1 Introduction

The emergence of Agent-Oriented Software Engineering (AOSE) has been claimed as the obvious evolution of current software theory and practice **?**. AOSE provides a new paradigm for developing robust, network aware, adaptive, sophisticated, and more efficient software solutions. This comes at an obvious price in increased complexity. For e.g., the notion that an agent is autonomous makes it complicated to analyze its behavior. Aspects like mobility and communication pose challenges whose solutions often involve multiple disciplines of computing as well as ideas from philosophy, ethics, biology, etc. The research community has responded with a range of frameworks, architectures, languages, and tools to deal with the analysis and development of multi-agent systems (MAS). These are often categorized by their level of abstraction, sophistication, and features. Our contention is that the available theories and tools, specifically those that have a formal foundation, are inadequate for several reasons including: inability to address all aspects of MAS in a single framework, a disconnect between theory and practice, ease of use, etc. Each aspect has several sub-issues, and there is a growing need for a critical analysis of this field that will help in developing more mature frameworks and tools. This paper is a step in this direction. We first discuss the ideal characteristics a formal framework should possess followed by a review of available frameworks and related architectures, languages and tools. The Belief-Desire-Intention model of cognitive agents in particular, has been popular for both modeling and implementing agent behavior and we discuss its core features. Based on the above we perform a critical analysis of the state of the art and derive elements of motivation for a new framework for multi-agent systems. The second half of the paper discusses the new framework in brief, addressing the issues raised in the critique.

2 Background

Some background issues related to agents and formal languages and frameworks are presented in this section which will be relevant to the review in the next section.

2.1 Formal Methods for AOSE

The use of formal methods in software engineering has been often been claimed as an integral part of overall software development. Although not adopted widely besides where the demand is on mission-critical and significantly complex software, the need for bringing formal methods as a standard component of the software development workflow is obvious: every mature field of engineering relies heavily on its mathematical foundation **?**.

Formal frameworks provide a mathematical foundation over which robust software can be built that is guaranteed to work properly upon deployment. Although a lot depends on how precise and accurate the requirements are from which the formal specification is built, once such a specification is available it can be confidently used for analyzing actual system behavior. This analysis could either be somewhat informal—such as graphical simulation—or verification by theorem proving, including proof search and model checking techniques. If the designer has intimate knowledge of the system's real-life environment, performance evaluation and other secondary analyses are possible with the right detail of specification.

With the advent of Multi-Agent Systems (MAS), software engineering as a field has taken a leap forward. AOSE is considered the next step in the field's evolution and has been adopted to great success in various commercial software solutions (**???**). The strengths of AOSE are also the source of its complexity: Agent autonomy, proactive behavior, a dynamic environment and the irregular nature of system composition, all contribute to a non-deterministic behavior that cannot be predicted in its entirety during development. This becomes harder when all we have at hand are ad-hoc techniques. Due to the unavailability of a mature and usable formal framework, most development of agent-based systems *has* been the result of an ad-hoc process: Usually there is no formal or mathematical foundation at work that can assure the developer or the user with a high degree of confidence that the system will indeed "behave how it's supposed to." It has often been argued that a wide-scale adoption of AOSE can only come about with the availability of high-level and sophisticated but easy to use specification, analysis, and development tools **??**. In summary, formal system specification and analysis are crucial for furthering the cause of adopting AOSE and developing robust agent-based systems.

2.2 Characteristics of An Ideal Formal Framework for AOSE

Due to the high degree of complexity and sophistication inherent in MAS, any formal framework that is used with the motivation outlined in section 2.1 must fulfill certain core criteria. What follows is a list of such criteria that we deem crucial. It *is* a tall order and in practice can often lead to problems of competing requirements; this will become more apparent in section 4.1 when we look at how the current research in the field measures up to these criteria.

2.2.1 Constructs for MAS Specification

The obvious and probably most important criterion is the ability to faithfully model the components of a practical MAS. MAS require advanced modeling tools with proper level of abstraction to natively model concepts such as agents, grouping, and mobility. It is a common observation that with the emergence of a new paradigm, there is a strong effort towards adapting old techniques and tools to the new paradigm. However, the more complex and revolutionary the new paradigm is, the stronger the chances that such an adoption will fail. Older tools that are used for specifying much simpler software systems can no longer

fully satisfy the needs for abstraction that AOSE demands. For example, although it is possible to specify any type of system in a low-level formalism such as π calculus (it is Turing-complete), such an attempt would invariably lead to an excessively complex model for MAS, too difficult to analyze with the absence of sufficient high-level abstractions.

2.2.2 Operational Semantics

System specification is half the story: proper operational semantics are required that will evaluate the temporal progress of a specification, i.e., demonstrate how a specified system will behave over time. This allows the system designer to analyze the behavior of the system. Of course, the level of detail at which this analysis is done depends upon the detail of the specification. Availability of a complete semantics of a formal language serves at least two purposes. Using a faithful specification, a virtual simulation of the progress and evolution of the system (textual or graphical) can be observed: this in itself can be of tremendous benefit for system analysis. Also, the semantics serve as the foundation for more well-grounded analysis including verification, discussed next. A surprising number of the available tools for MAS are developed without any formal semantics.

2.2.3 Verification

As the ultimate functionality of a formal framework, it should either directly support verification of the specified systems, or expose an interface for a supplementary verification techniques to be plugged in such as model checking. Even techniques that are less sophisticated such as equivalence checking (structural or behavioral) can prove to be of great benefit. In MAS, verification has usually been dealt with using model checking **???** due to the popularity of the logic-based languages for BDI, some process-calculaic and other approaches have been proposed as well **??**. Verification of MAS is in an early though active phase of research though it has been gaining momentum in the last decade.

2.2.4 Comprehensive Modeling Capability

A formal framework that can be used for reasoning about practical agent-based systems should ideally provide for the specification and analysis of all the relevant aspects of agent-based systems. These aspects include, the behavior of an individual agent—its proactive and reactive nature, mobility, autonomy; and the behavior of the overall agent system—inter-agent communication, organization and reorganization of agent composition, and security.

