Interfacing Belief-Desire-Intention Agent Systems with Geometric Reasoning for Robotics and Manufacturing

Lavindra de Silva[1], Felipe Meneguzzi[2], David Sanderson[1], Jack C. Chaplin[1], Otto J. Bakker[1], Nikolas Antzoulatos[1], and Svetan Ratchev[1]

[1]Institute for Advanced Manufacturing, Faculty of Engineering, University of Nottingham, UK
{firstname.lastname@nottingham.ac.uk}
[2]Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, RS, Brazil
{felipe.meneguzzi@pucrs.br}

Abstract. Unifying the symbolic and geometric representations and algorithms used in AI and robotics is an important challenge for both fields. We take a small step in this direction by presenting an interface between geometric reasoning and a popular class of agent systems, in a way that uses some of the agent's available constructs and semantics. We then describe how certain kinds of information can be extracted from the geometric model of the world and used in agent reasoning. We motivate our concepts and algorithms within the context of a real-world production system.

1 Introduction

Modern manufacturing environments require systems capable of dynamically adjusting to rapid changes in throughput, available production equipment, and end-product specifications. When there are complex and non-specialized machines or robots involved that are able to perform a multitude of tasks, intelligent and flexible systems are needed for modeling parts, the environment, and production processes, and for reasoning about how processes should manipulate parts in order to obtain the desired product. These systems typically reason in terms of concepts such as a machine's degrees of freedom, the positions and orientations of parts, and collision-free trajectories when moving parts during production. Such geometric reasoning is especially appealing in the context of manufacturing because detailed CAD models of parts and end-products are readily available, and production processes are often well defined. When a production system is controlled by a higher level software entity such as an agent system, however, which typically reasons in terms of abstract and symbolic representations that ignore the finer details present in the geometric model, it is crucial to be able to unify at least some aspects of the two representations so that they may be linked and information may be shared. Indeed, such a unified representation is also an important challenge for robotics and AI in general.

This paper focuses on interfacing a (single-agent) agent programming language, from the popular Belief-Desire-Intention (BDI) family of agents [1], with geometric reasoning in a way that exploits some of the agent's existing constructs and semantics.

adfa, p. 1, 2011.

We also give insights into the kinds of information that can be abstracted from the geometric model for the agent's benefit; this includes information about any new, previously unknown objects in the domain, and which objects are connected to each other and will therefore move together. Since BDI agent systems do not plan their actions before execution, but instead perform context-based expansion of predefined (user-supplied) plans during execution, our work differs from existing works such as [2, 3, 4, 5, 6, 7, 8] which focus on integrating symbolic *planners* with geometric reasoners. A notable exception is [9], who also interleave symbolic reasoning with acting as we do; however, they do not use a standard model of agency.

2 Background

Geometric Reasoning. In this paper we use the term *geometric reasoning* to refer to motion planning as defined in [10]. A *state*, then, is the 3D world $W = \mathbb{R}^3$, and its fixed obstacles are the subset $O \subset \mathbb{R}^3$. A robot is modelled as a collection of (possibly attached) rigid *bodies*. For example, a simple polygonal robot A could be defined as the sequence $A = (x_1, y_1, z_1), \dots, (x_n, y_n, z_n)$, where each $(x_i, y_i, z_i) \in \mathbb{R}^3$. A key component of motion planning is a *configuration space* C, which defines all the possible transformations that can be applied to a body such as A above. More specifically, a *pose* (or *configuration*) $c \in C$ is the tuple $c = \langle x, y, z, h \rangle$, where $(x, y, z) \in \mathbb{R}^3$ and h is the unit quaternion, i.e. a four dimensional vector used to perform 3D rotations; in a 2D world $W = \mathbb{R}^2$, h would instead be an angle in $[0,2\pi)$. With a slight abuse of notation the transformation of a body A by pose c is denoted $A(c)$. A robot's pose composed of bodies $A_1, \dots, A_n$ is an element of $C_1 \times \dots \times C_n$, where each C_i is the configuration space of A_i. If a body A_2 is attached via a joint to the end of some body A_1, some of A_2's degrees of freedom will be constrained, e.g. the x, y and z parameters of all poses of A_2 might depend on the corresponding ones in A_1.

