Multi-agent systems consist of a number of interacting autonomous agents, each of which is capable of sensing its environment (including other agents) and deciding to act in order to achieve its own objectives. In order to guarantee the overall design objectives of multi-agent systems, the behavior of individual agents and their interactions need to be regulated and coordinated \[23,29,30\]. The development of multi-agent systems therefore requires programming languages that facilitate the implementation of individual agents as well as mechanisms that control and regulate individual agents’ behaviors. It also requires computational tools to test and verify programs that implement multi-agent systems \[7\].

In the last two decades, various multi-agent programming languages have been proposed to facilitate the implementation of multi-agent systems \[6,7,11\]. These programming languages differ in detail, despite their apparent similarities \[8,9,15,16,18,20,21,22,24,25,28\]. They differ from each other as they provide programming constructs for different, sometimes overlapping, sets of agent concepts and abstractions. The expressivity of the programming constructs for overlapping agent concepts and abstractions may differ from one programming language to another. This is partly due to the fact that they are based on different logics or use different technologies. Moreover, some of these programming languages have formal semantics, but others provide only an informal explanation of the intended meaning of their programming constructs. Also, some programming languages capture in their semantics specific rationality principles that underlie agent concepts (e.g., relating beliefs and goals), while such principles are assumed to be implemented explicitly by agent programmers in other programming languages. Finally, some agent-oriented programming languages are based on declarative style programming, some are based on imperative style programming, and yet others combine these programming styles.

An example of a multi-agent programming language which combines declarative and imperative programming styles is 2APL (A Practical Agent Programming Language) \[11\]. 2APL is a BDI-based agent-oriented programming language that is developed to facilitate the implementation of individual agents in terms of cognitive concepts such as beliefs, goals, plans, events, and reasoning rules. The interpreter of 2APL is basically a decision process that selects and performs actions based on a sense, reason, and act cycle, often called deliberation cycle. In order to verify 2APL programs and to check their correctness a logic...
is developed that can be used to reason about such programs \[12\]. The logic, which is based on PDL (Propositional Dynamic Logic), can be used to reason about possible executions of agent programs. Also, a prototype implementation of the 2APL interpreter in Maude \[10\] (a rewriting logic software) is developed. This implementation allows the use of the model checking tools that come with Maude and allows the verification of 2APL programs.

One of the challenges in the design and development of multi-agent systems is to control and coordinate the behavior and interaction of individual agent programs in order to guarantee the overall system design objectives. There are two general approaches to develop coordination mechanisms for multi-agent systems: endogenous and exogenous coordination approaches. In the endogenous coordination approach, the coordination mechanism is incorporated in the individual agent programs while in the exogenous coordination approach the coordination mechanism is a separate program/module ensuring a clear separation between individual agent programs and their coordination mechanisms. An advantage of the exogenous approach is that the overall system design objectives can be developed and verified independent of the individual agent programs. Exogenous coordination mechanisms for multi-agent systems can be designed and developed in terms of concepts such as action synchronization and resource access relation \[3,23\], but also in terms of social and organizational concepts (e.g., norms, roles, groups, responsibility) and processes (e.g., monitoring actions and sanctioning mechanisms) \[14,17,19,26,27\].

A specific proposal for the design and development of exogenous coordination mechanisms for multi-agent systems is the introduction of an organization-based programming language \[12,13\]. This proposed programming language is designed to facilitate the implementation of organization based coordination artifacts in terms of norms and sanctions. Such artifacts, also called multi-agent organizations, refer to norms as a way to signal when violations take place and sanctions as a way to respond in the case of violations. Basically, a norm-based artifact observes the actions performed by the individual agents, determines the violations caused by performing the actions, and possibly imposes sanctions. The operational semantics of this programming language make it easy to prototype it in Maude \[10\]. The Maude implementation of the interpreter of this organization-based coordination language \[5\] allows model checking organization-based programs. Finally, in order to reason about organization-based coordination programs, a logic is devised to reason about such programs \[12\].

References