Although it is a difficult task to provide for a faithful specification of all these aspects under a single framework—the complexity is tremendous and the combination often leads to competing goals—it is necessary that if a formal framework aims to be used as a foundation for practical implementation, the specification itself should be complete. However, the nature of research in this complex field makes this necessity a rarity. Section 4.1 covers the incomplete nature of available frameworks in some detail.

2.2.5 Feasibility of Translating Theory to Practice

Although there is a place for formal tools that are used solely to analyze properties of MAS and to reason about their behavior, in the domain that we are concerned with, there should be a direct link from theory to implementable systems. For example, the concepts and abstractions employed in theory should have a equivalent mapping to tangible objects in practice. Also, the semantics of a formal language should be computationally tractable so that it can possibly be implemented as an interpreter or compiler. Ideally the semantics should have a direct correspondence to a high level language which can alleviate the complexity of composing a system specification in the more mathematical syntax of the formalism. Availability of a visual development environment based on the language would greatly enhance the chances of adoption by the industry.

2.3 The Belief-Desire-Intention Model

The Belief-Desire-Intention (BDI) behavioral model of cognitive agents has been popular both among researchers and the industry in the MAS field. It has been inspired by the work of the philosopher Bratman **?** in the field of reasoning theory and later developed by Rao and Georgeff **?** who gave a set of logics describing the BDI framework for agents. Part of its success is the simplicity with which it models agent behavior. An agent is seen as having "mental attributes" and "mental states" which map directly to human reasoning and behavior and therefore aids the designer or developer in specifying agent behavior. Another advantage of taking a BDI approach is that agents are reactive in real-time and therefore better suited in developing mission critical agent-based systems **?**.

The *beliefs* of an agent represent its knowledge about itself and its environment. *Desires* (usually undistinguished from *goals*) constitute the agent's motivations and guide its proactive behavior. The set of desires state what an agent is attempting to achieve and is a symbol of its autonomy. *Plans* are a collection of procedural actions or steps that if successfully concluded will lead to the fulfillment of particular goals. *Intentions* are those plans that are currently active, in pursuit of some goals.

Besides the logics introduced by Rao and Georgeff and later developed by other researchers that primarily explore the theoretical aspects of BDI agents, several languages and frameworks have been introduced for building practical BDI-based agent systems. They often do not share the terminology and features of the initial logic-based theory and some of them may provide semantics, if any, in a different system such as Z.

More details about BDI will be explored in the later sections.

3 Survey of Formal Frameworks for MAS

Given the importance of formal methods for MAS, the research community has taken up the initiative on several fronts and adopted different approaches while doing so. In broad terms, these can be divided into two somewhat mutually-exclusive categories covered in this section. We will not deal with frameworks and languages without a real formal foundation; for more details on these tools refer to **?** and **?**.

3.1 Formalisms for Organizational Specification

MAS tend to have a complex structure and composition: The number of agents, their dynamic creation and destruction, the evolving nature of inter-agent communication structure, possible mobility, and issues of coordination and cooperation between agents, all contribute to this complexity. Most of these issues cannot be resolved or predicted at design-time since the agents are blessed with autonomy, and *emergence* comes into picture. Traditionally, dynamic system structure has been studied best using formalisms that support concurrent computation such as Petri-Nets, Actor model, and process calculi. Process calculi that extend pi-calculus, with a well-grounded foundation based on communication, have been particularly popular for describing systems such as MAS with concurrent computing elements. Some efforts include: Ambient calculus, CLAIM, API-calculus, Seal calculus, and Nepi **????**. Ambient calculus concentrates on representing the hierarchical organization and mobility of agents in *milieus* and has been successful in attracting research to extend the basic calculus with security, communication, and typing. CLAIM is one of the more recent efforts: inspired by ambient calculus it provides support for modeling the behavioral aspects of agents as well (we will discuss CLAIM further in 4.1). API-calculus extends π calculus with notions of grouping, mobility, and a basic reasoning model. Seal calculus is a family of sophisticated calculi based on π calculus with the primary goal of specifying agent mobility in all its details **??** . Nepi is a low-level programming language that extends π-calculus with some basic constructs, such as data types and more sophisticated communication abilities.

Although formalisms in this class are suitable in modeling organization, composition, and communication aspects of MAS, for the most part they do not deal with reasoning, behavior, and planning. Specifying the behavior of the autonomous agents requires a framework for intelligence or behavioral representation such as the BDI model. The formalisms of the present class usually lack high-level constructs and semantics required for this purpose.

3.2 Formalisms for Behavioral Specification

Blessing software agents with autonomy gives them the ability to reason and exhibit independent behavior. This translates into some level of *intelligence* in an agent's behavior. Modeling this intelligent behavior has been the goal of this class of formal methods. Those that have gained prominence in this area have their foundation in logic, specifically multi-modal logic such as temporal and epistemic. The usual approach is to model agent behavior through a set of its cognitive skills. This includes the seminal work done by Rao and Georgoff **??** on modal logics for the Belief-Desire-Intention (BDI) model, which has been extended in several directions (**?????????**) and has also been used for practical agent frameworks with remarkable success **????**. Other notable efforts in behavioral specification using logic include Wooldrige's **?** and Fagin et al.'s **?** individual work on modal logic for reasoning. Besides BDI-based research, there are some other formalisms that deal with agent behavior. Vip, based on ψ calculus **??**, provides a process calculus-based semantics of plan execution systems, with a finer detail than the regular BDI model. CAN, part of the Prometheus methodology, provides a comprehensive model for behavior clearly inspired by BDI **?**. The SLABS framework is worth mentioning also **?**. CLAIM (mentioned in the last sub-section) also provides basic behavioral modeling based on an event-model **????**. Other examples can be referred to in **??**, although none of them have gained prominence as a definitive model.

