarios that require high evidence of conflicts to create a norm (e.g., on-line communities where accumulation of complains about a content are conflicts). This cannot be done with previous norm synthesis algorithms: IRON and SIMON, as they were created to synthesise norms on scenarios that require high reactivity to conflicts

and they create norms from one single conflict (e.g., traffic junction where collisions between cars are conflicts).

In the next Chapter 4 we are going to introduce the on-line community simulator that, afterwards, we attach to the created norm synthesis approach (DESMON) in Chapter 5. Finally, Chapter 6 will empirically evaluate normative systems synthesised with the created *deliberative* approach: DESMON and the two state-of-the-art *reactive* approaches IRON and SIMON, with the aim at synthesizing regulations capturing the majority users' opinion in an on-line community.

Chapter 4

Simulating Social Networks

In this chapter we detail our multi-agent social network simulator. So far we have explained the necessary background for a better comprehension of the work and the improvements we have applied to a state-of-the-art norm synthesis mechanism to synthesise norms for a given domain.

Now, in this chapter, we will describe the created domain, based on MAS simulation, to simulate an on-line community where humans are modelled as agents of the MAS. As we are trying to represent a social process, we have decided to use a MAS simulation because of the benefits we have detailed in Section 2.2.2. Our On-line Communities Simulator, that has been implemented with the aid of *Repast Simphony* [North et al., 2013], allows to perform agent-based time-discrete simulations of users' interactions within an on-line community scenario.

To the best of my knowledge, this is the first multi-agent based social network simulator. This thesis contributes with a novel and state-of-the-art tool that opens a new research branch for this certain domain that is growing rapidly.

In this chapter we outline its features. First of all, in Section 4.1 we are going to explain how our agent-based time-discrete simulation acts over a tick of time. Afterwards, once the work-flow of the simulation is clear, in Section 4.2 we are going to visualise our on-line community simulator. Moreover, as human users are modelled as agents, in Section 4.3 we will explain how we create these human behaviours with a stochastic model. Finally, Section 4.4 details the available extra functionalities the simulator offers and Section 4.5 will offer a summary of the Chapter.

Afterwards in Chapter 5 we are going to connect this simulator with the *norm synthesis mechanisms* explained in Chapter 2 and Chapter 3, namely IRON and DESMON respectively. With this connection we aim at avoiding frictions between users from the novel on-line community simulator, synthesising norms in a deliberative and participatory approach.

4.1 Work-flow of a Simulation

We now present a typical work-flow of a simulation within our on-line community simulator. Recall that our simulations are agent-based and time-discrete, so the work-flow explains the discrete sequence of events the simulator follows in each time period. We focus on detailing each event of the simulation through the graphical representation of a work-flow in Figure 4.1.

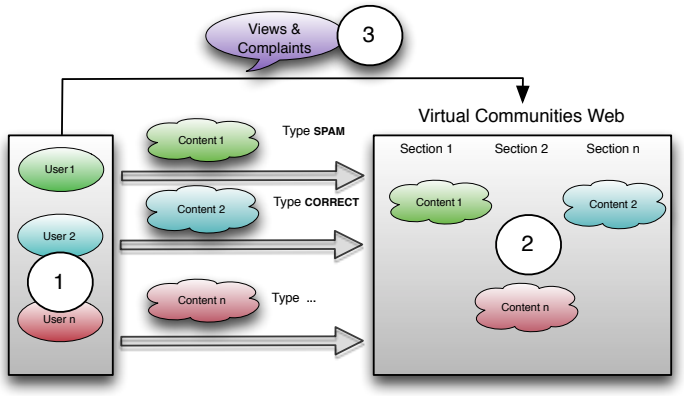

Figure 4.1: Social network simulation work-flow.

On the left hand side, we see a box with users and the box on the right hand side represents the set of contents published by the users in the on-line community. The actions of the users over the community are represented with arrows, in this case the users are able to **upload**, **view** and **complain** about contents of the Virtual Community. Moreover, we specify a taxonomy of contents, which describe the category of the contents the users of the on-line community upload. We classify several content categories as:

1. **Correct**: Content that is not spam, pornographic, violent or insult. Normally, it is the content that do not receive any complain.
2. **Spam**: Content which advertises something unsolicited: an entity, a person or some product. Also the sending of the same message repeatedly is call spam.
3. **Pornographic**: Content that shows sexual acts or nakedness explicitly.
4. **Violent**: Content that encourages violence between community users. It can range from insult, to offensive and provocative commentaries, tending to stir up conflicts.
5. **Insult**: Content that tries to offend and provoke to an individual user of the community.

A typical work-flow is divided in three main time-discrete events, depicted with numbers in Figure 4.1.

1. At step 1, the users that are connected to the social network reason whether to upload a content or not.

2. At step 2, the contents will be uploaded and visible for the rest of users.

3. At step 3, users are allowed to view and complain about contents uploaded to the virtual community. Every time step each user will view a content and decide whether to complain or not. These complaints done by users are going to be the triggers for the mechanism of norm synthesis to create norms (explained in Chapter 5).

4.2 Visualising a simulation

The visualization of the simulator comprises a table of "x" columns and "y" rows. The number of columns represents the contents uploaded to each of the available sections of the community. While the number of rows represents the number of users that are connected to the virtual community. Specifically, in our simulations we represent our virtual community scenario by 33 columns divided into three main groups, each one representing a section of the community, shown in the Figure 4.2. Moreover, each row contains the contents that each user uploads into the community, creating a direct relation between the users and the contents sharing the row. As the column number is finite for each section, in our case 11 contents per user in each section, whenever a user reaches this number of contents the next new content will appear as the last one and the first one will disappear, as a sliding window. These contents will disappear from the visualisation but will stay available for the users of the on-line community to view and complain about. Furthermore, below each content there is a cell that eventually contains an exclamation mark if someone has complained about that content.

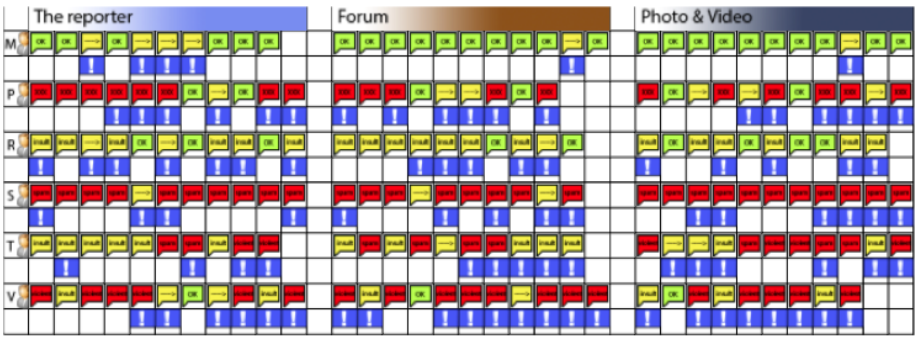

Figure 4.2: Simulator Structure

The purpose of the names of these sections is to maintain an analogy with the section names of the Fanscup website [Lanas and Garzón, 2005]. The Sections shown in the Figure 4.2 are:

- **Forum**: The forum is the most common section within virtual communities. It is a place where people give their opinion about diverse subjects that come out.
- **The Reporter**: The reporter is the section where people upload news from reliable sources. These contents tend to be of interest for the community members and come mainly in the form of plain text.
- **Photo and Video**: The last section is the multimedia one where people upload images and videos. These multimedia contents may vary depending the community users.

Moreover, the uploaded contents, which are classified by the taxonomy explained in Section 4.1, are associated to a colour scale based on the traffic lights colours. Being green a correct content, yellow an inappropriate content and red a noxious, provocative and non desirable content for the community. The association between colours and contents is depicted in Figure 4.3.

Figure 4.3: Categories of contents and their respective colours.

4.3 Users behaviour simulation

Users belonging to an on-line community interact by generating, viewing and complaining about contents. These users are modelled as agents in our MAS simulator. In this Section we present the different user types modelled by our simulator (Subsection 4.3.1) and also the behaviour parameters we tune to make personalized user types (Subsection 4.3.2).

4.3.1 User types

Our simulator allows to classify community users with different behaviours regarding which categories of contents they upload. In particular, a user can be classified in a category out of a set of categories, these customized user types are obtained from an analysis of the users in the *Fanscup* community:

1. **Moderate**: Agent that acts correctly and tries to maintain a harmony in the community. Usually, they are also the ones that complain about inappropriate contents.
2. **Spammer**: Agent that invades the community by uploading mass content, mainly advertisements.
3. **Pornographic**: Agent that publishes pornographic content.
4. **Rude**: Agent that does not insult people personally, but incite with his comments to a fight, creating a *flame war*.
5. **Violent**: Agent that attacks other users with insults and offensive commentaries.

These user types can be combined between each other, creating mixed-behaviour agents, or stay separately creating single-behaviour agents. For instance, a single behaviour agent may be composed only by moderate type so will only upload correct content and complain about all the bad or annoying contents of the community. On the other hand, as example mixed behaviour agent we can configure it with 50% moderate and 50% spammer types, so will upload spam and correct content. For the purpose of customising user types we created a population design tool, which is explained in Section 4.3.2.

4.3.2 User behaviour

Within an on-line community we may find users with different preferences and behaviours. With the aim of modelling different types of users, our simulator incorporates a population design tool, which is shown in Figure 4.4. It allows the designer to define users' preferences and specify a stochastic model of their behaviours with regard to the contents they upload, view and complain about. The top of Figure 4.4 shows the current population that is being specified (depicted with label "Standard"). The bottom of the Figure shows the composition of the population which is being designed (depicted with label "Current population"). In particular, it shows a population of five different agents, one of each of the user types presented in the previous Section 4.3.1.

