Data and Norm-aware Multiagent Systems for Software Modularization (Position Paper)

Matteo Baldoni[1], Cristina Baroglio[1], Diego Calvanese[2], Roberto Micalizio[1], and Marco Montali[2]

[1] Università degli Studi di Torino — Dipartimento di Informatica
c.so Svizzera 185, I-10149 Torino (Italy)
{firstname.lastname}@unito.it
[2] Free University of Bozen-Bolzano — KRDB Research Centre
Piazza Domenicani 3, I-39100 Bolzano, Italy c.so Svizzera 185, I-10149 Torino (Italy)
lastname@inf.unibz.it

Abstract. This work surveys the key proposals to the modularization of software, and trace them back to the common ground provided by Meyer's three forces of computation: processor, object, and action. We advocate that a paradigm should provide a good balance in exploiting *all* such forces, and support this stance by explaining the weaknesses of the examined proposals. Then, we focus on the agent paradigm because it emerges as pivotal for the achievement of a good balance. We trace directions that we think should be followed in order to complete the model, identifying, in particular, in data-awareness jointly with a norm-based representation of how data evolution is governed the key advancements that would bring to fullness the modularization of software.

1 Introduction

Research on agents and multiagent systems introduced many abstractions and tools to help designing modularized software, e.g., organizations, interaction protocols, artifacts, norms. This work provides a wide and systematic account of the major approaches to modularization, that were developed both by research on multiagent systems and by other research communities, leveraging Meyer's three forces of computation [31] as reference dimensions, along which all the considered proposals are positioned. The aim of this survey is to identify the lacks of the state of art together with possible directions of research. The paper is so organized. Section 2 introduces Meyer's forces of computation. Section 3 shows how functional decomposition, object-orientation, the actor model, business processes, artifact-centric approaches can be seen as manifestations of either the processor force or of the object force. Section 4 explains the strengths and the lacks of proposals from research area on agents. Section 5 explains the value of the action force, considered as ancillary by most of the examined approaches. Section 6 traces as open directions of research data and information-awareness

jointly with an extended norm-based representation that includes rules that govern the environment. Conclusions end the paper.

2 Meyer's forces: Processor, Action and Object

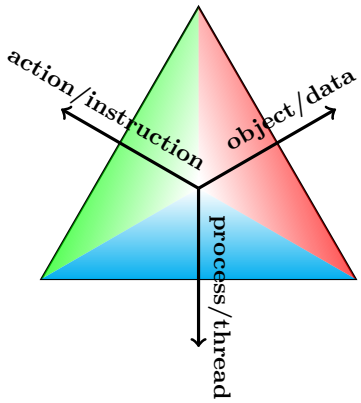

Fig. 1. Meyer's three forces of computation [31, Chapter 5, page 101].

The goal of software engineering is the production of quality software [31]. Among the desired qualities, *correctness* is the ability of software products to perform their tasks as defined by their specification; *robustness* is the ability to react appropriately to abnormal conditions; *extensibility* is the ease of adapting software products to changes of specification; *reusability* is the ability of software elements to serve for the construction of many different applications. In order for software to show these properties, it is necessary to identify proper *modularization mechanisms* that allow the programmer to design and develop software in a systematic way. To evaluate a modularization mechanism, one should not only consider how easy it is, by adopting it, to obtain a software module from scratch, but also how easy it is to maintain that software over time. We decided to use *Meyer's forces of computation* as a common ground for comparing the different proposals because they provide a neutral touchstone, unrelated to any specific programming approach or modularization mechanism. According to Meyer, three forces are at play when we use software to perform some computations (see Figure 1): *processors*, *actions*, and *objects*. A processor can be a *process* or a *thread* (in the paper we use both the terms processor and process to refer to this force); actions are the *operations* that make the computation; objects are the *data* to which actions are applied.

A software system, in order to execute, uses processes to apply certain actions to certain objects. The form of the actions depends on the considered level

of granularity: they can be instructions of the programming language as well as they can be major steps of a complex algorithm. Moreover, the form of actions conditions the way in which processes operate on objects. Some objects are built by a computation for its own needs and exist only while the computation proceeds; others (e.g., files or databases) are external and may outlive individual computations. In the following we analyse the most important proposals concerning software modularization, showing how they (sometimes implicitly) give more or less strength to Meyer's forces, and the drawbacks that follow.

3 Processor vs. Object: the big fight

It becomes apparent that processor and object are the two principal forces along which most approaches to modularization have been developed so far, while the action force remained subsidiary to one or another.