3.3 Other Formalisms

There are few other formal methods for MAS that are not easily classified under any of the above classes. This includes the use of older software engineering specification languages such as ETL **?** and Z **?** for MAS specification. Certain practical agent architectures provide some level of formal specification, such as INGENIAS **?** and PASSI **?**. Also, there are several efforts for verification of agent communication **??** that form an important subset considering the complexity of communication in MAS.

4 Critique

4.1 Issues with current Formalisms

Although we briefly touched on where the available formalisms fall short of the ideal requirements in the previous section, in the following this topic is explored in more detail. We also discuss some possible reasons as to why there is a dearth of an ideal formalism and mention some efforts towards this in current research.

4.1.1 Incomplete Specification Capability

The major issue with the available frameworks is that the majority focus on modeling certain specific aspects of MAS: reasoning and behavior, mobility, computation, communication, or organization and composition. The formalisms listed in section 3 are individually capable of modeling certain specific aspects, but they either do not consider other aspects or they allow only a high-level, abstract view of them. For example, in the traditional BDI model, there are several aspects of agency and MAS that

are not comprehensively dealt with but were addressed in individual efforts later. As Ancona et al. state in **?**: "most related works for extending BDI are for logical formalization of the theory behind the extended BDI model rather than for complete specification of an implementable system." The original BDI model was mainly concerned with reasoning about the behavior of an individual agent and did not explicitly deal with the *situatedness* of an agent in a multiple-agent environment (later theoretical efforts such as **?** provide some useful extensions). Similarly, *learning*, an important topic in agent research, does not find any mention in the traditional BDI model (see **?** for an extension). Also, as addressed in **?**, the popular logic-based BDI formalisms do not allow detailed specification of agent computations such as *plan execution*, which are eventually left for the implementation stage—aspects which in fact can be very important when analyzing agent behavior.

In a broad sense, most formal frameworks either deal with the behavior of the system (usually of an individual agent), or its dynamic organization and composition. However, when dealing with the formal analysis of actual systems, a more comprehensive outlook is needed.

4.1.2 Disconnect between theory and practice

The major motivation behind developing a new formal framework is the unavailability of a framework that will lead system designers from formal specification and analysis to system implementation and deployment. There is a definite lack of capable frameworks satisfying this requirement. As Luck observes in this older but surprisingly relevant survey of the field **?**, "... much work has tended to focus on either the development of practical applications of agent systems on the one hand, or the development of sophisticated logics for reasoning about agent systems on the other." Specifically in the field of BDI research, the development of theory doesn't have a one-to-one correspondence with practical BDI frameworks: it is usually just the higher-level, abstract concepts that are in common. Most efforts at building a bridge deal with providing semantics of existing practical frameworks in formalisms such as Z. These include the work on providing semantics for AgentSpeak(L) (**????**), dMARS (**?**), and 3APL (**?**). Such sporadic efforts appear more as an afterthought and do not result in a final, seamless and comprehensive framework. CLAIM, although not in the BDI domain, attempts to fill in this gap and give a comprehensive modeling framework **?**. It provides a high-level language used for system specification that can be translated to a lower-level language—with formal semantics—and can also be interpreted for generating Java-based skeletal implementations. However, the actual strategy for translation to the lower-level language is not explained beyond stating that the two syntax have one-to-one correspondence. Also, the behavioral model is very basic and lacks features present in BDI-based frameworks. Another effort is SPARK **?** which provides a framework to develop real-world agent-based systems and provides a formal semantics for its specification language. However, the environment of an agent is not explored beyond representation as the "outside world".

A formal specification system for MAS satisfies this property if it has a system of formal semantics that can be easily interpreted and implemented (see section 2.2). Pure syntactical efforts that only allow specification, although useful in their own way, have different motivations. In the words of Morley et al., "an ideal agent system should combine the sophisticated representations and control of the more practical systems with principled semantics that enable reasoning about system behavior." **?**

4.1.3 Complex Semantics

Related to the previous issue, complexity of semantics is a major deterrent for associating formal frameworks to a practical development workflow. This complexity of formal methods can be mitigated by providing practical high-level tools that are based on the theory but allow system designers to ignore the theoretical details. Indeed, one of the signs of maturity and adoption of any new technology is the gradual increase in the number of high-level tools that allow non-experts to use the technology without extensive knowledge of the underlying dynamics.

Most theoretical BDI frameworks have been proposed using multi-modal logic which although well suited for specifying the cognitive properties of agents, usually tend to be very complicated. They also assume that agents have near-infinite computational power, which is hardly the reality for most efficient agent systems. This has resulted in very few efforts at providing its semantics, and even fewer for building a robust verification theory (see **?**).

4.2 Issues specific to BDI related formalisms

Besides the general issues covered above, some issues specific to BDI-based formal frameworks are covered below.

4.2.1 System Organization and Composition

As mentioned earlier, the motivation behind the BDI model is to be able to specify agent behavior and not the composition of a whole agent system. Therefore ideas such as grouping, hierarchy, and mobility are not dealt with in the traditional model. The only work that we are aware of in this regard is the extension done for the TOMAS architecture **?** which is built over the Java-based BDIM toolkit **?**, although it lacks any formal foundation.

4.2.2 Multiple agents support

In traditional BDI theory, the stress is on developing a behavioral theory for a single agent. Issues related to the agent's "situatedness" in a neighborhood that includes other agents is not addressed explicitly. For example, the dynamics of communication between agents are not dealt with in any detail beyond the general idea of the occurrence of "events". More importantly—from the point of view of

behavior modeling—awareness of other capable agents allows an agent to search for external capabilities in situations where it cannot satisfy a goal by itself. Indeed, it is this cooperation that alleviates the excesses of autonomy and therefore needs to be expressed expertly in a behavioral theory like BDI. Coo-BDI gives a well-detailed extension to traditional BDI for cooperation **?** but it doesn't provide any formal specification.