A motion planning problem, then, is a tuple $\langle C, col, c_I, c_G \rangle$, where $col : C \rightarrow \{true, false\}$ is a function from poses to truth values indicating whether a pose $c \in C$ is in collision ($col(c) = true$) with some object or not, and $c_I, c_G \in C$ are the initial and goal poses [3]. A collision-free motion plan solving a motion planning problem is a sequence $\boldsymbol{c} = c_1, \dots, c_n$ such that $c_I = c_1, c_G = c_n$, and for each pose c_i, we have that $c_i \in C$ and $col(c_i) = false$.

BDI Agents. In this work we use the popular AgentSpeak [1] agent programming language to formally represent the large class of BDI agent systems in the literature. An AgentSpeak agent is a tuple $Ag = \langle E, B, Pl, I \rangle$ where: E, the event queue, is a set consisting of both external events (environment perception) and internal events (subgoals); B, the belief base, is a set of ground logical atoms; Pl, the plan library, is a set of plan-rules; and I, the intention stack, is a set of partially instantiated plan steps of plan-rules that were adopted. A plan-rule is a syntactic construct of the form $\langle e \rangle : \langle con \rangle \leftarrow \langle body \rangle$, where $\langle e \rangle$ is the triggering event; $\langle con \rangle$, the context condition, is a logical formula; and $\langle body \rangle$ is a sequence of steps to be executed in the environment. There are two types of triggering events relevant to this paper: $+\varphi$ or $-\varphi$ for an

atom φ indicates, respectively, that a belief in B has been added or removed, and $-!\,\psi$ or
$-?\,\psi$ indicates, respectively, that an achievement or test goal has failed, i.e. that either the plan to achieve ψ has failed during execution, or that belief ψ does not hold in B, respectively. Finally, $\langle body \rangle$ is constructed from the following elements: (i) the execution of an action in the environment; (ii) the adoption of a subgoal $!\,\psi$ or testing of a condition $?\,\psi$, both of which generate internal events; or (iii) the explicit modification of a belief ($+\varphi$ or $-\varphi$). An example of a plan-rule in our AgentSpeak-like language is the following:[1] $+!\,mov(R,F,T) : canMov(R,F,T) \leftarrow nav(R,F,T)\,;?\,at(R,T)$.

If the achievement goal $!\,mov(r1,t1,t2)$ is reached when some rule is executed, AgentSpeak looks up Pl for a rule that is both relevant and applicable for the goal. Our rule above is relevant because $mov(R,F,T)$ and $mov(r1,t1,t2)$ unify on the application of substitution $\theta = \{R/r1\,, F/t1\,, T/t2\}$ to the former; we use $mov(\boldsymbol{o})$ to denote the ground instance resulting from operation $mov(\boldsymbol{v})\theta$, where $\boldsymbol{v}$ and $\boldsymbol{o}$ are the vectors of variables and constants above. If the plan-rule is also applicable, i.e. belief $canMov(r1,t1,t2) \in B$, then the plan's body, after applying the substitution, is added to I as a new intention. Pursuing it involves executing action $nav(r1,t1,t2)$ and then checking for success by testing B via $?\,at(r1,t2)$. The action to navigate is defined by the following action-rule: $nav(R,F,T) : at(R,F) \wedge canMov(R,F,T) \leftarrow mvExec(R,F,T)\,;mvEff()$, where $mvExec(R,F,T)$ is associated with a procedure that moves the robot, and $mvEff()$ with one that returns, possibly after sensing the environment, a set of literals representing the result of moving.[2]

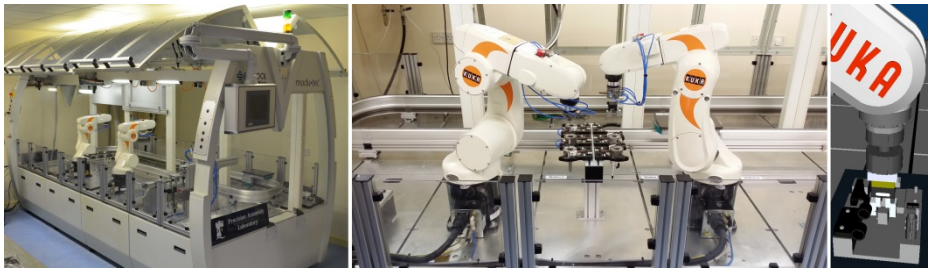

Fig. 1. The assembly platform, tool rack, and a simulation of the pallet being gripped