Functional Decomposition. The top-down *functional decomposition* is probably the earliest approach to building modularized software; it relies on a model that puts at the center the notion of process; namely, the implementation of a given function is based only on a set of actions made of instructions, provided by the programming language at hand, possibly in combination with previously defined functions [31]. Top-down functional decomposition builds a system by stepwise refinement, starting with the definition of its abstract function. Each refinement step decreases the abstraction of the specification. With reference to Figure 1, the approach disregards objects/data, just considered as data structures that are instrumental to the function specification and internal to processes. Actions are defined only in terms of the instructions provided by the programming language and of other functions built on top of them (subroutines), into which a process is structured. All in all, this approach is intuitive and suitable to the development of individual *algorithms*, in turn aimed at solving some specific *task*, but does not scale up equally well when *data are shared among concurrent processes* because it lacks abstractions to explicitly account for such data and their corresponding management mechanisms.

Object-Orientation. The *Object-Oriented* approach to modularization results from an effort aimed at showing the limits of the functional approach [31]. Objects (data) often have a life on their own, independent from the processes that use them. Objects become, then, the fundamental notion of the model. They provide the actions by which (and only by which) it is possible to operate on them (*data operations*). This approach, however, disregards processes and their modularization both internally and externally to objects. Internally, because objects provide actions but have a *static nature*, and are inherently passive: actions are invoked on objects, but the decision of which operations to invoke so as to evolve such objects is taken by external processes. This also implies that there is no decoupling between the *use of an object* and the *management of that object*.

Externally, because the model does not supply conceptual notions for composing the actions provided by objects into processes, and there is no conceptual support to the specification of tasks, in particular when concurrency is involved.

Actor Model, Active Objects. The key concept in the *actor model* [28] (to which *active objects* are largely inspired) is that *everything is an actor*. Interaction between actors occurs only through *direct asynchronous message passing*, with no restriction on the order in which messages are received. An actor is a computational entity that, in response to an incoming message, can: (1) send a finite number of messages to other actors; (2) create a finite number of new actors; (3) designate the behavior to be used in response to the next incoming message. These three steps can be executed in any order, possibly in parallel. Recipients of messages are identified by opaque addresses. Interestingly, in [28] Hewitt et al. state that "We use the ACTOR metaphor to emphasize the inseparability of control and data flow in our model. Data structures, functions, semaphores, monitors, [...] and data bases can all be shown to be special cases of actors. All of the above are objects with certain useful modes of behavior." The actor model *decouples* the sender of a message from the communications sent, and this makes it possible to tackle asynchronous communication and to define control structures as patterns of passing messages.

Many authors, such as [32, 44, 35], noted that the actor model does not address the issue of *coordination*. Coordination requires the possibility for an actor to have expectations on another actor's behavior, but the mere asynchronous message passing gives no means to foresee how a message receiver will behave. For example, in the object-paradigm methods return the computed results to their callers. In the actor model this is not granted because this simple pattern requires the exchange of two messages; however, no way for specifying patterns of message exchanges between actors is provided. The lack of such mechanisms hinders the verification of properties of a system of interacting actors. Similar problems are well-known also in the area that studies enterprise application integration [1] and service-oriented computing [43], that can be considered as heirs of the actor model and where once again interaction relies on asynchronous message passing. There are in the literature proposals to overcome these limits. For instance for what concerns the actor model. [35] proposes to use Scribble protocols and their relation to finite state machines for specification and runtime verification of actor interactions. Instead, in the case of service-oriented approaches, there are proposals of languages that allow capturing complex business processes as service compositions, either in the form of orchestrations (e.g. BPEL) or of choreographies (e.g. WS-CDL).

The above problem can better be understood by referring to Meyer's forces. The actor model supports the realization of object/data management processes (these are the internal behaviors of the actors, that rule how the actor evolves), but it does not support the design and the modularization of processes that perform the object use, which would be *external* to the actors. As a consequence, generalizing what [15] states about service-oriented approaches, the modularization supplied by the actor model, while favoring component reuse, does not

address the need of connecting the data to the organizational processes: data remains hidden inside systems.

Business Processes. Business processes have been increasingly adopted by enterprises and organizations to conceptually describe their dynamics, and those of the socio-technical systems they live in. Modern enterprises [14] are complex, distributed, and aleatory systems: complex and distributed because they involve offices, activities, actors, resources, often heterogeneous and geographically distributed; aleatory because they are affected by unpredictable events like new laws, market trends, but also resignations, incidents, and so on. In this light, *business processes* help to create an explicit representation of how an enterprise works towards the accomplishments of its tasks and goals. More specifically, a business process describes how a set of interrelated activities can lead to a precise and measurable result (a product or a service) in response to an *external event* (e.g., a new order) [47]. Business processes developed for understanding how an enterprise work can then be refined and used as the basis for developing software systems that the enterprise will adopt to concretely support the execution of its procedures [14, 25]. In this light, business processes become *workflows* that connect and coordinate different people, offices, organizations, and software in a compound flow of execution [1]. Among the main advantages of this process-centric view, the fact that it enables analysis of an enterprise functioning, it enables comparison of business processes, it enables the study of compliance to norms (e.g. [27]), and also to identify critical points like bottlenecks by way of simulations (e.g., see iGrafx Process[3] for Six Sigma). The adoption of a service-oriented approach and of web services helps implementing workflows that span across multiple organizations, whose infrastructures may well be heterogeneous and little integrated [1, 43].