4.2.3 Learning

The concept of *learning* is given major significance in traditional AI. However, the BDI model by itself does not provide a structured mechanism for agents to have learning capabilities. **?** addresses this issue of learning in a MAS by extending BDI, although it is done using the practical framework of dMARS **?**, and doesn't refer to any formalization of the concept.

4.2.4 Runtime Plan Construction and other Plan Execution Details

Most formal *and* practical BDI-based systems simply drop active goals if they cannot be accomplished by using the available plans. This is partly due to the fact that goals are not represented explicitly in such systems and are instead implicit in the occurrence of events; if an event cannot be responded to with an available event handler (a plan), it is instantly dropped. Such an ephemeral event does not represent the original concept of a persistent goal. One aspect of expressing a persistent goal is to allow agents the flexibility to construct a plan at runtime when the available plan library is not useful. This obviously enhances the intelligence and autonomy of the agent. An agent may also choose to construct or modify plans using the capabilities of other agents as discussed earlier. Planning is a complex topic—specially for autonomous agents. Refer to **???** for some efforts towards combining the mature planning techniques with the BDI model.

4.3 Motivation for a New Formal Methods

The above review and critique has motivated our research in building a new formal framework for multi-agent systems that specifically addresses these issues. The rest of the paper gives a brief introduction to the new formalism. Although motivated and inspired by the frameworks mentioned above, the new language has been developed independent of any existing formalism for foundation.

5 A New Formal Framework for Multi-Agent Systems

In the following we briefly present the new framework, named MAML (for *M*ulti *A*gent *M*odeling *L*anguage), specially in contrast with the issues discussed in the last section. MAML is both an agent framework based on an extension of core BDI concepts, and a formal language for expressing agent systems based on the framework. The framework extends BDI to address issues raised in last section while the language provides high-level constructs for MAS specification and process-calculus based semantics. The theme of this presentation is to give a feature-based overview of the framework and language in order to justify the motivation behind MAML's development. A more comprehensive discussion of the syntax and the semantics will be covered in an upcoming publication.

5.1 Constructs of the MAS Framework

The elements used for agent definition in MAML can be presented at two different, though related, levels: how these elements are viewed *conceptually* in the framework, and how these elements are constructed *syntactically* in the language.

Conceptually, the elements that constitute an agent in MAML can be broadly classified as one of the following two types:

- *Cognitive elements*: They capture an agent's proactive and reactive behavior and define its motivation, reasoning, and executive capabilities. These include *goals*, *plans*, *intentions*, and *capabilities*. Every agent in MAML must include a set of these cognitive elements so that it can exhibit proper behavior.

- *Informational elements*: They represent the information an agent has about itself and its environment and therefore differ from agent to agent. For example, a seller agent will include elements such as an inventory list, a list of buyer agents, an account balance, etc.

Syntactically, both above types of element are extensions of the same basic language construct, *belief*. Therefore, in MAML an agent's behavioral and informational elements have identical syntactic properties. For example, the same action that is used to add a new goal for the seller agent can also be used to add information about a new buyer to the buyer list. Furthermore for an agent, the collection of cognitive and informational belief elements *completely* defines the agent (see figure **??**). This notion of belief is somewhat different from the traditional view in BDI formalisms—inspired by the philosophical notion of knowledge—where an agent's behavioral elements are syntactically and conceptually different from its knowledge (the collection of all beliefs). The idea of a singular view of agent composition in MAML has several benefits and is explored in detail in section 5.2.2.

Agents themselves are composed in hierarchical groups called *clusters* which are the building blocks of MAS organization (figure **??**).

Before presenting some key aspects of MAML, we discuss the compositional details of MAML below.

Beliefs Beliefs are at the core of the framework. The basic belief structure provides the foundation for representing

an agent's information about the world, as well as for its behavioral elements.

A belief can be either non-decomposable (an *atom*) or compound (a *variable*) (figure **??**). The collection of all beliefs for an agent is referred to as its *knowledge*. Variables are composed of *items* which can be either beliefs or actions (actions themselves are not represented independently in *knowledge*). A belief can be associated with meta-data using *types*, which are themselves atomic beliefs. Types do not add expressiveness to the language by themselves, but do so in conjunction with the language semantics. Functionally, this meta-data is used for supporting several aspects of the framework: identifying belief owner in a multi-agent environment (i.e. if a belief does not belong to the local agent), asserting privacy control over individual beliefs, identifying state changes in beliefs such as goals and plans, and in general for working with categorized beliefs. Also, a set of reserved types is used to construct extended beliefs that constitute the agent's cognitive elements: its goals etc.

Actions Actions are the basic executive components of the language. The basic actions are related to either *belief manipulation* or *agent mobility*; compound actions based on these are used to express the executional details of plans, intentions, and capabilities. The basic actions are *assert a belief*, *remove a belief*, *query for a belief*, *conditional*, *looping*, and *movement to a specific cluster*. The *add* action also has a multi-agent version with the qualification that a belief asserted to a remote agent will be syntactically identified with the asserting agent. Besides passing information, the *remote* add action is used to trigger events in other agents. This event triggering is the basis of communication and reactive agent behavior and is explored in detail later in the section. Actions can be composed sequentially or concurrently and can occur as elements of a variable (such as *plans*, *capabilities*, and *intentions*).

Goal Goals specify the proactive behavior of agents at the highest level. Syntactically, goal is a derived belief, i.e. it is a variable associated with a specific type, "γ", which identifies it as a goal construct. Goals in MAML include both the declarative and the procedural aspects: the declarative aspect is a set of beliefs, called *achieve belief*, that must be part of the knowledge if the goal is to be achieved; and the *goal plan* represents the procedural aspect which defines the strategy to be used for goal achievement [1]. Including the declarative aspect in the goal specification allows for reasoning about agent behavior in detail while the procedural aspect makes the steps for goal achievement explicit (the relevance of including both these aspects is also discussed in **?**). Other items that constitute a goal in MAML are: an *abort* condition, and a *failure plan* to be invoked in case of goal abortion.