The Assembly Platform. We use the production system in Figure 1 [11] as a running example to motivate some of the concepts in this paper. The system combines the functionality of six independent workstations, each governed by a separate agent, to assemble detent hinges for lorry-cab furniture. Each station is served by a linear transfer system that transports a pallet carrier; this supports a pallet with the individual parts that need to be assembled, as well as the partially/fully assembled hinge. The six workstations, controlled by PLCs (Programmable Logic Controllers), are as follows:

[1] R is short for $Robot$, F for $From$, T for To and ti for $table\ i$

[2] An action-rule's body is adapted from STRIPS to be a sequence of functions that return a (possibly empty) set of literals, each of which is applied to the belief base B, i.e. the positive literals are added to B, and atoms associated with negative literals are removed from B.

two consist of a Kuka robot each; two accommodate one workspace each; one contains a tool changing rack; and one contains an inspection station. The tool changing rack is placed between the Kuka arms, which have access to the rack as well as to the workspaces that are used for carrying out assembly operations. The rack contains six slots which can hold up to six different types of end effectors such as pneumatic and two-finger grippers. RFID tags on the tools are used to determine which of them are currently on the rack, so that the Kuka arms may dynamically lock into the relevant ones during assembly. Finally, the inspection station is used to perform force and vision tests to verify whether the hinge was assembled correctly. The hinge that is assembled is composed of two separate leaves held together by a metal pin. Three metal balls need to be placed into adjacent cylindrical slots in the center of the hinge, three springs need to be placed into the same slots, and a retainer is used to close the hinge. By using only a subset of these parts to assemble a hinge, there can be four product variants, each having a different detent force.

3 Interfacing AgentSpeak with Geometric Reasoning

Like in works such as [2, 7], evaluable predicates are fundamental in linking AgentSpeak with geometric reasoning. While standard predicates are evaluated by looking up the agent's belief base, evaluable predicates are attached to external procedures, which for us involve searching for a viable trajectory within a geometric world/state W. Thus, we call such predicates *geometric predicates*. For example, predicate $canMov(R, curr, T)$ in our plan-rule from the previous section could be a geometric predicate which invokes a motion planner to check whether it is possible for Kuka arm R to move from its current pose $curr$ to tool T, specifically, to a position from where the arm can now easily lock into the tool with a predefined vertical motion. We use $curr$ as a special constant symbol to represent the current pose.

To evaluate a geometric predicate it needs to be associated with a collection of *goal poses*, from which at least one needs to have a viable trajectory from the current pose for the predicate to evaluate to $true$. Goal poses could either be determined manually or computed offline automatically with respect to the 3D model of the world and the objects involved. In our assembly platform, for example, the Kuka arms are manually trained on how to grasp the various shapes that might be encountered during production. This is especially important because objects like the pallet carrier are too heavy to be lifted from most seemingly good grasps and poses—there is only one pose that will work; indeed, a mere 3D model of the world that cannot also take into account additional information such as object weights will not be able to automatically predict such goal poses accurately. Consequently, we require that a "sampling" SMP from ground geometric predicates to their corresponding goal poses be provided by the user. For example, predicate $canMovGr(k1, gr1, curr, pc)$, which checks whether Kuka arm $k1$ combined with gripper $gr1$ can move to a pose from where pallet carrier pc can be grasped, will map to the set consisting of just the single pose depicted in Fig. 1.

We describe SMP as follows. Let $P = \{p_1(o_1, \ldots, o_j), \ldots, p_n(o'_1, \ldots, o'_k)\}$ be the set of ground instances of all geometric predicates occurring in the agent, and $P_s = \{p_1, \ldots, p_n\}$ and $O = \{o_1, \ldots, o_j, \ldots, o'_1, \ldots, o'_k\}$ their associated predicate and constant symbols, respectively. Then, if n_{max} is the maximum arity of a predicate in P, function SMP is denoted by the partial function SMP : $C \times P_s \times O_1 \times \ldots \times O_{n_{max}} \rightarrow 2^C$, where C is the configuration space and each $O_i = O$. Thus, function SMP is a user-defined "sampling" with only the goal poses that "matter" with respect to the current pose $c \in C$ and the given ground geometric predicate. In practice, the full goal pose for a task such as picking up an object could be computed dynamically from a user-supplied pose for the gripper—such as the one in Fig. 1—by first transforming the gripper's pose to "place" it relative to the object and within the current world W, and then using inverse kinematics to derive suitable poses for the geometric bodies that form the robot arm, which are attached to the gripper and to each other.