For a given population, our design tool allows to create and define the profile of each user type (a moderate agent in the picture). A user profile describes how often a user uploads, views and complains about contents.

Moreover, it also allows to specify the category of contents the user will choose to upload, view and complain about. As Figure 4.4 shows, a user profile contains three action-profiles: the *upload profile*, the *view profile*, and the *complaint profile*. Next, we describe each action-profile.

1. **The upload profile** describes: (i) the *upload frequency* of the user, namely the probability of the user of uploading contents at each tick; and

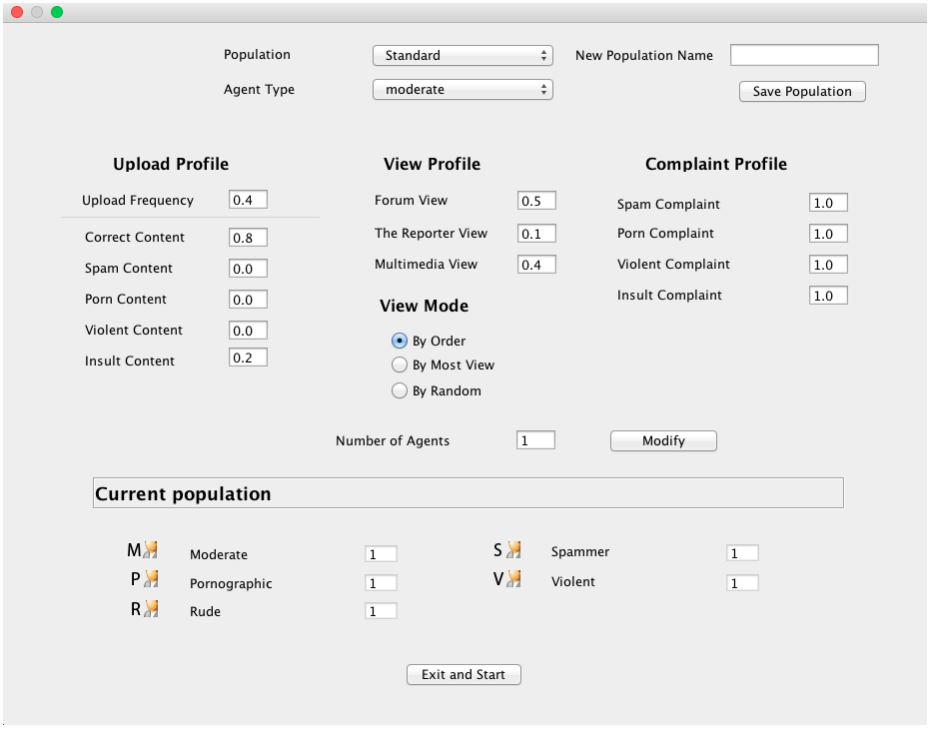

Figure 4.4: Windows to choose the behaviour of the population

(ii) different *upload probabilities* for each content category. Thus, whenever a user uploads a content, the category of this content (correct, spam, porn, insult or violent) depends on some probability. Notice that all the probabilities assigned to different types of contents must sum up 1. As an example, the left part of Figure 4.4 shows the upload profile of a moderate user. It describes a user that uploads contents at each tick with probability 0.4. Every time she uploads contents, she has probability 0.8 of uploading correct contents, while she uploads insults with probability 0.2. So we can deduce from the uploading profile, that this agent is a mixed-behaviour agent.

2. **The view profile** defines users' preferences in terms of the probability of viewing contents from each Section of the community. The central part of Figure 4.4 shows a user's view profile. Likewise the upload profile, all the probabilities assigned to different Sections must sum up 1. Moreover, the view profile also considers the *view mode*, which describes three different ways to choose contents to view:

(a) *By order*. This method chooses the most recently uploaded contents in the community. Specifically, this method works as follows. First, it sorts

all uploaded contents by the time they were uploaded. Second, it assigns a probability to each content according to a gamma distribution[1]. Third, it randomly chooses the next content to view, according to the gamma distribution.

(b) *Most viewed.* It chooses the contents with a larger number of visits. Specifically, it sorts contents by number of views, assigning to each content a probability according to a gamma distribution.

(c) *Randomly.* This method chooses contents according to a uniform random distribution.

3. **The complaint profile** defines the probability of a user of complaining about each type of visited content. Notice that, unlike previous probabilities, these values are independent and, hence, their addition is not required to be 1. The right hand side of Figure 4.4 depicts the complaint profile of a moderate user that complains about all type of conflicting contents with probability one.

Notice that, even though two users may belong to the same category, their user profiles may be different because of the percentages. As a convention, whenever a user is configured to upload a single type of content (with probability 1 and 0 in the rest), we say that this user is *single-behaviour* user. By contrast, whenever a user is configured to upload more than one type of content , we say that it is a *mixed-behaviour* user. Figure 4.4 depicts a *mixed-behaviour* user's profile, since it has 80% probability of uploading correct content and 20% of uploading insults.

4.4 Functionalities of the simulator

Our network simulator offers several functionalities:

- **Scenario generation:** It is able to define, at design time, the population of agents that will be part of the simulation and their action-profiles.
- **Repeatability:** It allows to repeat an execution exactly as a previous one.
- **User behaviour simulation:** It is able to simulate the real community users' behaviours in a discrete MAS at runtime from a stochastic model.
- **Discrete multi-agent simulation:** It simulates an on-line community by means of discrete-events on time, executing each time actions as: upload, view and complain.
- **Graphical facilities:** It is able to execute the on-line community simulation in two modes:

[1] A graphical representation of this distribution can be found at `http://en.wikipedia.org/wiki/File:Gamma_distribution_pdf.svg`, in our case we have used $\Gamma(1, 2)$

– With GUI: Watching the presented simulator visualisation in 4.2.
– In Batch: We have created two batch modes to execute the simulator without the graphical mode in order to run experiments. On the one hand, in an ordinary simulation in a single computer. On the other hand, in a computer cluster where the processes can be separated in different threads paralleling the experiments with the aim of gaining time.

4.5 Summary

To sum up, we have introduce a novel and state-of-the-art on-line community simulator that performs agent-based time-discrete simulations of users interactions within the community. This simulator allows MAS simulations, where real human users are modelled as agents. Each user has its own action-profile, which we are able to modify in design time and this stochastic model is the one that simulates human behaviours in runtime.

Recall from Section 2.4 the computational model of IRON, which is depicted in Figure 4.5.

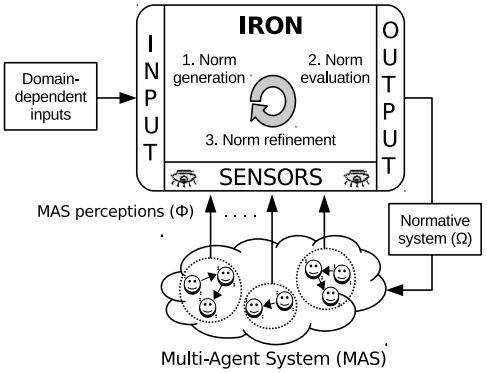

Figure 4.5: Computational model of IRON.

In this Chapter we have introduce the MAS that the norm synthesis mechanisms will use to generate the norms. In the next Chapter 5 we will explain the different domain dependant inputs that the norm synthesis approaches need to operate over our novel MAS simulator and pursue our goal of synthesising deliberative and participatory normative systems.

Chapter 5

Synthesising norms in Social Networks

The norm synthesis is a task we are going to make in an on-line manner as commented on previous chapters. We will try to pursue our goals of synthesising deliberative and participatory normative systems using our novel and state-of-the-art social network simulator, introduced in Chapter 4, integrated with the different norm synthesis algorithms: IRON (Chapter 2) and DESMON (Chapter 3). To integrate a norm synthesis algorithm with the social network simulator we will have to define some domain dependant inputs as, explained in Section 2.4.3:

1. A function to perceive observations (Section 5.2.1).

2. A grammar to synthesise norms in the domain agents language (5.2.2).

3. A function to detect conflicts (Section 5.2.3).

4. A function to detect the applicability of the norms (Section 5.2.4).

5. A function to evaluate norms (Section 5.2.5).

This chapter follows the next structure. First of all, Section 5.1 explains the general architecture resulting from the integration of the multi-agent social network simulator with a norm synthesis algorithm. Afterwards, Section 5.2 shows the domain dependant inputs, mentioned above, necessary to synthesise normative systems for our social network domain.

Finally, once that everything is connected and working we are going to evaluate it empirically and draw some conclusions of the experiments in Chapter 6.

5.1 Social network legislation system: a general architecture

The general architecture of the system to legislate a social network, as shown in Figure 5.1, is composed of the social network simulator described in Chapter 4 together with a *norm synthesis machine* (*NSM* hereafter).