On the negative side, business processes, by being an expression of the process force, show the same limits of the functional decomposition approach. Specifically, they are typically represented in an activity-centric way, i.e., by emphasizing which flows of activities are acceptable, without providing adequate abstractions to capture the data that are manipulated along such flows. Data are subsidiary to processes.

Artifact-centric Process Management. The *artifact-centric approach* [7, 20, 15] counterposes a data-centric vision to the activity-centric vision described above. *Artifacts* are concrete, identifiable, self-describing chunks of information, the basic building blocks by which business models and operations are described. They are business-relevant objects that are created and evolve as they pass through business operations. They include an *information model* of the data, and a *lifecycle model*, that contains the key states through which the data evolve, together with their transitions (triggered by the execution of corresponding tasks). A change to an artifact can trigger changes to other artifacts, possibly of a different type. The lifecycle model is not only used at runtime to track the evolution

[3] `http://www.igrafx.com/`.

of artifacts, but also at design time to understand who is responsible of which transitions.

On the negative side, like in the case of the actor model, business artifacts disregard the design and the modularization of those processes that operate on them. Moreover, verification problems are much harder to tackle than in the case where only the control-flow perspective is considered. In fact, the explicit presence of data, together with the possibility of incorporating new data from the external environment, makes these systems infinite-state in general [15].

4 Towards Reconciliation: Agents and the A&A meta-model

In [40, 49], *agents* are defined as entities that observe their environment and act upon it so as to achieve their own goals. Two fundamental characteristics of agents are *autonomy* and *situatedness*. Agents are autonomous in the sense that they have a sense-plan-act deliberative cycle, which gives them control of their internal state and behavior; autonomy, in turn, implies proactivity, i.e., the ability of an agent to take action towards the achievement of its (delegated) objectives, without being solicited to do so. Agents are situated because they can sense, perceive, and manipulate the environment in which operate. The environment could be physical or virtual, and is understood by agents in terms of (relevant) data. From a programming perspective, it is natural to compare agents to objects. Agent-oriented programming was introduced by Shoham as "a specialization of *object-oriented programming*" [41]. The difference between agents and static objects is clear. Citing Wooldridge [49, Section 2.2]: (1) objects do not have control over their own behavior[4], (2) objects do not exhibit flexibility in their behavior, and (3) in standard object models there is a single thread of control, while agents are inherently multi-threaded. Similar comments are reported also by other authors, like Jennings [29]. However, when comparing agents to actors, the behavioral dimension is not sufficient: [49, page 30] reduces the difference between agents and active objects, which encompass an own thread of control, to the fact that "active objects are essentially agents that do not necessarily have the ability to exhibit *flexible* autonomous behavior". In order to understand the difference between the agent paradigm and objects it is necessary to rely on both the abstractions introduced by the agent paradigm, that are that of agent and that of environment [48]. Such a dichotomy does not find correspondence in the other models and gives a first-class role to both Meyer's process and object force (see Figure 2). Processes realize algorithms aimed at achieving objectives, and this is exactly the gist of the agent abstraction and the rationale behind its proactivity: agents exploit their deliberative cycle (as control flow), possibly together with the key abstractions of belief, desire, and intention (as logic), so as to realize algorithms, i.e., processes, for acting

[4] This is summarized by the well-known motto "Objects do it for free; agents do it because they want it".

in their environment to pursue their goals[5]. Contrariwise, active objects and actors do not have goals nor purposes, even though their specification includes a process. As we said, they are a manifestation of the object force. In the agent paradigm the manifestation of the object force is the environment abstraction. The environment does not exhibit the kind of autonomy explained for agents even when its definition includes a process. Its being reactive rather than active makes the environment more similar to an actor whose behavior is triggered by the messages it receives, that are all served indistinctly.

Most of the research in multiagent systems typically focuses on the abstraction of agent only, completely abstracting away from the notion of environment. Proposals like [23, 48] overcome this limit by introducing first-class abstractions for the environment, to be captured alongside agents themselves. In particular, [48] states that "the environment is a first-class abstraction that provides the surrounding conditions for agents to exist and that mediates both the interaction among agents and the access to resources." This proposal brought to important evolutions like the A&A meta-model [36] and its implementation CArtAgO [38].

