Chapter 2
Requirements for Designing Normative Open Multi-Agent Systems

In this chapter, we try to answer the research question: *Which are the common requirements for developing normative open MAS?* As is presented in Sect. 1.1, normative open systems have common features, challenges, and requirements that must be considered during their development. This section analyzes the characteristics that an agent methodology for analyzing and designing systems of this kind should have. This analysis is derived from our previous studies [54, 56, 58], related literature [12, 31, 34, 40, 75, 86], and the study of case studies from different application domains [15, 24, 61, 79, 87, 115].

In the specialized literature, there is no consensus about the terminology that must be used to specify normative open MAS. Therefore, in this section, we analyze the requirements for designing normative open systems from a semantic point of view and associate these semantics to specific terms in order to reuse them in the following sections. These specific terms are highlighted in bold.

Software methodologies are composed by the specification of design constructs, a development process, and a set of guidelines that supports or automatizes some of the development decisions. In that sense, the rest of the section is organized as follows: First, Sect. 2.1 analyzes the metamodel constructions and design abstractions that are necessary to represent systems of this kind. Second, Sect. 2.2 analyzes the support during the development process that it is necessary in order to completely analyze and formalize these systems. Finally, Sect. 2.3 analyzes how the final design of the system should be validated.

2.1 Design Abstractions

This section analyzes the design abstractions that a metamodel for modeling normative open MAS should integrate. These design abstractions are related to the common properties of systems of this kind detailed in Sect. 1.1.

E. Garcia et al., *Regulated Open Multi-Agent Systems (ROMAS)*,
DOI 10.1007/978-3-319-11572-6_2

Regarding the assumption of autonomous and social behavior in normative open systems, we conclude that for designing systems of this kind, it is necessary to explicitly specify individual entities (called **agents** in MAS) and **organizations**. Agents represent individual entities with their personal objectives, capabilities, and resources [119]. Organizations represent a group of agents that have a common objective or real-world institutions [54]. The explicit representation of organizations at design time is beneficial in the sense that: (1) Organizations allow to divide a large subsystem in subsystems facilitating the design, the implementation, and the maintenance of the model [75]; (2) Organizations are a high-level abstraction, very close to real life that facilitates the design and the comprehension of the clients and domain experts; (3) Organizations allow creating different contexts inside the same system (each context can have its own resources, regulations, and features) [11]. The internal structure of the organizations of the system will determine how the functionality of the system is divided between its entities, the social relationships and communications among entities and how the system interacts with its environment [73].

Regarding the assumption of heterogeneity, interactions and interchanges of services and products should be standardized and formally described in order to isolate the internal characteristics of the actors of the system from their interactions with the rest of the system [37]. Common to other works [49, 97], we propose standardizing the interchanges by means of **services**. Services standardize the interactions without restricting the technology or the process followed in order to offer this functionality [56]. Services are a powerful interaction mechanism at design and also at implementation time. The use of services during the design time helps in the specification of different levels of abstraction. Services allow to specify what an entity offers or requires from the system separately from the internal features of this entity and separately from how it is going to offer or use this functionality.

Regarding the assumption that we deal with systems in regulated environments, we conclude that the behavior of the entities and institutions in the system should be bounded by rights and duties. **Norms** provide a mechanism to explicitly represent which actions are permitted, forbidden, and obliged inside the system [88]. In that sense, norms provide users and members of a system with expectations about what other agents will do or not do. Norms provide confidence in the quality and correctness of what occurs in the system. Norms avoid critical status of the systems to occur and also try to ensure that the system follow the law regulations established in a specific domain or institution. The explicit representation of norms forces developers to analyze and consider the normative environment at design time [12]. Beyond that it allows external developers to know which behavior is expected from the software that he/she is going to implement.

The interaction of several institutions and entities from different spheres of control arise the need of specifying **different normative contexts** inside the same system [61]. First, we will need to specify the normative context of a system. It is considered to be the set of norms that regulates the behavior of each entity and the set of contracts that formalizes the relationships between entities and institutions. Second, for each institution involved in the system, we will need to specify the normative context of

this institution. They are specified by the set of norms that affects the entities that are members of this institution. Third, we will need to specify the normative context of each entity. It is specified by the set of norms that directly affects the behavior of this entity in a specific moment.