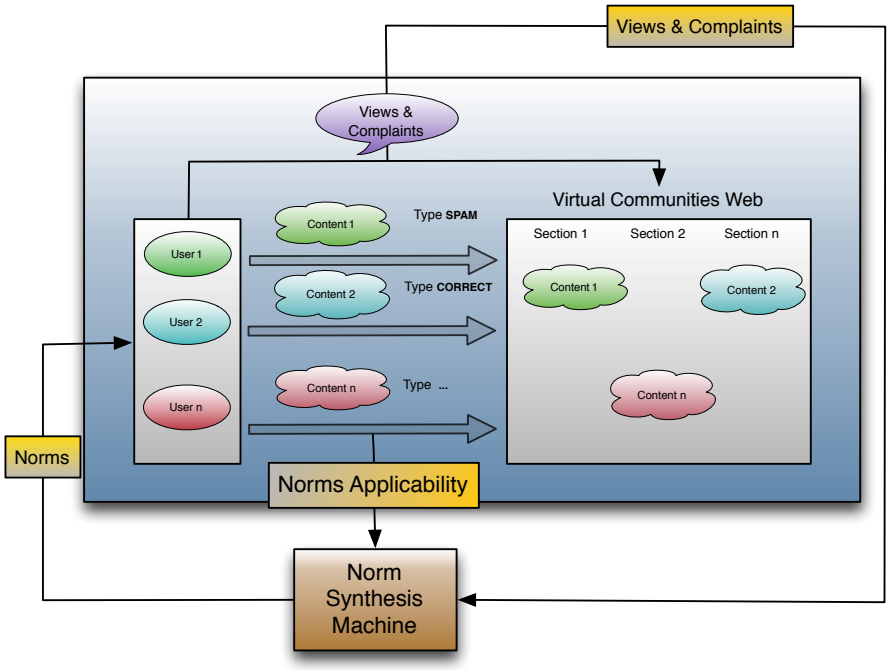

Figure 5.1: social network legislation system: a general architecture.

As we have observed in the work-flow presented in Section 4.1, the users of a simulated social network can *upload*, *view* and *complaint* about contents. Once the simulation starts each tick the NSM will spread the current legislation (normative system) to the users, so that they can reason about the contents they are going to *upload*. In case a user decides to comply with the applicable norms, the content will be deleted and the user will not upload anything. On the contrary, even knowing that some norm applies to the content, if the user decides not to follow the norm, the content will be uploaded but tagged as a violation and the corresponding norm.

Regarding *views* and *complaints*, each user has its own behaviour, as mentioned in Section 4.3, which will model the users' action-profiles. Every user will view a content at every time tick, but only the ones that have a probability of com-

plaining about certain contents will complain.

When a complain is issued we will consider the content as controversial and it will be sent to the NSM to be treated. If there was no norm preventing the conflict, the NSM will create one. On the other hand, if there was already a norm regulating the friction this will increase the norms necessity, as a harmful fulfilment. Nevertheless, if there was already a norm regulating the friction but no one complaints about the content this will decrease the necessity of the norm as the norm existence is not helpful, namely a successful infringement.

5.2 Integrating NSM's with a social network simulator

At this point, we have described our virtual communities simulator, as well as the norm synthesis approaches presented in Sections 2.4 and Chapter 3, namely IRON and DESMON respectively. In this Section we describe how to configure these machines to synthesise norms for the agents in our on-line communities simulator.

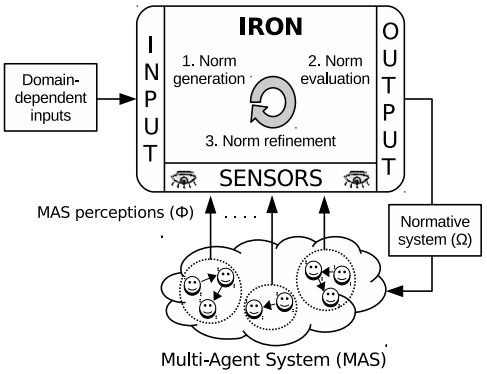

Figure 5.2: Norm synthesis architecture: components and inputs.

Recall the architecture of the norm synthesis approaches shown in Figure 5.2. Even though IRON and DESMON are abstract mechanisms (i.e., scenario independent) during the norm synthesis process they employ some elements that are scenario dependant. First of all, the perceived observations/views are scenario dependant since each scenario will be composed of different elements and may be described in different ways (e.g., a traffic junction where the collisions between cars are the conflicts, instead of our social community where the inappropriate contents are the conflicts). Secondly, in order to execute the main three stages of the strategy, that is, norm *generation*, *evaluation* and *refinement*, the norm synthesis approach requires some scenario dependant inputs. These inputs will

allow a communication between the scenario and the NSM to generate normative systems. In the next sections we explain how to integrate our simulator and a NSM.

5.2.1 Perceiving On-line Communities

Recall from Chapter 4 that our on-line community scenario is divided into three different sections where the users upload, view and complain about contents. In order to perceive the scenario, at each time step the simulator generates *observations* for the NSM.

An *observation* is composed of three lists, one for each section. Each element in a list contains a content that has been uploaded, viewed, or complained about, as well as the corresponding action, and the identifier of the user who performed the action. Moreover, each content incorporates the information about its category and its owner, namely the user that uploaded the content. Thus, an *observation* is composed of nine lists (three lists for each one of the three sections) that describe users' uploads, views and complaints in the community during the current time step.

As an example, consider that during the current time step user u_1 has uploaded a `correct` content in Section `Forum`, and user u_2 has viewed and complained about a `spam` content (whose owner is user u_1) in Section `Multimedia`. The corresponding observation can be thus described as shown in Table 5.1. In Section `Forum`, it shows user u_1 that has performed the action `upload` over a content of category `correct`. In Section `Multimedia`, the table shows that user u_2 has performed two actions: a `view` and a `complain` over a content with category `spam`, whose owner is user u_1.

The Reporter	**Forum**	**Multimedia**
	content: $id(1)$ • action: $u_1 \rightarrow$ `upload` • category: `correct` • owner: u_1	content: $id(2)$ • action$_1$: $u_2 \rightarrow$ `view` • action$_2$: $u_2 \rightarrow$ `complain` • category: `spam` • owner: u_1

Table 5.1: An example of a NSM's *observation*.

5.2.2 Norms for On-line Communities

We must provide an NSM with a specific **grammar** to synthesise norms for the on-line community scenario. Following the grammar specification in Section 2.4.3.1, we instantiate our grammar as follows.

On the one hand, a norm precondition has three unary predicates with predicate

symbols in $P = \{\texttt{user}, \texttt{section}, \texttt{contentType}\}$. The term for predicate *user* is one out of the set of user identifiers U (e.g., u_i is the *i-th* user in U). The term for predicate *section* is one out of the set of Section names $S = \{\texttt{Forum}, \texttt{Multimedia}, \texttt{The Reporter}\}$. The term for predicate *contentType* is one out of the set of content categories $C_{type} = \{\texttt{correct}, \texttt{spam}, \texttt{porn}, \texttt{insult}, \texttt{violent}\}$.

On the other hand, we consider each norm's consequence only specifies a prohibition to perform the action `upload` of the content in the context specified by the norm precondition (thus, the deontic operator is $\theta = phr$). Therefore, norms establish prohibitions for users to upload certain types of contents into some community sections.

$$n : \langle (user(u_1), \quad section(\texttt{Multimedia}), \quad contentType(\texttt{spam})), \quad prh(\texttt{upload}) \rangle$$
$$n' : \langle (user(u_3), \quad section(\texttt{Forum}), \quad contentType(\texttt{violent})), \quad prh(\texttt{upload}) \rangle$$

Figure 5.3: Examples of norms for the on-line community scenario.

Figure 5.3 shows two examples of norms n and n' automatically synthesised by an NSM by means of this grammar. Norm n prohibits user u_1 to upload spam contents into the `Multimedia` section, while norm n' prohibits user u_3 to upload violent contents into the `Forum` section. Recall that an NSM generates new norms whenever it detects new conflicts in the scenario. In what follows we explain how to detect conflicts in our on-line community scenario.

5.2.3 Conflict detection in On-line communities

A function that identifies conflicts in an *observation* perceived by the NSM. This function will be composed of a view that contains what did users during the last time tick and goal to achieve, in this case to avoid users complaints. It is designed to assess if the contents within this given *observation* is conflictive or not.

Particularly, in our on-line community scenario, conflicts can be identified based on the complaints users report about contents. Consider that a user uploads some content, and then that content receives a complaint. In that case, the action of uploading the content is considered as a conflict. Recall that an NSM's norm synthesis is conflict-driven, since it generates, evaluates and refines norms based on conflicts. Therefore, thanks to our definition of conflict, the whole norm synthesis process becomes a participatory process which is guided by users' complaints.

For each given content, the NSM uses function $Conflictive(content)$ to check if it is conflictive. This function looks if someone has complained about the given content in the its life period.

$$Conflictive(content) = \begin{cases} true & \textbf{if } (complaints(content) \geq 1) \\ false & \textbf{otherwise} \end{cases}$$

As an example, we will consider that user u_1 uploads a spam content in Section `Multimedia`. Consider now that another user views the content and complains about it, reporting a complaint of type spam. Once a user has complained about the content, then uploading spam contents in Section `Multimedia` is considered as a new conflict. Therefore, the users of the on-line community trigger, by means of their complaints, the generation of norm n (depicted in Figure 5.3), which prohibits user u_1 to upload spam contents into Section `Multimedia`.

But to generate a norm, an NSM needs to know the origin of the conflictive content. In the case of our scenario, recording the origin of a content is very expensive in computational costs, because it would require an array of time since its creation. For this purpose we create a synthetic view from a conflictive content that is created when a conflict arises. This conflictive content contains all the necessary information (content type, owner id, action) to create the synthetic view. In Figures 5.4 and 5.5 we can see the difference between the realistic view, made of an incremental backup saving, and our synthetic view approach.