Since in the agent paradigm each agent is an independent locus of control, coordination means become essential towards regulating the overall behavior of the system. As it is well underlined in [29], the agent-based model allows to naturally tackle the issue of coordination by introducing the concepts of *interaction protocol* [18], and that of *norm* [26, 46]. These concepts are at the heart of the design of multiagent systems. The deliberative cycle of agents is affected by the norms and by the obligations these norms generate as a consequence of the agents' actions. Each agents is free to adapt its behavior to (local or coordination) changing conditions, e.g., by re-ranking its goals based on the context or by adopting new goals.

Institutions and organizations set the ground for coordination and cooperation among agents. Intuitively, an institution is an organizational structure for coordinating the activities of multiple interacting agents, that typically embodies some rules (norms) that govern participation and interaction. In general, an organization adds to this societal dimension a set of organizational goals, and powers to create institutional facts or to modify the norms and obligations of the normative system [8]. Agents, playing one or more roles, must accomplish the organizational goals respecting the norms. Institutions and organizations are, thus, a way to realize functional decomposition in an agent setting.

5 The Rise of the Action Force

Actions are the capabilities agents have to modify their environment. The process force is mapped onto a cycle in which the agent observes the world (updating its beliefs), deliberates which intentions to achieve, plans how to achieve them, and finally executes the plan [12]. Beliefs and intentions are those components of the process abstraction that create a bridge respectively towards the object/data

[5] Summarizing, objects "do it" for free because they are data, agents are processes and "do it" because it is functional to their objectives.

force (i.e., the environment) and the action force. Beliefs concern the environment. Intentions lead to action [49], meaning that if an agent has an intention, then the expectation is that it will make a reasonable attempt to achieve it. In this sense, intentions play a central role in the selectin and the execution of action. Consequently, instead of being subordinate to the process force the action force is put in relation to it by means of intentions. This is a difference with respect to functional decomposition, where actions are produced by refining a given goal through a top-down strategy.

A fundamental step towards raising the value of the action force is brought by *normative multiagent systems* [30, 9], which take inspiration from mechanisms that are typical of human communities, and have been widely studied in the research area on multiagent systems. According to [9] a normative multiagent system is: "a multiagent system together with normative systems in which agents on the one hand can decide whether to follow the explicitly represented norms, and on the other the normative systems specify how and in which extent the agents can modify the norms". Initially the focus was posed mainly on *regulative norms* that, through obligations, permissions, and prohibitions, specify the patterns of actions and interactions agents should adhere to, even though deviations can still occur and have to be properly considered [30]. More recently, regulative norms have been combined with *constitutive norms* [8, 17, 21], which support the creation of institutional realities by defining institutional actions that make sense only within the institutions they belong to. A typical example is that of "raising a hand", which counts as "make a bid" in the context of an auction. Institutional actions allow agents to operate within an institution. Citing [21], the impact on the agent's deliberative cycle is that agents can "reason about the social consequences of their actions". In this light, going back to Meyer's forces, if agents are abstractions for processes and environments for objects, then *norms* are abstractions of the *action force* (see Figure 2) because norms model actions and, thus, condition the way in which processes operate on objects. In fact, norms specify either institutional actions, or the conditions for the use of such actions, consequently regulating the acceptable behavior of the agents in a system. This view is also supported by the fact that norms concern "doing the right thing" rather than "doing what leads to a goal" [46].

6 Data and Norm Awareness

The difficulty of engineering multiagent systems lies in the fact that the environment includes a process but such process is typically not represented in a way that can be reasoned about. Not only the environment should be given in terms of a data information model, specifying the structure of the information, and a data lifecycle, specifying data state transitions, (*data awareness*) but the two should be explicitly represented and accessible to the agents in their deliberative cycle. To this aim, such an explicit representation should constitute a body of *norms*, which describe how data evolution is governed, allowing agents to reason about the consequences of their actions and to have expectations about