Regarding the scope of a norm, tree types of norms can be considered [12]. First, the **institutional norms** are the norms that regulate the behavior inside a specific institution or group of entities. These norms are related to internal regulations of this institution in the real-world or law restrictions associated to this kind of institutions in this domain. Second, the **role norms** are the norms that any entity playing a specific role inside the system must follow. The term role is used to specify a set of functionalities inside a system. The role abstraction is very close to real-life systems. For example, every person that interacts inside a university has a role (students, teachers, directors of departments, etc.). Third, the **agent's norms** are the norms that affect only to a specific entity of the system. Every individual entity may have special rights or restrictions associated to its own design and implementation. These norms are not related to the general structure of the system or the roles that this entity plays, but with the specific features of each individual entity. For example, in a virtual market where all the clients are obliged to pay in advance, one specific client may have arrived to an agreement with the company that allow this client to pay after receiving the goods.

Norms can also specify the internal structure of the system and the social relationships between their components. This means that the social structure emerges from the norms and social relationships between entities. Moreover, the use of norms to specify the structure allows the entities to reason about the structure of their system, and allows the structure to be updated dynamically at runtime. In the literature, norms of this kind are called **structural norms** [43].

Social structure architectures imply the specification of the relationship between several entities of the system. In dynamic and flexible systems, as well as in real human societies, the specific terms of the social relationship between entities can be negotiated between the entities involved. Common to other works [40, 110], we use the abstraction of **contract templates** to specify at design time the features that any contract of a specific type should have. The use of contract templates to specify these relationships provides flexible architectures and maintains the autonomy of the system about how to implement their commitments. In this work, these kinds of contracts are called **social relationship contracts** [40]. Contracts have been used in many domains in order to formalize restrictions without compromising the autonomy of the entities. This is because contracts are expressive and flexible. They allow agents to operate with expectations of the behavior of other agents based on high-level behavioral commitments, and they provide flexibility in how the autonomous agents fulfill their own obligations [115]. Contracts also allow the negotiation of the specific terms of the engagement between a stakeholder and a role. Although contracts should be specified and negotiated at runtime, at design time contract templates should be defined in order to specify contract patterns that any contract of this type should fulfill.

Regarding the assumption of openness, the design abstractions used should be able to specify how an external entity interacts with the system and how it becomes part of the system. Using a service-oriented architecture when an external entity needs to interact with the system, it only have to follow the standard specified by the service. In the case that an external entity wants to be integrated in the system, i.e. to become part of the system by offering part of the internal functionality of the system, it has to acquire a specific role of the system. Commonly in agent literature, the internal functionality of complex systems is divided using roles [119]. A role is high-level abstraction that allows specifying system using terms close to the ones used in real-life systems. For example, in a commercial interchange, we will use the roles *client* and *provider*. The external entities that want to be integrated in the system can be heterogeneous and they can be developed outside the scope of the system. However, once a stakeholder enters in a normative system its behavior should fulfill the rights and duties of the roles that it is playing. Therefore, when an entity wants to play a specific role, it has to be informed about the rights and duties associated to this role. Moreover, flexible and dynamic systems may allow entities to negotiate at runtime how each entity is going to play each role. Therefore, the rights and duties associated to each role should be described by means of contracts. Similarly to other works [43], the contract templates that specify at design time the general terms that any entity should fulfill in order to play a specific role are called **play role contract**.

Interchanges of services and resources are formalized at runtime; however, in regulated systems it may be necessary to specify at design time which kind of relationships are allowed and under which terms. Therefore, it is necessary to specify contract templates that formalize these restrictions and may establish the interaction protocols that should be executed in order to negotiate, execute, and resolve conflicts related to these contracts. In this work, these types of contract templates are called **contractual agreements**.

2.2 Support During the Development Process

The development of normative open MAS requires complex tasks such as the integration of the individual and social perspective of the system or the integration of the system restrictions in the design of the individual entities. Therefore, software methodologies should provide a set of guidelines that simplifies or automatizes these tasks. Following we present a summary of the most important guidelines that a complete methodology for normative open MAS should provide.

One of the challenges in the design of normative open systems is determining the most suitable social structure and when the system should be structured into suborganizations [75]. Although the process of analyzing which is the most suitable organizational topology could seem to be as simple as mirroring the real-world structure, it is in fact rather complex. On the one hand, if the system supports or automates existing relationships between institutions, developers should identify and analyze these relationships in order to extract the specific requirements.

On the other hand, if the system allows new interactions, they could change the existing social structure. Since the structure of the system determines the relationships and interaction among the entities and the division of tasks between them, a bad choice in the selection of the social structure could derive problems such bottlenecks, a reduction of the productivity of the system, and an increase of the reaction time of the system [116]. Therefore, methodological guidelines that support the decision of which is the most suitable structure are necessary [41, 74].