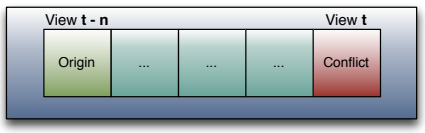

Figure 5.4: Incremental backup.

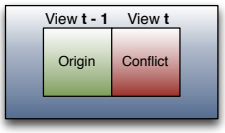

Figure 5.5: Synthetic view.

5.2.4 Detecting norm applicabilities

Our implementation of the applicability function required by an NSM works as follows. Given an *observation*, it performs the following steps:

i) It *generates a list* with the identifiers of the users that have uploaded new contents into the community during the current time step.

ii) It *computes the individual context* of each user that has uploaded contents.

iii) It *retrieves the norms* that apply to each agent's individual context.

As an example, consider that the current normative system contains the norms in Figure 5.3, namely $\Omega = \{n, n'\}$. Consider now that an NSM perceives that user u_1 uploads a spam content in Section `Multimedia`. First, it builds a list with the identifier of user u_1, since it has recently uploaded contents. Second, it computes u_1's individual context in the same format than the precondition

of a norm as specified by the grammar. Thus, u_1's individual context can be compared with the preconditions of norms in order to detect what norms apply to the user. For instance, say that u_1's individual context is:

$$(user(u_1), section(\texttt{Multimedia}), contentCatg(\texttt{spam}))$$

Third, the applicability function retrieves the norms within the current normative system that apply to this specific individual context. Notice that the individual context of u_1 is equal to the precondition of norm n. Therefore, norm n applies to user u_1, and hence the prohibition to upload spam contents to Section `Multimedia` holds for her.

5.2.5 Evaluating norms

Recall that an NSM provides default functions to evaluate norms in terms of their `effectiveness` and `necessity`. Within our on-line community scenario, norms cannot be evaluated in terms of their effectiveness since norm applicability is not observable. As an explanation, recall that (i) norms are aimed at prohibiting users to upload conflicting contents, and (ii) the effectiveness of a norm is computed from the outcomes of its *fulfilments*. Whenever a user fulfils a norm, then it does not upload conflicting contents and conflicts do not arise. As a consequence, the NSM cannot detect whether the absence of conflicts is because either the user fulfilled a norm or because she did not have the intention to upload conflicting contents.

By contrast, norms can be evaluated in terms of their necessity. Whenever a user uploads conflicting contents, then conflicts arise and the NSM can detect that the user has infringed a norm. Specifically, we evaluate norms' necessity by means of the NSM's default necessity function μ_{nec}. Specifically, the necessity of a norm is computed as its ratio of harmful infringements. On the one hand, norm infringements that lead to conflicts make the NSM to evaluate the norm as necessary. On the other hand, norm infringements that lead to non-conflictive situations make the NSM to evaluate the norm as unnecessary.

5.3 Summary

In this chapter we have described the necessary steps to integrate the multi-agent on-line community simulator with an NSM. Over-viewing its architecture and the domain dependant inputs needed to create the regulations. Without these domain inputs an NSM will not be able to synthesise norms for the given domain as it will not know: (i) what is happening in the domain each tick (perceptions), (ii) what language do the agents of the scenario understand (grammar), (iii) what is a conflict, (iv) if a norm applies to an agent, and (v) how to evaluate norms necessity.

In the next Chapter 6 we will empirically evaluate the on-line community simulator attached with different normative system machines.

Chapter 6

Empirical analysis and results

We now perform an empirical evaluation and comparison of IRON's, SIMON's and DESMON's norm synthesis in the on-line community scenario described in Chapter 4. We first detail in Section 6.1 the empirical settings of our experiments. Thereafter, in Section 6.2 we empirically show the differences between the approaches and their results for the on-line community scenario. More specifically, we first perform a macro analysis of the convergence, illustrating *when* they converge, and *what* types of normative systems they synthesise. Second, we perform a micro analysis of their norm synthesis process. Our purpose is to shed light on *how* IRON, SIMON and DESMON, manage to synthesise normative systems, and what type of data structures (normative networks) they synthesise with this aim.

6.1 Empirical Settings

In this section we describe the empirical settings used in the empirical evaluation, we detail settings for: the on-line community, the simulator and the different norm synthesis mechanisms.

6.1.1 On-line community settings

With the aim of comparing both IRON and DESMON, we employ a discrete agent-based simulator that implements the scenario described in 4. In particular, we aim at regulating a simulated on-line community of 10,000 users. However, as described in [Nielsen, 2006], within an on-line community only the 1% of its users are *heavy contributors*, while the remaining 99% are *intermittent contributors* and *lurkers*, namely users that barely contribute to the community. For the sake of simplicity, we focus on a 10,000-user community by means of a population of 100 heavy contributors.

At each time step (i.e, tick) each user: (i) views one content with probability 1; and (ii) uploads one content with probability 0.05. Notice that the probability to upload contents is much lower than the probability to view contents. With this setting, we aim at simulating users' behaviour in real on-line communities, in which users view contents more often than they upload.

At each tick, each user decides whether to fulfil or infringe those norms that apply to it according to a *norm infringement rate*, which is fixed to *low* value (0.3) at the beginning of each simulation and is the same for all users. A user behaves within the community according to its *profile*, which describes *when* and *how* it uploads, views, and complains about contents. Specifically, a user's profile is composed of three sub-profiles (Section 4.3.2). As an example, Table 6.1 depicts the profile of what we call a *moderate* user. That is, a user that only uploads *correct* contents, and complains about all the inappropriate contents it views. More specifically, Table 6.1 describes a user with probability 1 to upload *correct* contents, and probability 0 to upload inappropriate contents (i.e., $spam, porn, violent, insult$). As described in the complain profile, it complains about each type of inappropriate content with probability 1, and complaints about *correct* contents with probability 0. Finally, the view profile defines the different probabilities of the user to view contents from each section.

Upload Profile	
Type	P
Correct:	1
Spam:	0
Porn:	0
Violent:	0
Insult:	0
TOTAL:	1
Upload Frequency	
Frequency:	1

View Profile	
Section	P
Forum:	0.34
The Reporter:	0.33
Multimedia:	0.33
TOTAL	1
View Mode	
By Order	•
Most Viewed	○
Random	○

Complain Profile	
Type	P
Correct:	0
Spam:	1
Porn:	1
Violent:	1
Insult:	1

Table 6.1: A user's profile

We consider three different populations composed of *moderates* and *spammers* in different proportions. On the one hand, moderates behave according to the profile depicted in Table 6.1. On the other hand, spammers upload *spam* contents with probability 1, and never complain about *spam*. We then consider the following populations:

1. A population with a *majority* of moderate users, composed of 70% moderates and 30% spammers (hereafter abbreviated as 30M-70S).

2. A *balanced* population of 50% moderates and 50% spammers (50M-50S).

3. A population with a *minority* of moderate users, composed of 30% moderates and 70% spammers (70M-30S).

6.1.2 Simulator settings

Each section of the community is an sliding window of limited capacity, in our particular case 1000 different contents. Thus, whenever a section is full and a new content arrives, the oldest content is removed in order to place the newly arriving content. Each simulation finishes when it reaches 5,000 ticks. We consider a *"warm-up"* period of 500 ticks for all simulations that allows a simulation to reach normal conditions in on-line communities, where the users enter in the community and there already exist contents to view and complain about. Thus, from tick 0 to 500, users only upload contents, and from tick 500 onwards, users upload, view, and complain about contents. We consider that a simulation has converged whenever the normative system remains unchanged during a 1000-tick period.

6.1.3 Norm synthesis settings

We recall from Chapter 3 that DESMON is a deliberative norm synthesis approach while IRON (2.4) and SIMON (2.5) are reactive approaches. In this empirical evaluation we are going to demonstrate it empirically. We configure IRON's, SIMON's, and DESMON's parameters as depicted in Table 6.2. For each new norm n, IRON and SIMON sets its initial effectiveness and necessity to 0.5, namely $eff_1^n = nec_1^n = 0.5$. By contrast, DESMON initialise a norm with 0 effectiveness and necessity, namely $eff_1^n = nec_1^n = 0$. Thus, it avoids the activation of the norm until it accumulates at least 50 evidences ($evid_{min} = 50$) to assess its activation (or discard). We set IRON's generalisation threshold to 0 ($\alpha_{gen} = 0$) so that it generalises any norm that performs well enough to be active. Analogously, SIMON and DESMON generalise any active norm since they do not take α_{gen} into account. We configure SIMON and DESMON to perform *deep* generalisations ($\mathsf{G}_M = Deep$) with a generalisation step $\mathsf{G}_S = (1)$, since SIMON obtained its best results with this configuration in [Morales et al., 2014a]. There, SIMON synthesised norms for avoiding collisions between travelling cars in a traffic junction scenario.

As described in Section 3.2.2.2.2, norms in this on-line community scenario can be evaluated only in terms of their necessity. Thus, IRON and SIMON deactivates a norm whenever its necessity performance range is under a necessity deactivation threshold α_{spec}^{nec}, which stands for the minimum necessity required for a norm to remain active. Analogously, DESMON activates a norm whenever its cumulative necessity is over a necessity threshold band $\langle \alpha_{nec}^{-}, \alpha_{nec}^{+} \rangle$, and discards a norm whenever it is under the threshold band. We compute the upper bound of the threshold band as $\alpha_{spec}^{nec} + \varepsilon_{nec}$, and its lower bound as $\alpha_{spec}^{nec} - \varepsilon_{nec}$, where $\varepsilon_{nec} = (0.05)$. Therefore, α_{spec}^{nec} directly affects to the preservation and discard of norms, and hence to the convergence to a normative system. We then aim at analysing norm synthesis for different low, medium and high deactivation thresholds, namely $\alpha_{spec}^{nec} \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$.