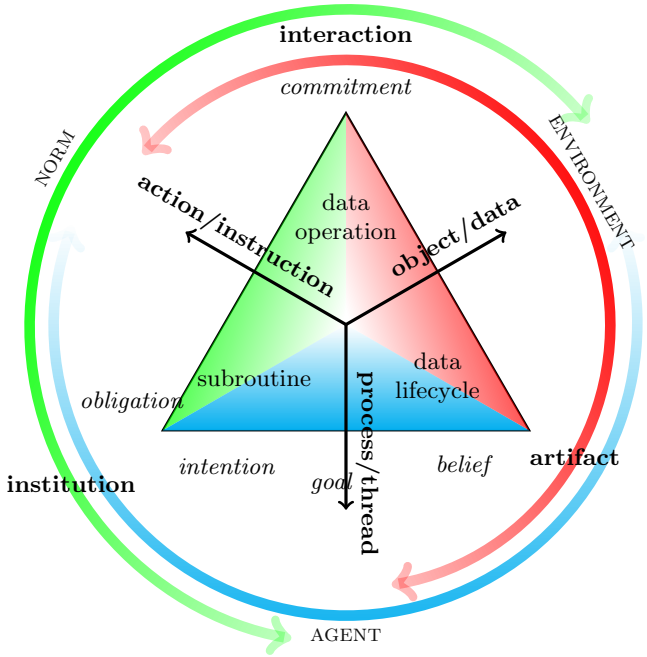

Fig. 2. Rereading Meyer's forces.

the evolution of the environment (*norm awareness*). Only a holistic, data- and norm-aware view would allow agents to reason and to have expectations on the evolution of the *whole* system, autonomously deciding the course of actions to apply.

Such a holistic solution where constitutive norms are used to specify both agent actions and data operations, and where regulative norms are used to create expectations on the overall evolution of the system (agents behavior and environment evolution) is, however, still missing. Object-Orientation associates operations to data; the set of executable operations can sometimes change along time depending on an object's lifecycle, but the paradigm did not push the study towards a normative representation. Similarly, while business artifacts provide both a rich description of their data and their lifecycle, they do not provide any link to a corresponding normative understanding, thus making impossible for the agents to leverage this knowledge for reasoning about how to act. Artifacts in the A&A model are radically different from the business artifacts because they do not come with an explicit information model for data, and they do not expose their lifecycle. Consequently, this lifecycle information cannot be exploited at design time, nor at runtime to reason about which actions should be taken towards the achievement of the agent goals.

A data- and norm-aware perspective would also bring advantages from a software engineering perspective, mainly residing in an increased decoupling among the agent system components, in a way that resembles what happens with business artifacts [20]. This is due to the fact that, at design time, norms would provide a programming interface between agents and their environment, given in terms of those state changes that are relevant in the environment.

6.1 A data-centric Approach to Interaction

A first step in the direction of having data and norm awareness is provided by the JaCaMo+ platform [3], which allows Jason agents [11] to engage commitment-based interactions [42], in turn reified as CArtAgO [37] artifacts. JaCaMo+ artifacts implement the social state of the interaction and provide the roles that are then enacted by the agents. The explicit representation of the social state enables the realization of a data-aware approach, where the data are the events occurring in the social state, while commitments provide the information necessary to agents in their interaction. Both agents and artifacts, encoding social states, are first-class elements in the design of the multiagent system. A commitment $C(x, y, s, u)$ captures that agent x (debtor) commits to agent y (creditor) to bring about the consequent condition u when the antecedent condition s holds. Antecedent and consequent conditions are conjunctions or disjunctions of events and commitments. Besides having an information model commitments have a lifecycle [45] that can be captured by a set of norms [22]. A commitment is *null* right before being created; *active* when it is created. Active has substates: *conditional* (as long as the antecedent condition did not occur), and *detached* (when the antecedent condition occurred, the debtor is engaged in the consequent condition of the commitment). An active commitment can become:

pending if suspended; *satisfied*, if the engagement is accomplished; *expired*, if it will not be necessary to accomplish the consequent condition; *terminated* if the commitment is canceled when conditional or released when active; and finally, *violated* when its antecedent has been satisfied, but its consequent will be forever false, or it is canceled when detached (the debtor will be considered liable for the violation). Commitments in JaCaMo+ belong to the social state and are shared by the interacting agents as resources. So, they are information, that is created and evolves along the interaction with event occurrence, and that contributes to the specification of the environment in which the agents operate. In this light, the social state can be seen as a special kind of business artifact in the sense of [7, 20, 15]. JaCaMo+ allows specifying agent programs as Jason plans, whose triggering events amount to the change of the state of some commitment [2]. Suppose, to make an example, that the commitment goes to the state "detached" and that this event triggers a plan in the agent which is the debtor of that commitment: the connection between the commitment and the associated plan is not only causal (event triggers plan), but rather the plan is explicitly attached to the commitment, in the sense that its aim is to satisfy the consequent condition of the commitment (norm-awareness).