[1] this is another difference between MAML and other BDI-based frameworks: rather than goals being implicit in the occurrence of events, they are explicitly represented in MAML.

Plan A plan defines a strategy to achieve a goal. The executive item of a plan is its *intention*, which is variable whose elements are a set of actions. Besides the plan intention, the specification includes *context beliefs* which provide the condition without which the plan will not be initiated, *achieve beliefs* which advertise to plan-less goals what the plan can achieve, *abort beliefs* that state under what condition the plan will be aborted, and *failure intention* which is a set of actions to be executed when the plan fails for any reason.

Capability A capability is a composition of actions and can be related to a function or subroutine in traditional programming. One of the main motivations for having a capability construct along with plans is that capability is the only high-level construct that can be shared between agents. Also, capability has a notion of being triggered, i.e., the event when its actions start executing. The elements of a capability are: *conditions*, a set of beliefs that provide the context, parameter, or trigger for the capability; *effects*, which specify the beliefs that the successful execution of the intention will bring about; and *actions* which is a composition of actions that are executed in order when the capability is triggered.

Cluster A cluster is a group of agents that can be viewed as a collection based on some shared criteria, either physical or functional. For example, in a distributed system, each machine can be seen as a single cluster and clusters in a single network can be seen as another cluster. Similarly, agents can be combined based on their common functionality; agents sharing similar set of beliefs, or similar goals can belong to a common cluster. An agent can give other agents in its own cluster special preference regarding belief sharing, specially capabilities. Gaining access to these preferences is the primary motivation for agent mobility across clusters. Also, an agent can send broadcast messages to other agents in a cluster which simplifies communication amongst homogeneous agents.

5.2 Key Aspects of the Framework

In this section we cover some of the key features of the MAML framework. These features reflect the motivation behind its development and also the areas where it may offer benefits over other approaches. This section also serves to present how MAML's language and framework measure up to the critique of formal MAS frameworks in section 4.

5.2.1 BDI orientation

BDI has emerged as a popular and well-developed theoretical foundation for reasoning about agent behavior. Although it does have certain shortcomings, specially when considering non-behavioral aspects of agents, it is a mature theory for representing proactive behavior. The core concept of BDI—representing agent behavior with cognitive

constructs—is used in MAML. Agents are seen as primarily proactive: they have goals that they pursue, plans that are used for goal achievement, and intentions that represent the current activity that the agent is committed to act upon.

Most of the issues mentioned earlier regarding the BDI-related formalisms have been addressed in MAML. In many ways MAML adopts the philosophy of BDI and gains from the experience of previous efforts in the field. Most significantly, agent proactivity and reactivity is considered explicitly in context of a multiple agent system. One issue that MAML does not deal with currently is a direct representation of agent learning. However, with the extensible nature of the belief construct and the availability of belief manipulation actions, agent learning can be incorporated into the framework by extending the core language.

A major reason for developing MAML from the ground up, rather than as an extension of traditional BDI, is that although BDI serves a major purpose in MAML, it is not intended to be the core concept. Rather the BDI extension is a component of the underlying agent theory, along with the organizational and compositional aspects.

5.2.2 Belief programming

A key idea in MAML is the explicit and detailed representation of beliefs. The set of agent beliefs holds all information that an agent "knows" about. All information is represented as beliefs and is referred together as the agent's *knowledge*. Even executing actions don't store any information "inline"— for example, any new variable an action may set up is entered as a belief into the agent's knowledge. Incoming information from remote agents is asserted as beliefs in the knowledge as well.

The notion of belief and knowledge in MAML is somewhat different from that of epistemic logic— often used in logic-based BDI systems—where belief is more formally connected to its philosophical connotations. The use of beliefs in MAML is less restrictive; beliefs are representative of every aspect of an agent including its knowledge. Therefore, an agent's goals, plans and capabilities are viewed as forms of beliefs (i.e., the agent must *know* what its goals, plans, and capabilities are) and are therefore derived syntactically from the basic form of belief (figure **??**) . As mentioned earlier, this has the advantage that an agent can add, remove, and modify its goals, plans, etc. with the same basic actions that it uses for manipulating its regular information. This simplified view is at the core of MAML's philosophy and drives almost every aspect of the semantics.

An important aspect of beliefs is their use as implicit *events*. Unlike most traditional BDI frameworks where events are explicitly represented as such, in MAML raising an event for an agent is simply the assertion or removal of beliefs. It is the occurrence of these implicit events that controls an agent's reactive and proactive behavior. Therefore, beliefs serve as the foundation for representing the agent behavior.

MAML deals with beliefs for most purposes as a hierarchical collection of elements not very unlike XML documents. Besides the use of conditions in conditional actions, there is no other use of logic in MAML's belief representation. The logic-based view, as used in some other formalism, allows for much more extensive reasoning of knowledge, though at the cost of complexity and often unrealistic execution semantics as discussed earlier. On the other hand, MAML allows for a more detailed view of beliefs—at least syntactically—which can be used for sophisticated reasoning with use of the belief manipulation actions.

Individual and compound beliefs in MAML can be "typed" with metadata. As mentioned earlier, typing is the foundation of several other features of the language. For example, a type can be used for the purpose of classification of beliefs. It is this classification that allows the extension of basic beliefs. This classification is also used by an agent to exert access control over the beliefs in its knowledge: they can be made accessible to just the agent itself (local access), or to a group of agents.

5.2.3 Comprehensive Modeling of MAS

The central premise in MAML is to present a single formal framework for modeling every relevant aspect of MAS is the most significant motivation behind its development. Below we discuss in brief how MAML addresses the different aspects of MAS.

Organization Agents in MAML are organized in groups called *clusters*, which themselves can be hierarchically composed within each other. Cluster provide the following functions:

- *Functional and logical grouping*: Quite often, a collection of agents is seen as a collection exhibiting some functional or conceptual commonality. In MAML, these agents may be identified as a cluster which makes identification, communication, cooperation, and other aspects easier to model at the group level. Also, hierarchical grouping can provide a more sophisticated grouping opportunity for system designers than a flat, set-based grouping.