Parameter	Description	IRON	SIMON	DESMON
eff_1^n, nec_1^n	Default norm effectiveness/necessity	0.5	0.5	0
$evid_{min}$	Minimum number of evidences to assess a norm's activation and discard	—	—	50
α_{gen}	Norm generalisation threshold	0	—	—
G_M	Norm generalisation mode	—	*Deep*	*Deep*
G_S	Norm generalisation step	—	1	1
α_{spec}^{nec}	Norm specialisation threshold	(0.1, 0.3, 0.5, 0.7, 0.9)		
ε_{nec}	Norm necessity threshold bound	—	—	0.05

Table 6.2: Norm synthesis mechanism parameters summary.

6.2 Empirical Results

Our first comparison mainly focuses on when IRON, SIMON and DESMON manage to converge, and what type of normative systems they synthesise. With this aim, we perform 10 different simulations for each population described in 6.1, and for each specialisation threshold (α_{spec}^{nec}) described in Table 6.2.

6.2.1 IRON's macro analysis

Table 6.3 depicts averaged results of 10 different simulations. Each cell depicts, for each population and specialisation threshold: either (i) the amount of norms that IRON converged to; or (ii) symbol "X" if it was not able to converge to a normative system. As we can observe, for a population of 30 moderates and 70 spammers (30M-70S) IRON is able to converge to a stable normative system of 70 norms whenever the specialisation threshold is low ($\alpha_{spec}^{nec} < 0.3$). By contrast, it does not converge for medium and high thresholds ($\alpha_{spec}^{nec} \geq 0.3$). Along the same lines, for population 50M-50S, it converges to 50 norms only for low and medium specialisation thresholds ($\alpha_{spec}^{nec} < 0.5$), and converges to 70 norms for population 70M-30S for low, medium and high thresholds ($\alpha_{spec}^{nec} < 0.7$). In particular, IRON generates a norm for *each spammer* to prevent it from uploading spam in any section. For instance, for a population of 70 spammers it generates 70 norms similar to the following norm

$$\langle \{user(u_1), section(anySection), cntType(spam)\}, prh(upload) \rangle$$

which prohibits a specific user u_1, that is acting as a spammer, to upload spam to any section. We note that IRON is capable of generalising norms for their *section* predicate, but it is not capable of synthesising a general norm that prohibits *all* the spammers to upload spam to any section.

We also note that moderates are the only users who complain about inappropriate contents, and hence can trigger the creation of norms. Therefore, the proportion of moderates in a population represents its capability to complain and to synthesise norms. We then refer to the proportion of moderates in a population as its *complain power* (C_{Pw}). As a general rule, we observe that IRON

	Specialisation threshold α^{nec}_{spec}				
Population	0.1	0.3	0.5	0.7	0.9
30M-70S	70	X	X	X	X
50M-50S	50	50	X	X	X
70M-30S	30	30	30	X	X

Table 6.3: Number of norms that IRON converged to.

	Specialisation threshold α^{nec}_{spec}				
Population	0.1	0.3	0.5	0.7	0.9
30M-70S	1	X	X	X	X
50M-50S	1	1	X	X	X
70M-30S	1	1	1	X	X

Table 6.4: Number of norms that SIMON converged to.

	Specialisation threshold α^{nec}_{spec}				
Population	0.1	0.3	0.5	0.7	0.9
30M-70S	1	X	0	0	0
50M-50S	1	1	X	0	0
70M-30S	1	1	1	X	0

Table 6.5: Number of norms that DESMON converged to.

converges to a stable set of norms whenever the *complain tpower* of the population *is above* the specialisation threshold, namely when $\mathsf{C}_{Pw} > \alpha^{nec}_{spec}$. As an example, in population 30M-70S, 30% of the users consider spam as inappropriate, and hence complain about it. As a consequence, norms that prohibit spam have a necessity around 30%, and hence they are preserved only whenever the specialisation threshold is $\alpha^{nec}_{spec} < 0.3$. This is so because the complaint power of a population is directly related with the necessity of synthesised norms. That is, the more users complain about inappropriate contents, the more necessary the norms that prohibit to upload those contents. Notice then that the specialisation threshold α^{nec}_{spec} acts as a *consensus degree* that establishes the minimum amount of users that must agree about the necessity of creating (and preserving) norms.

Additionally, if the *complain power* of the population *is equal to* the specialisation threshold (that is, $\mathsf{C}_{Pw} = \alpha^{nec}_{spec}$), IRON cannot converge, since there is no agreement to include or discard norms. As explained above, in population 30M-70S norms to prohibit spam are around 30% necessary. In fact, their necessity fluctuates around 0.3, continuously going above and below the specialisation threshold, and being repeatedly deactivated and re-activated. In other words, IRON continuously activates and deactivates norms, being unable to converge to a normative system.

Finally, when the *complaint power* of the population *is under* the specialisation threshold (that is, $\mathsf{C}_{Pw} < \alpha^{nec}_{spec}$), while it should converge to an empty normative

system. That is, since the complain power is under the specialisation threshold, the necessity of synthesised norms will be under the threshold as well, and hence they should be eventually deactivated and remain inactive. However, recall from Chapter 2 that IRON is highly reactive to conflicts. It does not cumulate complaints to decide a norm's activation, but it activates a norm based on one single user complaint. As a consequence, punctual complaints trigger the re-activation of discarded norms, even though, in general, there are no enough complaints to consider their re-activation.

In Table 6.6 we can observe the summary of combinations between the *complaint power* and the *consensus degree* and the results obtained with the norm synthesis mechanism of IRON.

Case	Convergence
$\mathsf{C}_{Pw} > \alpha^{nec}_{spec}$	Multiple norms
$\mathsf{C}_{Pw} = \alpha^{nec}_{spec}$	No
$\mathsf{C}_{Pw} < \alpha^{nec}_{spec}$	No

Table 6.6: Summary of IRON's macro analysis.

6.2.2 SIMON's macro analysis

Table 6.4 depicts SIMON's results. Likewise IRON, it is able to converge to a normative system whenever the complain power of a given population is over the consensus degree ($\mathsf{C}_{Pw} > \alpha^{nec}_{spec}$). Moreover, it is not able to converge whenever the complain power is equal or under the consensus degree ($\mathsf{C}_{Pw} \leq \alpha^{nec}_{spec}$). However, when SIMON converges, it is capable to synthesise a normative system that significantly outperforms IRON's normative systems. Specifically, SIMON can synthesise a single norm that generalises both the *user* and the *section* predicates:

$$\langle \{user(anyUser), section(anySection), cntType(spam)\}, prh(upload) \rangle$$

and hence prohibits *any* user to upload spam to *any* section. Here SIMON benefits from its optimistic approach to norm generalisation, which allows it to perform further generalisations than IRON, hence outperforming it in terms of compactness. In Table 6.7 we can observe the summary of combinations between the *complaint power* and the *consensus degree* and the results obtained with the norm synthesis mechanism of SIMON.

Case	Convergence
$\mathsf{C}_{Pw} > \alpha^{nec}_{spec}$	1 norm
$\mathsf{C}_{Pw} = \alpha^{nec}_{spec}$	No
$\mathsf{C}_{Pw} < \alpha^{nec}_{spec}$	No

Table 6.7: Summary of SIMON's macro analysis.

6.2.3 DESMON's macro analysis

Finally, Table 6.5 shows DESMON's results. Likewise SIMON, DESMON (i) converges to a compact normative system of one general norm whenever the complain power of the population is over the consensus degree ($\mathsf{C}_{Pw} > \alpha^{nec}_{spec}$); and (ii) is not capable to converge whenever the complain power is equal to the consensus degree ($\mathsf{C}_{Pw} = \alpha^{nec}_{spec}$). However, when the complain power is under the consensus degree ($\mathsf{C}_{Pw} < \alpha^{nec}_{spec}$), DESMON converges to an empty normative system that does not include norms to prohibit spam. This is so because DESMON is non-reactive to conflicts (i.e., users' complaints). Unlike IRON and SIMON, it accumulates users' complaints until it has enough evidences to consider the activation of norms. This deliberative approach to norm generation allows DESMON to implement our desired participatory regulation mechanism. It regulates the on-line community by imposing norms to the users only whenever there is *enough consensus* (enough evidence) about the necessity of including norms.

In Table 6.8 we can observe the summary of combinations between the *complaint power* and the *consensus degree* and the results obtained with the norm synthesis mechanism of DESMON.

Case	Convergence
$\mathsf{C}_{Pw} > \alpha^{nec}_{spec}$	1 Norm
$\mathsf{C}_{Pw} = \alpha^{nec}_{spec}$	No
$\mathsf{C}_{Pw} < \alpha^{nec}_{spec}$	0 Norms

Table 6.8: Summary of DESMON's macro analysis.

6.2.4 Analysis of synthesised normative networks

We now compare the size and structure of the normative networks synthesised by IRON, SIMON and DESMON to understand *how* they manage to synthesise normative systems. In particular, both SIMON and DESMON obtain the same results in terms of compactness. Therefore, for the sake of simplicity we will refer to both of them as DESMON.