While the representation of commitments in the JaCaMo+ platform is propositional, the Cupid language [19] provides a more sophisticate and information-centric representation that distinguishes between a schema (what occurs in a specification) and its instances (what transpires and is represented in a database), reserving the term commitment only for schemas. This avoids the inadequacy of first-order in representing commitment instances by relying on relational database queries. The advantages, brought to the analysis of properties, of a data-aware approach are proved in DACMAS [34], which incorporates commitment-based MASs but in a data-aware context. In general, in presence of data transition systems become typically infinite-state [15]. On the one hand, this is due to the fact that there is no bound on the number of tuples that can be added to database relations as the computation goes on. On the other hand, even when the number of tuples does not exceed a certain threshold, it is possible to populate them using infinitely many different data objects. Interestingly, when a DACMAS is state-bounded, i.e., the number of data that are simultaneously present at each moment in time is bounded, verification of rich temporal properties becomes decidable. Notably, this shows that, by suitably controlling how data are evolved in the system, it is possible to make agents data-aware without compromising their reasoning capabilities [6, 34].

7 Conclusion

Section 2 introduced properties that characterize quality software. Let us see how the rereading of Meyer's forces, that is depicted in Figure 2, impacts on the desired qualities of software. *Robustness* is the ability to react appropriately to abnormal conditions. The view of the action force as captured by norms allows agents to reason on the lifecycle of data in the environment, thus adding to the

already available capability of reasoning about deviations from agent's expected behavior, an enhanced capability of reasoning about abnormal conditions in the environment and decide how to react to them. So, in principle, the robustness of the system should be increased. The fact that data structure and lifecycles are explicitly represented in a way that can be reasoned about makes agents and their environment more decoupled, avoiding the need of customizing agent programs depending on the environment. This, in turn, increases both *extendibility* and *reusability* of all the components of the MAS. Last but not the least, data-awareness joint with a norm-based representation both enables a fully fledged range of verifications and helps modularizing the verification of properties inside a MAS, thus enhancing the *correctness* quality. In particular, if norms allow both for the specification of the environment and for the specification of action, it becomes possible to perform the analysis of properties at the level of norms rather than on the system as a whole. For instance, given a coordination artifact, it will be possible to verify deadlock freedom on the norms that it encodes and that represent it. The outcome will hold for any instance of the artifact that will be created. Of course, for each use it will be necessary to check that the usage of the artifact, done by a specific agent, conforms to the specification but this is a much simpler kind of verification [4]. A language for representing norms that guarantees a priori the decidability of property analysis would be a great advancement being the tool that agents need to reason and decide which action to take, thus leveraging their autonomy. JaCaMo [10], simpAL [39], JaCaMo+ [2] are existing platforms for the development of MAS that have the right potential for developing the view depicted in Figure 2. The next step would be the introduction of information-centric artifacts, whose lifecycle and data evolution are realized by way of query languages that, as for DACMAS [34], guarantee decidability when certain constraints are met. For commitment-based platforms, the Cupid [19] language would provide analogous features.

Concerning agent-based design, many proposals are found in the literature on Agent-Oriented Software Engineering, where agents are used as high-level software components that are characterized by autonomy and high-level communication, that is based on speech acts. Briefly, SODA [33] is an agent-oriented methodology for the analysis and design of agent-based systems, adopting a layering principle and a tabular representation. It focuses on inter-agent issues, like the engineering of societies and environment for MAS, and relies on a meta-model that includes both agents and artifacts. GAIA [50] is a methodology for developing a MAS as an organization. Tropos [13] is a requirements-driven methodology for developing multiagent systems. The 2CL Methodology [5] is an extension of [24]. It supports the design of commitment-based business protocols that include temporal constraints, and allows the verification of properties. New methodologies, however, are needed to tackle the norm-oriented and data-aware vision that we have illustrated. CoSE [2] is a commitment-driven methodology for programming agents.

Finally, [39] explores agent-oriented programming as a general purpose programming paradigm. It compares agent-based programming to actor-based pro-

gramming from a qualitative perspective, to explain the maturation process that lead to the development of the simpAL programming language. The simpAL languages is grounded on the concepts of agent, artifact, and workspace. A simpAL program is an organization where agents play roles. A static typing mechanism is provided to enact compile-time verifications concerning the implementation and interaction of agents and artifacts. [16] compares Jason, as a representative of agent programming language, against Erlang and Scala, that are two actor-oriented programming languages, in a communication benchmark, in order to verify if actor languages have better performances. The reported quantitative results (which concern time, memory and core usage), show that despite the fact that agent programming languages require a significant overhead when used to develop complex agents, Jason has reasonable performance.

Acknowledgements. The authors would like to thank the anonymous reviewers for the helpful comments. This work was developed during the sabbatical year that Matteo Baldoni and Cristina Baroglio spent at the Free University of Bolzano-Bozen. It was partially supported by the *Accountable Trustworthy Organizations and Systems (AThOS)* project, funded by Università degli Studi di Torino and Compagnia di San Paolo (CSP 2014).