- *Access control*: In support of grouping, an agent can limit access to some of its beliefs to just those agents present in its parent cluster, or a collection of clusters.

- *Mobility*: An agent can migrate from its current cluster to any other. Proactively, the motivation for mobility may come from the access control that agents impose on their beliefs. For example, consider two agents A and B which operate in different clusters. If A wishes to use agent B's capability bQ, which can only be accessed by agents in the same cluster as B, then A would need to migrate to B's parent cluster to gain access to aQ.

Communication Remote belief manipulation provides a simple means for communication between agents. Communication in MAML has the following different aspects:

- *Relaying information*: Asserting a new belief in another agent's knowledge may be done solely for the sake of passing information. A belief asserted in such a manner will automatically be typed with the asserting agent's name for identification and security purposes.
- *Events*: As mentioned earlier, communication between agents may lead to implicit events in the recipient agent: asserting a belief in an agent's knowledge can trigger one of the receiving agent's capability that would act as an event handler.

Agent behavior Agent behavior in MAML is inspired by the basic BDI model of agent cognition.

- *Proactive behavior*: The *goals* of an agent provide the motivation for proactive behavior. Goals specify what an agent wishes to achieve and can optionally include an assigned plan which is committed towards that achievement. The agent's "wish" defines the *declarative* part, while the plan defines the *operative* part of goal statement. Having both aspects allows a richer goal definition leading to a distinct separation between goal and plan progress and better goal reasoning. *Plans* define the path to achieving a goal by using *capabilities* and sub-goals. Of course there may be several plans that may possibly achieve a goal, and it is the choice of the designer at design-time or the runtime plan construction procedure at runtime, to assign a particular plan to a goal. A capability includes a sequential or parallel set of actions and an action is the most basic executive component of agent behavior.
- *Reactive Behavior*: An agent behaves reactively by processing events: a belief assertion (executed locally or by another agent) can trigger a capability which acts as the event handler.
- *Runtime Plan Construction*: A plan may be missing from the goal if it was not assigned one during the initial agent specification, or if the assigned plan fails during execution. The first step in case of a missing plan is to match what the goal's desired state with what an available plan can achieve; if a matching plan is found then it is assigned to the goal. Otherwise the runtime construction procedure is initiated. The procedure is somewhat involved and is attempted as the last step in achieving a goal. It involves a two step process: first, the available capabilities are evaluated as to whether they can play a role in goal achievement at any stage, starting from the final stage that the goal wishes to achieve to the current state of the system. The capabilities evaluated may include those belonging to remote agents. Second, the relevant capabilities are composed together in a sequential or parallel fashion to produce a plan. In case there are stages in the goal achievement for which there are no available capabilities, they are delegated as sub-goals with the intention that plans or capabilities for achieving them may be available in future states of the agent. Of course the plan constructed in this manner may be more prone to failure and the procedure itself may be inefficient, but it is presented as a last resort towards goal achievement.

5.2.4 Theory and Practice

The formal MAML specification of a system can be easily translated to a functionally analogous version implemented in a high-level language. Towards this end the development of MAML is geared towards making the language accessible for practical purposes besides formal analysis. *Actions* in MAML, specially those related to belief programming, can be seen as simple methods of a high-level language, while a group of simple methods can be seen as a capability. All belief constructs can have a parallel implementation in an object-oriented language. As the first step towards developing a practical model for MAML, we are currently working on an interpreter of the semantics that would translate the formal specification into an XML-based description. Eventually, this would then be used to drive a UML-based tool to generate a skeletal system in an object-oriented language like Java.

Our goal is towards a platform for developing MAS based on the formal MAML specification but with the assistance of graphical tools for composition and observation of system progress.

5.3 Informal Semantics

Although a detailed operational semantics of the language is not possible in the length of the current paper, the following gives an overview of the semantics with the help of an example. The intention behind this section is to provide an overview of the language and how it can be used to model a simple system. Majority of the syntax used in this section mirrors the original athough simplification is made wherever necessary.

An Illustrative Multi-Agent System A multi-agent system representing a simple market spread over two cities is used as the example (figure **??**). The market consists of two kinds of agent: *CustomerAgent* and *SellerAgent*. There are two cities in the system modeled as clusters *CityA* and *CityB*. Each city has a single seller agent, SA and SB respectively. There are multiple customer agents in each city: CA_1, CA_2, CA_3 in *CityA*, and CB_1, CB_2, and CB_3 in *CityB*. The *CustomerAgents* share a similar basic composition, which can be written in its skeletal form

as

$$
\begin{aligned}
&CA, B_{1,2,3} \\
&\quad :: \texttt{Goals} :: finishShopping[] : \gamma, status \\
&\qquad \texttt{Plans} :: shopLocal[] : \pi, status; \\
&\qquad\qquad shopRemote[] : \pi, status \\
&\texttt{Capabilities} :: pay[] : \phi \\
&\qquad\qquad \texttt{BB} :: balanceAmount[], shoppingList[]
\end{aligned}
$$

A customer has a single goal, typed γ that directs its proactive behavior to accomplish its shopping; two plans, typed π that will attempt to buy the next item on the shopping list either locally and in another city respectively; and a capability, typed ϕ that can be used to pay for an item. Both goals and plans have a type *status*, which indicates its current state as one of those in figure **??**. The above are part of the agent's knowledge along with at least two other belief variables: one that stores the balance left to shop with and the other a list of items to shop for.