Figures 6.1 and 6.2 depict prototypical normative networks synthesised by IRON and DESMON, respectively, for a population of 70 moderates and 30 spammers. There, each circle represents a norm, and each edge represents a generalisation relationship between two norms. In particular, *red* circles represent *general* norms, that is, norms that concisely represent several norms. On the one hand, IRON synthesises slightly generalised normative networks. The network depicted in Figure 6.1 contains 30 general (active) norms that concisely prohibit each one of the 30 spammers to upload spam in any section. In particular, each general norm generalises three norms that prohibit a user to upload spam in the three different sections. Figure 6.3 illustrates an example of the generalisation of norms n_1, n_2, n_3 to norm n_4 described below.

$$n_1 : \langle \{user(u_1),\ section(forum), \qquad cntType(spam)\},\ \ prh(upload)\rangle$$

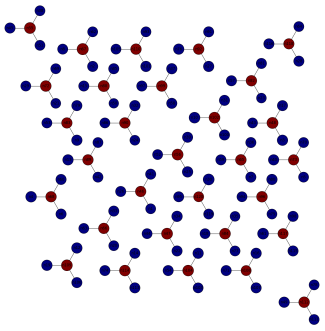

Figure 6.1: Example of a prototypical normative network synthesised by IRON.

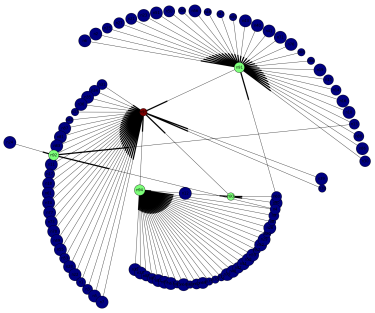

Figure 6.2: Example of a prototypical normative network synthesised by DESMON.

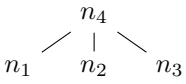

Figure 6.3: Example of IRON's generalisation.

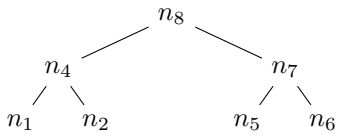

Figure 6.4: Example of SIMON's and DESMON's generalisation.

$$n_2 : \langle \{user(u_1),\ section(the\ reporter), cntType(spam)\},\ prh(upload)\rangle$$
$$n_3 : \langle \{user(u_1),\ section(multimedia),\ cntType(spam)\},\ prh(upload)\rangle$$
$$n_4 : \langle \{user(u_1),\ section(anySection),\ cntType(spam)\},\ prh(upload)\rangle$$

Norms n_1, n_2, n_3 prohibit user u_1 to upload *spam* to different sections, and n_4 generalises n_1, n_2, n_3, prohibiting user u_1 to upload *spam* to *anySection*. This lack of generalisation with respect to DESMON stems from IRON's conservative generalisation approach. This is because IRON cannot synthesise a general norm to prohibit to upload spam to all users in any section unless it synthesises one norm to prohibit each user to upload spam. However, it is impossible for this scenario since there are moderate users, who never upload spam, and hence IRON never generates norms for them.

By contrast, DESMON is capable of performing further generalisations than IRON. The network depicted in Figure 6.2 contains one general norm

$$\langle \{user(anyUser), section(anySection), cntType(spam)\}, prh(upload)\rangle$$

(which is the only active one) that prohibits *any* user to upload spam to *any* section. In fact, this norm generalises (and hence represents) each one of the norms

in the normative network. Thus, the represented normative system contains one single norm that prohibits all the users to upload spam to any section. Considering norms n_1, n_2, n_3, n_4, and norms n_5, n_6, n_7, n_8 described below, Figure 6.4 illustrates an example of this generalisation. It shows the optimistic generalisation from: (i) norms n_1 and n_2 to n_4 (which prohibits u_1 to upload *spam* to *anySection*); (ii) norms n_5 and n_6 to n_7 (which prohibits u_2 to upload *spam* to *anySection*); and finally (iii) from norms n_4 and n_7 to n_8 (which prohibits *anyUser* to upload *spam* to *anySection*);

$$
\begin{array}{lllll}
n_5 : \langle \{user(u_2), & section(forum), & cntType(spam)\}, & prh(upload)\rangle \\
n_6 : \langle \{user(u_2), & section(the\ reporter), & cntType(spam)\}, & prh(upload)\rangle \\
n_7 : \langle \{user(u_2), & section(anySection), & cntType(spam)\}, & prh(upload)\rangle \\
n_8 : \langle \{user(anyUser), & section(anySection), & cntType(spam)\}, & prh(upload)\rangle
\end{array}
$$

We then observe a structure which is different from the normative network synthesised by IRON. DESMON's network shows a hierarchical structure of two levels, where the most general norm has two of its predicates generalised. Moreover, the optimistic approach of DESMON permits to it to generalise without gathering all the possible evidences. Therefore, we conclude that DESMON manages to outperform IRON in terms of compactness of the normative system since it generates normative networks with a structure that IRON is not capable of generating.

6.2.5 Micro Analysis

We now analyse IRON's, SIMON's and DESMON's norm synthesis processes in the transition from converging to a normative system with norms, to converging to an empty normative system (in the case of DESMON). With this aim, we choose the balanced population of 50 moderates and 50 spammers (with complaint power $\mathsf{C}_{pw} = 0.5$), and study its convergence for low, medium and high consensus degrees, namely when α^{nec}_{spec} takes on values 0.3, 0.5 and 0.7 respectively.
Figure 6.5 depicts the evolution of the size of the normative system synthesised by each approach for a very low consensus degree ($\alpha^{nec}_{spec} = 0.3$). It shows averaged results of 10 simulations. Each value in the x-axis stands for a simulation *tick*, and the y-axis shows the number of norms in the normative system. As Figure 6.5 shows, from tick 0 to 500 none of the approaches synthesises norms. This happens because of the 500-tick *warm-up* period, in which users do not complain about contents. From tick 500 onwards, each approach starts synthesising norms and performing generalisations to compact its normative system. Finally, they are all able to converge to a normative system. This is so because the complain power of the population ($\mathsf{C}_{pw} = 0.5$) is over the consensus degree ($\alpha^{nec}_{spec} = 0.3$), and hence all synthesised norms are considered as necessary. However, while IRON converges to 30 norms, SIMON and DESMON converge to one single, general norm.
Figure 6.6 shows the size of the normative system along time for a medium consensus degree ($\alpha^{nec}_{spec} = 0.5$). In this case, none of the approaches are able to converge. Since the complain power of the population is 0.5 ($\mathsf{C}_{pw} = 0.5$), the necessity of norms fluctuates around 0.5, going above and below the consensus

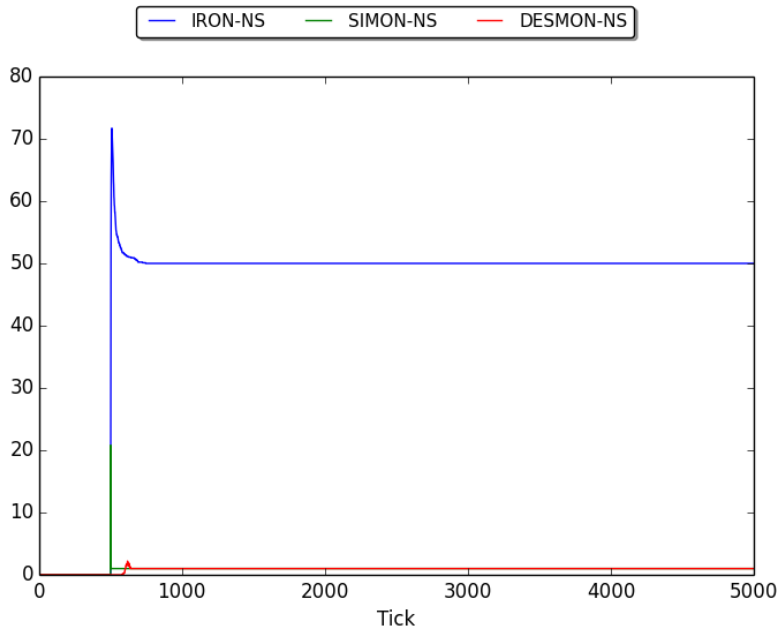

Figure 6.5: Cardinality of the NS for the different NSM approaches with low consensus degree ($\alpha^{nec}_{spec} = 0.3$).

degree α^{nec}_{spec}. Therefore, the three approaches continuously activate and deactivate norms, hence generalising norms and backtracking generalisations repeatedly. Finally, Figure 6.7 shows results for a high consensus degree ($\alpha^{nec}_{spec} = 0.9$). Since the complaint power ($\mathsf{C}_{pw} = 0.5$) is lower than the consensus degree, each approach should converge to an empty normative system. However, IRON and SIMON cannot converge to a normative system. This is so because IRON and SIMON are reactive to complaints, and hence they continuously re-activate discarded norms, even though they should remain inactive. By contrast, DESMON manages to converge to an empty normative system. In fact, it creates norms, but never activates them since their necessity, which fluctuates around 0.5, is below the consensus degree. Here DESMON benefits from its deliberative approach to norm synthesis, which considers cumulative complaints, instead of single complaints, to consider norms' activation.

6.3 Summary

In this final chapter we have empirically evaluated the norm synthesis machines connected with our multi-agent based on-line community simulator.