References

1. Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services*. Springer, 2004.
2. Matteo Baldoni, Cristina Baroglio, Federico Capuzzimati, and Roberto Micalizio. Empowering agent coordination with social engagement. In Marco Gavanelli, Evelina Lamma, and Fabrizio Riguzzi, editors, *AI*IA 2015, Advances in Artificial Intelligence - XIVth International Conference of the Italian Association for Artificial Intelligence, Ferrara, Italy, September 23-25, 2015, Proceedings*, volume 9336 of *Lecture Notes in Computer Science*, pages 89–101. Springer, 2015.
3. Matteo Baldoni, Cristina Baroglio, Federico Capuzzimati, and Roberto Micalizio. Leveraging Commitments and Goals in Agent Interaction. In D. Ancona, M. Maratea, and V. Mascardi, editors, *Proc. of XXX Italian Conference on Computational Logic, CILC*, 2015.
4. Matteo Baldoni, Cristina Baroglio, Amit K. Chopra, Nirmit Desai, Viviana Patti, and Munindar P. Singh. Choice, Interoperability, and Conformance in Interaction Protocols and Service Choreographies. In K. Decker, J. Sichman, C. Sierra, and C. Castelfranchi, editors, *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2009*, pages 843–850, Budapest, Hungary, May 2009. IFAAMAS.
5. Matteo Baldoni, Cristina Baroglio, Elisa Marengo, Viviana Patti, and Federico Capuzzimati. Engineering commitment-based business protocols with the 2CL methodology. *JAAMAS*, 28(4):519–557, 2014.
6. Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. A computationally-grounded semantics for artifact-centric systems and abstraction results. In Toby Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 738–743. IJCAI/AAAI, 2011.

7. Kamal Bhattacharya, Nathan S. Caswell, Santhosh Kumaran, Anil Nigam, and Frederick Y. Wu. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Systems Journal*, 46(4):703–721, 2007.
8. Guido Boella and Leendert W. N. van der Torre. Regulative and constitutive norms in normative multiagent systems. In Didier Dubois, Christopher A. Welty, and Mary-Anne Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, June 2-5, 2004*, pages 255–266. AAAI Press, 2004.
9. Guido Boella, Leendert W. N. van der Torre, and Harko Verhagen. Introduction to normative multiagent systems. In Guido Boella, Leendert W. N. van der Torre, and Harko Verhagen, editors, *Normative Multi-agent Systems, 18.03. - 23.03.2007*, volume 07122 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.
10. Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, 78(6):747 – 761, 2013.
11. Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, 2007.
12. Michael E. Bratman. What is intention? In P. Cohen, J. Morgan, and M. Pollack, editors, *Intensions in Communication*, pages 15–31. MIT Press, Cambridge, MA, 1990.
13. Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.
14. David M. Bridgeland and Ron Zahavi. *Business Modeling: A Practical Guide to Realizing Business Value*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.
15. Diego Calvanese, Giuseppe De Giacomo, and Marco Montali. Foundations of data-aware process analysis: a database theory perspective. In Richard Hull and Wenfei Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 1–12. ACM, 2013.
16. Rafael C. Cardoso, Jomi Fred Hübner, and Rafael H. Bordini. Benchmarking communication in actor- and agent-based languages. In Massimo Cossentino, Amal El Fallah-Seghrouchni, and Michael Winikoff, editors, *Engineering Multi-Agent Systems - First International Workshop, EMAS 2013, St. Paul, MN, USA, May 6-7, 2013, Revised Selected Papers*, volume 8245 of *Lecture Notes in Computer Science*, pages 58–77. Springer, 2013.
17. Amit K. Chopra and Munindar P. Singh. Constitutive interoperability. In *Proceedings of the 7th Int. J. Conf. on Autonomous agents and multiagent systems, Volume 2*, pages 797–804. International Foundation for Autonomous Agents and Multiagent Systems, 2008.
18. Amit K. Chopra and Munindar P. Singh. Agent communication. In Gerhard Weiss, editor, *Multiagent Systems, 2nd edition*. MIT Press, 2013.
19. Amit K. Chopra and Munindar P. Singh. Cupid: Commitments in relational algebra. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 2052–2059. AAAI Press, 2015.
20. David Cohn and Hull Richard. Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009.