Similarly, *SellerAgents* share the composition:

$$
\begin{aligned}
SA, SB :: \texttt{Goals} :: &keepInventoryStocked[] : \gamma, status \\
\texttt{Plans} :: &inventoryUpdate[] : \pi, status \\
\texttt{Capabilities} :: &handleRequests[] : \phi, \\
&sellItem[] : \phi, cluster \\
\texttt{BB} :: &itemsSelling[], balanceAmount[], \ldots
\end{aligned}
$$

A seller has a single goal for keeping its inventory stocked (therefore it differs from the *finishShopping* goal in that it is geared towards maintenance rather than achievement); a single plan that is employed by this goal; two capabilities to handle queries for whether an item is available for sale and to sell an item; and at least two other beliefs: a list of items available for sale, and the balance amount. The type *cluster* associated with *handleRequests[]* restricts triggering of this capability by only the agents in the same cluster as the seller agent. Therefore, only customers in the same city as the seller can handle sale requests.

We now consider some aspects of MAML that can be illustrated by the above system's progress. This is not intended to be comprehensive but rather it is provided in order to give the idea behind the framework's philosophy.

Initiation of proactive behavior Consider the customer agent CA_1 in cluster A. Like other customer agents, it has the *finishShopping* goal which drives its proactive behavior. It's core structure is:

$$
\begin{aligned}
finishShopping ::&AB[shoppingList[\emptyset]] \\
&AbB[cancelShopping] \\
&P_g[shopLocal[]] \\
&P_f[\emptyset] \\
&: \gamma
\end{aligned}
$$

Therefore the goal will be considered achieved when the *shoppingList* variable does not have any elements, i.e. the shopping is complete. Initially this variable has the structure:

$$shoppingList[milk[2], eggs[12], bread[1]]$$

with each element of the variable being an item on the list, and each element has a numeric atom that holds the quantity desired. The goal will abort on the condition that a belief, *cancelShopping*, is asserted in the knowledge. However, the goal does not have a failure plan that may be activated in case of an abortion. It has an assigned plan *shopLocal* (which has its own abort condition) that will attempt to achieve this goal. The lifecycle of a goal is shown in figure **??**. Transitions in the lifecycle are identified by the *status* type. For example, initially the goal is inactive, and will appear in the knowledge as $finishShopping[] : \gamma, inactv$. Since the abort conditions don't hold, its P_g is initiated, and both the plan and the goal become *active* (figure **??** and **??**).

Multiple plans are expected to be available for a goal to choose from. A plan may be assigned to a goal at design time if there is enough information available. If a goal does not have an assigned plan at execution stage, a plan is searched for—plans are matched to a goal based on whether their *achieve beliefs* are in concordance.

The plan *shopLocal* has the following composition:

$$
\begin{aligned}
shopLocal ::&AB[shoppingList[\emptyset]] \\
&CB[shoppingList[]] \\
&AbB[noLocalSeller] \\
&I[shopLocalI[]] \\
&I_f[+(shopRemoteOnly)] \\
&:\phi
\end{aligned}
$$

The *achieve belief* (AB) is the same as the parent goal—for a plan to be accepted by a goal it is necessary that $AB(Goal) \subset AB(Plan)$. The *context belief* ($CB$) requires a belief variable *shoppingList* to be available before the plan becomes active. The plan aborts (AbB) in case there is no local seller available. After activation, the plan starts executing actions listed in the intention *shopLocalI* and if the plan aborts it will assert a belief *shopRemoteOnly* that will qualify future shopping attempts.

Remote communication and Events The first action of intention *shopLocalI* is $+(lookingFor[shoppingList[1]]; SA)$ which asserts the first item in shopping list to the seller agent SA's knowledge as an element of a new variable *lookingFor*. The asserted belief includes the asserting agent's name as a special type, for e.g. $lookingFor[milk, 2]; CA_1$, that will be required for identification later. In case CA_1 could not identify the seller agent by name, it could do a broadcast to all agents in the cluster with the action $+(lookingFor[shoppingList[1]]; thisCluster)$ which will assert the belief to all agents in the current cluster; if there

were multiple seller agents they could be intimated by the $+(lookingFor[shoppingList[1]]; SA_1, SA_2, \ldots)$ action (figure **??**).

Assertion of a belief in a local or remote knowledge is equivalent to raising a possible event if there is a capability that will be asserted by the belief assertion. The *handleRequests* capability in S_a has the composition

$$\begin{aligned} handleRequests ::& C[lookingFor[]] \\ & E[itemsSelling[]] \\ & Acts[\ldots] \end{aligned}$$

and since the only condition (C) is the presence of a belief *lookingFor*, the capability is triggered at this time. This implies the execution of the sequence of actions in the capability's *Acts* which is given in a simple form as:

$$\begin{aligned} Acts ::& \texttt{if}(?(lookingFor[1] \in itemsSelling[] \\ & \quad \texttt{AND}\ lookingFor[2] \leq itemsSelling[lookingFor[2]]) \\ & \texttt{then}\ +(itemAvailable[lookingFor[1]]; \\ & \quad ?_{agent}(lookingFor[])) \\ & \texttt{else}\ +(itemAvailable[\emptyset]; ?_{agent}(lookingFor[]))) \end{aligned}$$

The above sequence will check whether the item queried for is available and if the quantity is adequate; if so an *itemAvailable* variable is asserted with the item's name in agent CA_1's knowledge, otherwise an empty variable is asserted ($?_{agent}$ queries for the asserting agent's name).

Assuming that *SA* responds with a non-empty *itemAvailable* assertion, CA_1 will continue its shopping procedure from *SA*. The intention *shopLocalI*'s next set of actions will check the *itemAvailable* variable for a positive response from the seller agent. If so it will trigger the agent's own *pay*[] capability to complete the transaction. After successful purchase of the first item, the plan will iterate through the rest of the elements in the *shoppingList* variable. If *shopLocalI* exhausts its actions, then the plan would attain the "achieved" state and thus finish successfully. At this point ideally the goal will also transition into the "achieved" state, but this transition is not automatic after plan achievement: in fact the goal will be "achieved" if and only if its *AB* holds true. Therefore after plan exhaustion, there is a final check made to ensure that the goal's condition hold true by itself.