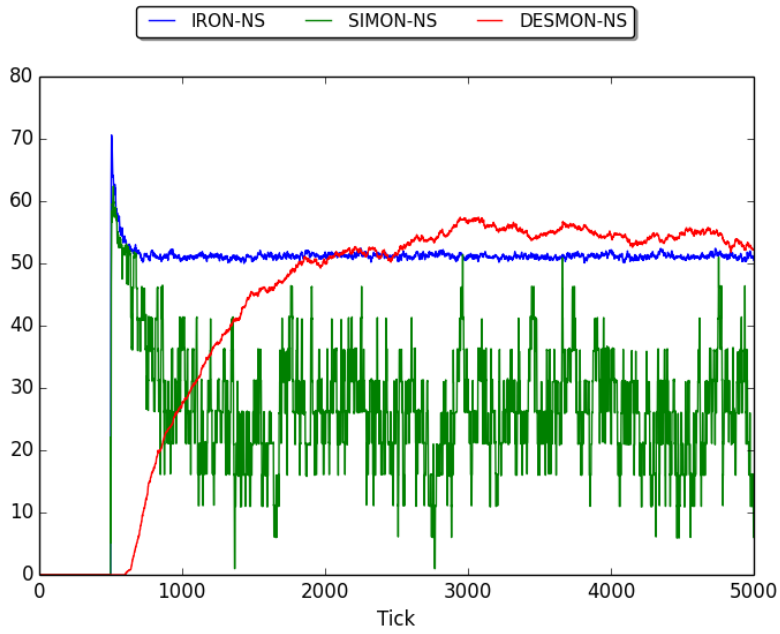

Figure 6.6: Cardinality of the NS for the different NSM approaches with medium consensus degree ($\alpha^{nec}_{spec} = 0.5$).

On the one hand, we have found a correlation between the capacity of complaining user population (complain power = C_{pw}) and the specialisation threshold (α^{nec}_{spec}), which can be interpreted as a consensus degree. These two values affect the convergence of the NSM's. As we can observe in Table 6.9 we detail a summary observed from the macro analysis of this chapter.

	Convergence		
Case	IRON	SIMON	DESMON
$\mathsf{C}_{Pw} > \alpha^{nec}_{spec}$	Multiple norms	1 Norm	1 Norm
$\mathsf{C}_{Pw} = \alpha^{nec}_{spec}$	No	No	No
$\mathsf{C}_{Pw} < \alpha^{nec}_{spec}$	No	No	0 Norms

Table 6.9: Summary of NSM's macro analysis.

Whenever the complain power is higher than the consensus degree ($\mathsf{C}_{pw} > \alpha^{nec}_{spec}$) all of the NSM's will converge to their compacter normative system, namely IRON (1 norm per each spammer), SIMON (1 norm prohibiting spam to all users) and DESMON (1 nor prohibiting spam to all users). On the contrary, when they both are equal (C_{pw}=α^{nec}_{spec}) convergence will not occur. This is so, because the

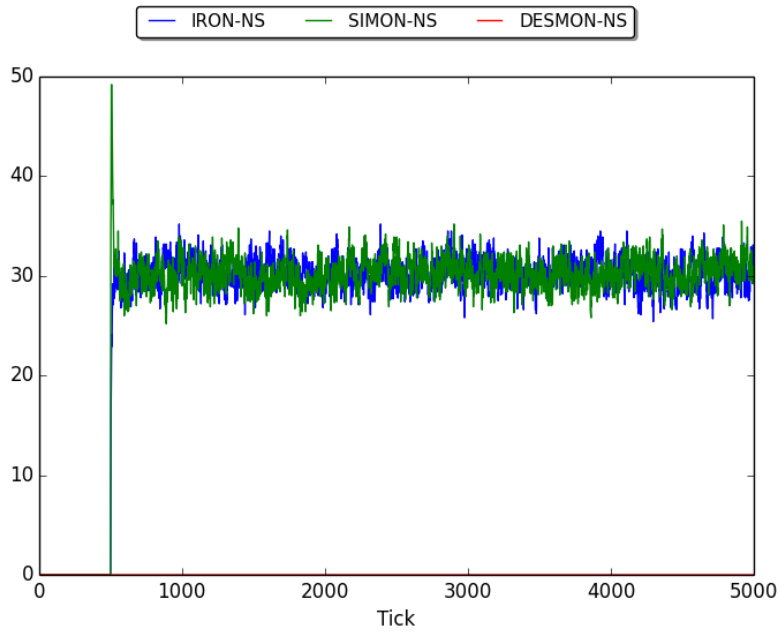

Figure 6.7: Cardinality of the NS for the different NSM approaches with high consensus degree ($\alpha^{nec}_{spec} = 0.7$).

NSM's are continuously activating and deactivating norms, without reaching a convergence state. Finally, when the consensus degree is bigger than the complain power ($\mathsf{C}_{pw} < \alpha^{nec}_{spec}$) the reactive approaches as IRON and SIMON will not converge to any normative system, as they are too reactive to complains and precipitates in the addition of the norms. However, the deliberative approach, DESMON, will converge to a normative system without norms as the minimum consensus degree to create norms is not achieved by the complains of the users (C_{pw}. Thus, DESMON is capable of capturing the opinion of the community and only generates norms whenever a minimum consensus is reached.

Moreover, we proved that DESMON also overcomes the compactness drawback of IRON. As shown in Section 6.2.4, we can observe that as DESMON is based on SIMON it inherits the optimistic norm generalisation approach and makes normative systems that are compacter than IRON's. The normative systems synthesised by DESMON are based on a simple norm prohibiting the upload of spam content, while IRON synthesize one norm per each user prohibiting each one the upload of spam content.

Furthermore, in the micro analysis we observed the transition from convergence to no convergence for a certain population of 50 moderates and 50 spammers.

Focusing on how the normative systems fluctuates with different NSM's and different consensus degrees.

Chapter 7

Conclusions and future work

In this chapter we first summarize the work developed during this thesis drawing the most notable conclusions attained from it, and finally present some lines for future research.

7.1 Summary

In this thesis we have build DESMON, a deliberative and participatory approach for norm synthesis, overcoming the drawbacks of SIMON (the improved version of IRON presented in [Morales et al., 2014a]). Moreover, we have applied DESMON to our novel and state-of-the-art multi-agent on-line community simulator, where real humans that share, view and complain about contents are modelled by agents. The aim of this application is to build frictionless communities, that is, communities that avoid frictions (conflicts) between their users. The conflicts that arise as frictions, and hence synthesising norms of the normative system, are the complaints about the contents made by the users from the same community.

Therefore, we can divide this thesis in three main contributions. First, we have tackle the drawbacks of SIMON (Chapter 3) for: *reactiveness* of the norm addition and *conservativeness* of the norm deletion. We introduce DESMON, a general approach to norm synthesis based on SIMON. DESMON is capable of synthesising norms in a deliberative manner being a participatory approach capable to synthesise norms acquiring the majority of users' opinion. Moreover, it presents a new state transitions for the norms life cycle (Figure 3.1) to monitor the norms performance before activating them, as the norm deactivation is done.

Second, we have presented an on-line community simulator (Chapter 4) that provides a MAS scenario for the norm synthesis approaches to regulate. Our sim-

ulator allows to design different populations of users with different behaviours, to build a realistic virtual community simulator. Moreover, it provides different execution modes to analyse on-line the norm synthesis: (i) it allows the visualization on display to visualise the agents interactions in the community, (ii) a batch mode without visualization and (iii) a parallel execution mode for computer clusters. Furthermore, we have configure and exploit the norm synthesis machines in order to synthesize norms for our on-line community (Chapter 5). With this aim we have described our domain specific implementation of the NSM's inputs. As a result, we obtain synthesis of norms for the users of the on-line community, preventing them from uploading inappropriate contents, and hence avoiding friction between users.

Third, we have performed an empirical evaluation (Chapter 6) with the aim of studying the differences between different approaches IRON, SIMON and DESMON. On the one hand, we have performed a *macro analysis* to analyse *when* it occurs the convergence and *what* type of normative systems it synthesize for each different approach. On the other hand, we have also performed a *micro analysis* to focus on the transition from convergence to non convergence and analyse how they manage to synthesise norms. These experiments shows us that: (i) SIMON and DESMON outperforms IRON in compactness and (ii) DESMON is able to deliberatively synthesise and converge to normative systems without norms, which IRON and SIMON are not able because they are reactive approaches that give more importance to the activation and maintenance of the norms than the deactivation.

7.2 Future work

As future work, we have seen several steps we would like to take into account.

First of all, we have used computer simulation as a method of social research. We aim to create a model of the real world on-line communities, that are our targets, and simulate them. Once that we have gathered the data of the model we have to validate it with the data of a real world scenario, in our case this would be to make a test with real persons. The logic of the simulation is shown graphically on Figure 7.1 taken from [Gilbert and Troitzsch, 2005]. Therefore, after comparing the gathered data by our simulated model and the real world scenario we could follow with the next steps.

Secondly, we have made experiments with 100 agents that, as described in [Nielsen, 2006], represents the heavy contributors of an on-line community of 10.000 users. Even though, as it is observed in the usage numbers of the social networks in [Keckley and Hoffmann, 2010] we are far away from the biggest social networks of the market (e.g., Facebook, Twitter, Linkedin, YouTube). For this purpose, the next step after validating our data, we are going to seek for the scalability of our system. Focusing on the coverage of communities with large

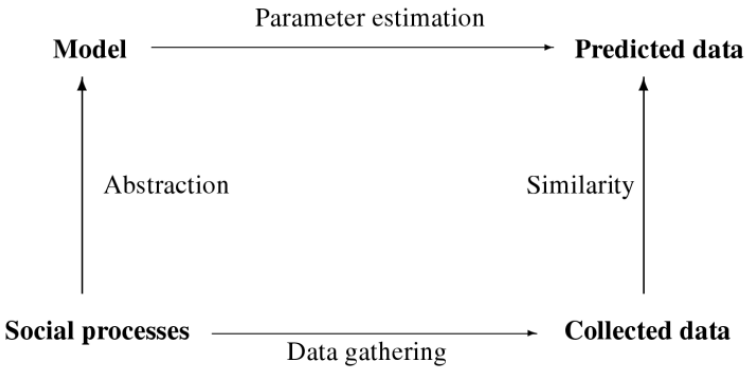

Figure 7.1: The logic of the simulation.

quantities of users and big movement of data.