Another challenge is the identification and formalization of the normative context of a system. In the previous section, a set of different types of norms that should be formalized at design time are introduced. These norms can be derived from: (1) the specific requirements of the system (e.g., a system in which the main goal is to increase productivity during a specific period would forbid any entity from taking a vacation during this period) [104]; (2) legal documents that formalize governmental law or internal regulations of each institution (e.g., the National Hydrological Plan, the governmental law about water right interchanges) [17]; and (3) design decisions [116]. The identification of the normative context of a system is not trivial because: (1) the description of the requirements of the system provided by domain experts might be incomplete; (2) individual entities might have their own goals that conflict with the goals of the system; (3) in systems composed of different institutions, each could have its own normative context that needs to be integrated into an overall system; and (4) legal documents are written in plain text, which means that the terminology of the domain expert and these legal documents could be different.

A poor or incomplete specification of the normative context can produce a lack of trustworthiness and robustness in the system. In open systems in which every entity could be developed by a different institution, if the rights and duties are not formally specified, an entity that tries to join a system would not know how to behave. Entities could perform actions that harm the stability of the system (e.g., in a non-monopoly system, a client could buy all the resources of one type). Therefore, specific guidelines should be added to the **requirements analysis** stage in order to identify and formalize the norms that are directly related to the requirements of the system. Also, specific guidelines for identifying the norms that should be implemented in a system derived from the **legal documents** associated to the system should be provided. This identification is a complex process because such documents are usually written in plain text and the semantic meaning of the concepts described in the legal documents and in the system design can be inconsistent.

The **structure of the system** and the **relationship between the roles and entities of the system** can be explicitly specified by means of norms and contracts. As is presented in previous sections, this explicit representation provide benefits, however, it can be a complex task in complex systems. Therefore, the methodology should provide specific guidelines that simplify and automatize the task. In that sense, the methodology should provide specific guidelines for identifying institutional, role and agent norms, as well as guidelines to formalize play role and social relationship contracts.

Another challenge is the identification of when is beneficial for the system that two entities collaborate [102]. A complete methodology should help developers in

this process. Beyond that, the formalization at design time of these interchanges can be a complex task. The formalization should specify which terms of the contract are mandatory and which are forbidden. So, a complete methodology should offer specific guidelines to the **identification and formalization of contractual agreements**.

Contracts are more than a set of norms [22, 89]. The specification of **negotiation, execution and conflict resolution protocols** is also an important issue in contract-based systems [114]. These protocols should fulfill the normative context of the contract and ensure that all the terms of the contract are agreed and executed. Therefore, methodological guidelines could be very beneficial in order to avoid incoherence between the contracts' clauses and their protocols and to ensure the correctness and completeness of these protocols.

2.3 Evaluation of the Final Design

The validation of the fact that the designed system fulfills all the requirements identified in the analysis stage and the verification of the coherence of the system are common open issues in any development approach. For normative open systems, these validations and verification have even greater importance due to two specific features. First, systems of this kind integrate the global goals of the system with the individual goals of each party, where these parties are completely autonomous and their interests may conflict. It is thus crucial to help developers to verify that the combined goals of the parties are coherent and do not conflict with the global goals of the system. If any incoherence is detected, the developer should be able to determine when this issue will affect the global goals and whether it is necessary to introduce norms to avoid related problems. Second, such systems usually integrate different normative contexts from the different organizations involved, which must be coherent with the contracts defined in the system. It is necessary to ensure that each single normative context has no conflicts, and also that the composition of all the normative contexts is itself conflict-free. In this respect, an open question is how consistency and coherence of norms and contracts can be automatically checked inside an organization. Therefore, guidelines for validating that the design **fulfills the normative requirements** and for verifying the **coherence of the goals of the different parties in the system** and the **coherence of the normative context** should be offered by the methodology and integrated in the development process.

As well as verification and validation, traceability is another topic that has a special importance in normative open MAS. Requirements traceability refers to the ability to describe and follow the life of a requirement, in both forward and backward direction [23]. Traceability improves the quality of software system. It facilitates the verification and validation analysis, control of changes, as well as reuse of software systems components, and so on. The ability of following the life of a requirement associated to a norm is even more important due to the dynamicity of the normative contexts of a system. For example, in the mWater case study [59], the whole system should follow the National Hydrological Plan legislation. Without traceability, any

change in this law would imply the revision of the whole system. However, if it would be possible to trace each norm individually, only the norms that had changed should be revised and only the parts of the system affected by these norms should be redesigned. Therefore, **traceability of the normative context** is a desired feature in a methodology for developing normative MAS.

http://www.springer.com/978-3-319-11571-9

Regulated Open Multi-Agent Systems (ROMAS)
A Multi-Agent Approach for Designing Normative Open Systems
Garcia, E.; Giret, A.; Botti, V.
2015, VII, 151 p. 57 illus., Hardcover
ISBN: 978-3-319-11571-9