Plan and Goal Failure If CA_1 receives an empty *itemAvailable* variable from *SA*, *shopLocalI* recognizes the failure of the current plan and performs the action $+(noLocalSeller)$ which results in the abortion[2] of the current plan since its *abort belief* holds true now; the plan state changes to "abort". With the abortion the failure intention, I_f asserts $+(shopRemoteOnly)$ which will aid the adoption of the secondary plan. Due to the separate definition and existence of plans and goals, the failure of a plan does not automatically lead to the corresponding goal's failure. The *finishShopping* goal has the opportunity to search for an alternate matching plan. The second plan *shopRemote* has the structure

$$\begin{aligned} shopRemote ::& AB[shoppingList[\emptyset]] \\ & CB[shoppingList[], shopRemoteOnly] \\ & AbB[noRemoteSeller] \\ & I[shopRemoteI[]] \\ & I_f[\emptyset] \\ & :\phi \end{aligned}$$

Since its *achieve belief* matches that of the goal, and its *context beliefs* hold true it is adopted by the goal as its P_g. The working of the new plan will be covered later, but a few points regarding goal failure are relevant here: in case the *shopRemote* failure was not available, the goal would have aborted. Goal abortion changes the goal state to "abort"; in this state the failure plan P_f, if any, is adopted to perform any necessary clean-up. After completion of P_f, unlike plans which end their execution at this point, goal returns to its "inactive" state until it can find another plan to adopt. Of course, besides abortion due to non-availability of a plan, the goal can also abort if its *abort belief* hold true at any point. The same holds true regarding its transition to the "achieved" state: it can do so at any point in the progress of the system if its *achieve beliefs* hold true. Which bring us to an important property of goal in this framework: goal achievement may or may not be a result of its P_g entering the "achieved" state and goal failure may not be an (indirect) result of its P_g failure.

Inter-cluster interaction and mobility Continuing from the last section, if the *shopRemote* plan is adopted, its *shopRemoteI* intention starts execution. We will consider the actions concisely: in order to query remote agents, the first action needs to assert the *lookingFor* belief in remote clusters. There are three cluster-realted assertion actions: $+(b; thisCluster)$, which asserts to all agents in the current cluster to which the agent belongs; $+(b; upClusters\{\})$, which recursively asserts to agents in clusters that are organizationally superior to the current cluster; and $+(b; downClusters\{\})$, which recursively asserts to agents in clusters inferior to the current cluster. Since the only remote cluster that can be queried is a sibling to the current one, $+(lookingfor[shoppingList[1]]; upClusters\{\})$ is used. This asserts the variable to agents in *CityB* including *SB* (figure **??**). Since agent *SB* has a similar structure to *SA*, its *handleRequests* capability will be triggered by this event and will respond in a similar manner. If it has the requested item it will assert a positive response which will appear in CA_1 as $itemAvailable[someItem]; up[CityB\{SB\}]$; the belief is appended by a type that identifies agent *SB*'s position relative to CA_1's (figure **??**).

After receiveing a positive response, agent CA_1 will now send the request as earlier for sale by triggering *SB*'s *sellItem* capability. However, since this *sellItem* is accessible

[2] *abort* and *failure* are equivalent terms in the framework

only to agents in *CityB* (it is typed as *cluster* in the definition), CA_1 will receive a negative response to signify this. At this stage, CA_1 will move from its current location to *CityB* by the action $move(SB)$, i.e., move to the same cluster as SB which is equivalent to specifying the cluster name: $move(up\{CityB\})$. Doing this will allow CA_1 to buy the item from the remote seller (the situation becomes obviously more complex if there are other goals and intentions that are active and have allegiance to the current cluster).

Other aspects There are several other aspects that have been omitted in the above discussion. These include: typing of agents to provide finer-grained organization than the cluster level; issues of concurrency of goals, intentions, and capabilities; runtime plan construction for goals; details regarding the "formal" aspect of the language: detailed semantics, congruence (structural and behavioral), behavior of maintenance goals.

6 Conclusions and Future Work

Our current work is the first step towards developing a framework and language for the faithful representation and analysis of MAS. The framework has several benefits over other formal frameworks for MAS, and addresses the core issues with them as detailed in section 5.2. This paper is intended mainly as a justification for developing this framework, specially because there is not a dearth of frameworks and languages for MAS. We believe that not our work will not only fill a void in the available technologies related to developing MAS, but will also aid in exploring aspects of MAS concepts that would not have been possible earlier.

Future Work In continuation of the work presented in this paper, we are working on a mature platform based on the MAML framework for MAS development and analysis. The core of this platform will be a high-level declarative language (HMAML) for MAS specification that can be compiled to the low-level MAML syntax and also be interpreted to a high-level object-oriented language. The components of the proposed platform includes:

- a *graphical MAS specification engine* that will allow a high-level and efficient approach to MAS design. The MAML syntax is well suited for system specification, but using it for specifying the composition and behavior of large MAS will be a time consuming process. The specification engine will allow systems described in HMAML by compiling it to MAML. HMAML will include high-level constructs that will abstract details of the MAML syntax. Also, a graphical interface built over HMAML will allow an even more abstract view. The graphical interface will be specially useful for analyzing the system composition and communication structure at an initial stage of development. It will also give a useful overview of the system at a glance—something that is hard to accomplish with a textual description.
- an *analysis engine* that can be used to evaluate a system description passed on by the specification engine. The operational semantics of MAML can be employed for analysis using two complimentary processes:
  - *graphical analysis*: by using the graphical representation a temporal view of system behavior can be presented in an animated fashion. Such a dynamic and high-level, though informal, view of system progress can be invaluable in gaining insight about system characteristics.
  - *formal analysis*: more formal verification techniques such as structural/behavioral congruence, and model-checking can be built over the MAML syntax (currently, we have an automated mechanism for structural congruence checking only).
- a *MAS implementation and deployment environment* where the formal system specification is automatically translated to a skeletal definition in an object-oriented language such as Java. This definition can then be implemented in detail manually to produce a deployable agent-based system. We are working on a translation scheme to map agents and agent-based systems from MAML to the Java-based JADE platform for deployment.