Thirdly, the regulations that we have in the real life have a punishment if they are infringed. This punishments make the rule or law stronger, as people is frightened of the consequences. In our on-line community scenario we haven't taken into account the punishments the users would have by infringing a norm. In real virtual communities exists some punishments that more or less act in consensus between different sites. This punishments as described in [Hall and George, 1999] are:

1. User banning: The user site is banned and no one from that address can access the virtual community.

2. User deletion "Toading": The user is deleted. If the user wants to access again has to create a new account.

3. User imprisonment: The user is placed within a limited area of the community and prohibited from interacting with other users (during certain time).

4. Removal of status/ experience: The user will loose all the "points" or values of the community that make some users more relevant than others.

Therefore, in order to make a more realistic approach we would apply punishments after the infringement of some norms or after the infringement of several, depending the gravity of the infringement. As an initial guess we could apply a state machine to the users, making a transition between the punishments described above and the original state, as depicted in Figure 7.2 (following the level of gravity with the colour intensity).

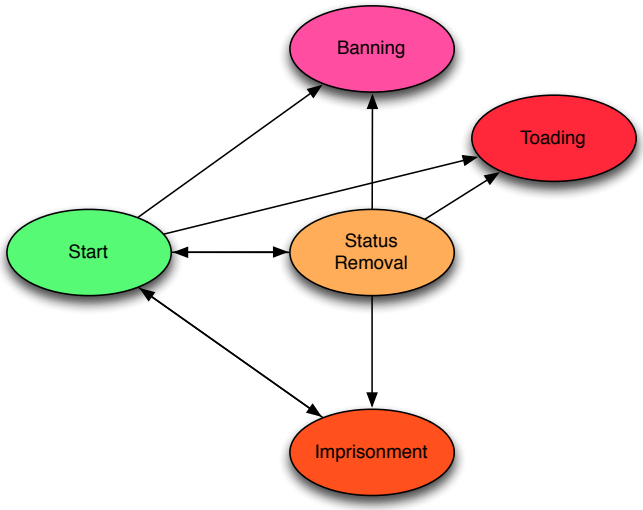

Figure 7.2: State machine transitions for users punishments.

Finally, our last aim and objective to pursue will be the application of the norm synthesis mechanism to an actual world on-line community, automatically synthesising norms for humans and creating a participatory approach where the opinions of all the users are listened. And as the norm synthesis mechanism is domain independent we could extrapolate this mechanism for other purposes, such as the creation of new civil participation mechanisms for avoiding frictions in real cities.

Bibliography

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59.

Armengol, E. and Plaza, E. (2000). Bottom-up induction of feature terms. *Machine Learning*, 41(3):259–294.

Boella, G., van der Torre, L., and Verhagen, H. (2006). Introduction to normative multiagent systems. *Computational & Mathematical Organization Theory*, 12(2-3):71–79.

Chou, Y. (1969). *Statistical analysis: with business and economic applications*. Holt, Rinehart and Winston Quantitative Methods Series. Holt, Rinehart and Winston.

Christelis, G. and Rovatsos, M. (2009). Automated norm synthesis in an agent-based planning enviroment. In *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS'09, pages 161–168. IFAAMAS.

DAEDALUS (2009). Stilus Forum: Automatic moderation of web forums. `http://www.daedalus.es/en/blog-and-resources/on-stilus/stilus-forum-automatic-moderation-of-participation-forums/`. [Online; accessed April 20, 2015].

Dignum, F. (1999). Autonomous agents with norms. *Artif. Intell. Law*, 7(1):69–79.

Fitoussi, D. and Tennenholtz, M. (1998). Minimal social laws. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 26–31. John Wiley & Sons LTD.

Fitoussi, D. and Tennenholtz, M. (2000). Choosing social laws for multi-agent systems: Minimality and simplicity. *Artificial Intelligence*, 119(1):61–101.

García-Camino, A., Rodríguez-Aguilar, J. A., Sierra, C., and Vasconcelos, W. (2009). Constraint rule-based programming of norms for electronic institutions. *Autonomous Agents and Multi-Agent Systems*, pages 186–217.

Gil Alonso, A. and Roig, H. (2007). Sourpanel: Software for automatic moderation of comments. `http://www.sourtech.com/servicios/moderacion-comentarios-18`. [Online; accessed April 20, 2015].

Gilbert, N. and Conte, R. (1995). *Artificial Societies: The Computer Simulation of Social Life*. Taylor & Francis, Inc., Bristol, PA, USA.

Gilbert, N. and Troitzsch, K. (2005). *Simulation for the social scientist*. McGraw-Hill International.

Griffiths, N. and Luck, M. (2010). Norm Emergence in Tag-Based Cooperation. In *9th International Workshop on Coordination, Organization, Institutions and Norms in Multi-Agent Systems.*, COIN'10, pages 80–87.

Guzmán, J. (12008). Keepcon: a company based on semantic technology that applies content moderation. `http://www.keepcon.com`. [Online; accessed April 20, 2015].

Hall, L. and George, C. E. (1999). Law and punishment in virtual communities. *Proceedings of Cybersociety*.

Hinds, D. and Lee, R. M. (2011). Assessing the social network health of virtual communities. In Hershey, E. I. R. M. A., editor, *Virtual Communities: Concepts, Methodologies, Tools and Applications*, chapter 49, pages 715–730. IGI Global.

Keckley, P. H. and Hoffmann, M. (2010). Social networks in health care: Communication, collaboration and insights. *Deloitte Center for Health Solutions*.

Kittock, J. E. (1993). Emergent conventions and the structure of multi-agent systems. *L. Nadel and D. Stein, eds.*

Lanas, P. and Garzón, D. (2005). FansCup. `http://www.fanscup.com`. [Online; accessed April 20, 2015].

Lewis David, K. (1969). Convention: A philosophical study.

Morales, J., López-Sánchez, M., and Esteva, M. (2011). Using Experience to Generate New Regulations. In *International Joint Conference in Artificial Intelligence (IJCAI'11)*, pages 307–312. AAAI Press, USA.

Morales, J., Lopez-Sanchez, M., Rodriguez-Aguilar, J. A., Wooldridge, M., and Vasconcelos, W. (2013). Automated synthesis of normative systems. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 483–490. International Foundation for Autonomous Agents and Multiagent Systems.

Morales, J., Lopez-Sanchez, M., Rodriguez-Aguilar, J. A., Wooldridge, M., and Vasconcelos, W. (2014a). Minimality and simplicity in the on-line automated synthesis of normative systems. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 109–116. International Foundation for Autonomous Agents and Multiagent Systems.

Morales, J., Mendizabal, I., Rodriguez-Aguilar, J. A., Sánchez-Pinsach, D., Lopez-Sanchez, M., Wooldridge, M., and Vasconcelos, W. (2015a). Extending normlab to spur research on norm synthesis (demonstration).

Morales, J., Mendizabal, I., Sanchez-Pinsach, D., López-Sánchez, M., and Rodríguez-Aguilar, J. A. (2015b). Using iron to build frictionless on-line communities. *AI Communications*, 28:16.

Morales, J., Mendizabal, I., Sánchez-Pinsach, D., Lopez-Sanchez, M., Wooldridge, M., and Vasconcelos, W. (2014b). Normlab: A framework to support research on norm synthesis (demonstration).

Nielsen, J. (2006). The 90-9-1 rule for participation inequality in social media and online communities. http://www.nngroup.com/articles/participation-inequality/.

North, M., Collier, N., Ozik, J., Tatara, E., Macal, C., Bragen, M., and Sydelko, P. (2013). Complex adaptive systems modeling with repast simphony. *Complex Adaptive Systems Modeling*, 1(1):3.

Ostrom, E. (1990). *Governing the commons: The evolution of institutions for collective action*. Cambridge university press.

Salazar, N., Rodriguez-Aguilar, J. A., and Arcos, J. L. (2010). Robust coordination in large convention spaces. *AI Commun.*, 23(4):357–372.

Sen, O. and Sen, S. (2010). Effects of social network topology and options on norm emergence. In *Proceedings of the 5th international conference on Coordination, organizations, institutions, and norms in agent systems*, COIN'09, pages 211–222, Berlin, Heidelberg. Springer-Verlag.

Shoham, Y. and Leyton-Brown, K. (2009). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York.

Shoham, Y. and Tennenholtz, M. (1995). On social laws for artificial agent societies: off-line design. *Journal of Artificial Intelligence*, 73(1-2):231–252.

Villatoro, D., Sabater-Mir, J., and Sen, S. (2011). Social instruments for robust convention emergence. In Walsh, T., editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, IJCAI'11, pages 420–425. IJCAI/AAAI.

Walker, A. and Wooldridge, M. (1995). Understanding the emergence of conventions in multi-agent systems. In *International Conference on Multiagent Systems*, ICMAS'95, pages 384–389.

Wooldridge, M. (2009). *An introduction to multiagent systems*. John Wiley & Sons.

Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(02):115–152.

Yu, C., Zhang, M., Ren, F., and Luo, X. (2013). Emergence of social norms through collective learning in networked agent societies. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS'13. IFAAMAS.