21. Natalia Criado, Estefania Argente, Pablo Noriega, and Vicent Botti. Reasoning about constitutive norms in bdi agents. *Logic Journal of IGPL*, 2013.
22. Mehdi Dastani, Leendert van der Torre, and Neil Yorke-Smith. Commitments and interaction norms in organisations. *J. Autonomous Agents and Multiagent Systems*, pages 1–43, 2015.
23. Yves Demazeau. From interactions to collective behaviour in agent-based systems. In *Proceedings of the 1st. European Conference on Cognitive Science*, pages 117–132, Saint-Malo, 1995.
24. Nirmit Desai, Amit K. Chopra, and Munindar P. Singh. Amoeba: A methodology for modeling and evolving cross-organizational business processes. *ACM Trans. Softw. Eng. Methodol.*, 19(2), 2009.
25. Antonio Di Leva and Salvatore Femiano. The bp-m* methodology for process analysis in the health sector. *Intelligent Information Management*, 3(2):56–63, 2011.
26. Jack P. Gibbs. Norms: The problem of definition and classification. *American Journal of Sociology*, 70(5):586—594, 1965.
27. Guido Governatori. Law, logic and business processes. In *Third International Workshop on Requirements Engineering and Law, RELAW 2010, Sydney, NSW, Australia, September 28, 2010*, pages 1–10. IEEE, 2010.
28. Carl Hewitt, Peter Bishop, and Richard Steiger. A universal modular ACTOR formalism for artificial intelligence. In Nils J. Nilsson, editor, *Proceedings of the 3rd International Joint Conference on Artificial Intelligence. Standford, CA, August 1973*, pages 235–245. William Kaufmann, 1973.
29. Nicholas R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.
30. Andrew J.I. Jones and José Carmo. Deontic logic and contrary-to-duties. In Dov Gabbay, editor, *Handbook of Philosophical Logic*, page 203–279. Kluwer, 2001.
31. Bertrand Meyer. *Object-oriented Software Construction (2Nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.
32. John C. Mitchell. *Concepts in programming languages*. Cambridge University Press, Cambridge, New York (N. Y.), 2002.
33. Ambra Molesini, Andrea Omicini, Enrico Denti, and Alessandro Ricci. SODA: A roadmap to artefacts. In *Engineering Societies in the Agents World VI*, volume 3963 of *LNAI*, pages 49–62. Springer, 2006. 6th Int. Workshop (ESAW 2005).
34. Marco Montali, Diego Calvanese, and Giuseppe De Giacomo. Verification of data-aware commitment-based multiagent system. In Ana L. C. Bazzan, Michael N. Huhns, Alessio Lomuscio, and Paul Scerri, editors, *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*, pages 157–164. IFAAMAS/ACM, 2014.
35. Rumyana Neykova and Nobuko Yoshida. Multiparty Session Actors. In eva Kühn and Rosario Pugliese, editors, *Coordination Models and Languages - 16th IFIP WG 6.1 International Conference, COORDINATION 2014, Held as Part of the 9th International Federated Conferences on Distributed Computing Techniques, DisCoTec 2014, Berlin, Germany, June 3-5, 2014, Proceedings*, volume 8459 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2014.
36. Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, December 2008. Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems.
37. Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the A&A meta-model for multi-agent systems. *JAAMAS*, 17(3):432–456, 2008.

38. Alessandro Ricci, Michele Piunti, and Mirko Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192, 2011.
39. Alessandro Ricci and Andrea Santi. From Actors and Concurrent Objects to Agent-Oriented Programming in simpAL. In Gul A. Agha, Atsushi Igarashi, Naoki Kobayashi, Hidehiko Masuhara, Satoshi Matsuoka, Etsuya Shibayama, and Kenjiro Taura, editors, *Concurrent Objects and Beyond - Papers dedicated to Akinori Yonezawa on the Occasion of His 65th Birthday*, volume 8665 of *Lecture Notes in Computer Science*, pages 408–445. Springer, 2014.
40. Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.
41. Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, March 1993.
42. Munindar P. Singh. An ontology for commitments in multiagent systems. *Artif. Intell. Law*, 7(1):97–113, 1999.
43. Munindar P. Singh and Michael N. Huhns. *Service-oriented computing - semantics, processes, agents*. Wiley, 2005.
44. Samira Tasharofi, Peter Dinges, and Ralph E. Johnson. Why Do Scala Developers Mix the Actor Model with Other Concurrency Models? In *Proceedings of the 27th European Conference on Object-Oriented Programming*, ECOOP'13, pages 302–326, Berlin, Heidelberg, 2013. Springer-Verlag.
45. Pankaj R. Telang, Munindar P. Singh, and Neil Yorke-Smith. Relating Goal and Commitment Semantics. In *Post-proc. of ProMAS*, volume 7217 of *LNCS*. Springer, 2011.
46. Göran Therborn. Back to norms! on the scope and dynamics of norms and normative action. *Current Sociology*, 50:863–880, 2002.
47. Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.
48. Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *JAAMAS*, 14(1):5–30, 2007.
49. Michael J. Wooldridge. *Introduction to multiagent systems, 2nd edition*. Wiley, 2009.
50. Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Trans. Softw. Eng. Methodol.*, 12(3):317–370, 2003.