Function SMP is used within an "intermediate layer" like the ones used in [2, 3], which we actualise here via a special evaluable predicate denoted by INT : $P_s \times O_1 \times \ldots \times O_{n_{max}} \rightarrow \{true, false\}$, where P_s, n_{max} and each O_i are as before. For example, if $n_{max} = 4$ in the given domain, the agent developer might invoke the intermediate layer via function INT$(canMov, k1, curr, t1, null)$, where $null$ is a symbol reserved for unused parameters. Function INT$(canMov, k1, curr, t1, null)$ is defined as follows. Suppose c_I is the current pose of the robot, and that SOL ("solution") and FCT ("facts") are global variables initialised to the empty sequence and empty set, respectively. Then, if there is a pose $c_G \in$ SMP$(c_I, p, o_1, \ldots, o_{n_{max}})$, and a collision-free motion plan from c_I to c_G, we first assign the motion plan to SOL and then return $true$, and otherwise we assign the set of facts describing why there was no trajectory—specifically the obstruction(s) that were involved—to FCT and return $false$. This approach keeps trajectories and poses transparent to the agent developer.

4 Encapsulating Geometric Reasoning within AgentSpeak

AgentSpeak-like languages offer some useful, built-in mechanisms that allow a clean embedding of motion planning. In particular, we can encapsulate each geometric predicate $p(\boldsymbol{v})$ occurring in the agent within a unique achievement goal $!\,e_p(\boldsymbol{v})$ via the plan-rules and action-rules shown below. Specifically, we first associate the achievement goal with the two plan-rules in the left-hand column below:

$+!\,e_p(\boldsymbol{v}) : true \leftarrow actSucc_p(\boldsymbol{v})$	$actSucc_p(\boldsymbol{v}) : \text{INT}(p, \boldsymbol{v}) \leftarrow exec(\,);\, post(\,);\, \Phi^{\top}$
$-!\,e_p(\boldsymbol{v}) : true \leftarrow actFail_p(\boldsymbol{v})$	$actFail_p(\boldsymbol{v}) : \neg\text{INT}(p, \boldsymbol{v}) \leftarrow post(\,);\, \Phi^{\bot}$

Since the bottom plan-rule handles a goal-deletion event, it is only triggered if the top plan-rule fails, i.e. if the precondition of the ground action $actSucc_p(\boldsymbol{o})$, which involves motion planning, is not applicable. Moreover, as per the semantics of goal-deletion events, once the bottom rule finishes executing, the associated achievement goal $!\,e_p(\boldsymbol{o})$ will still fail. These are the semantics we desire in order to, before failing, compute and include the beliefs/facts relating to why the failure occurred. Sets $\Phi^{\top}$ and $\Phi^{\bot}$ are predefined beliefs denoting any "predictable" changes resulting from the

achievement goal's execution; for example, geometric predicate $canMovGr(K, Gr, curr, PC)$ might have $\Phi^{\top} = \{r\}$ and $\Phi^{\bot} = \{\neg r\}$, with $r = reachable(K, Gr, PC)$ (i.e. pallet carrier PC is reachable to arm K with gripper Gr).

The second step in our encapsulation is defined in the right-hand column above by the action-rules associated with actions $actSucc_p$ and $actFail_p$.[3] In our definition, $post(\,)$ is a function that returns the set of (symbolic) facts representing either the pose that resulted from executing $exec(\,)$, or the "reasons" why there was no trajectory while evaluating the precondition, i.e. the set FCT computed by INT$(p, \boldsymbol{o})$. Likewise, $exec(\,)$ is associated with a procedure that executes (in the real world) a given motion plan, which in our case is the one that was assigned to SOL when INT$(p, \boldsymbol{o})$ was called. Action $actFail_p(\boldsymbol{o})$ is not associated with any such function because its action-rule is only chosen when there is no viable motion plan. Thus, the rule's precondition confirms that ¬INT$(p, \boldsymbol{o})$ still holds, just in case there was a relevant change in the environment after INT$(p, \boldsymbol{o})$ was last checked, causing INT$(p, \boldsymbol{o})$ to now hold (in which case there are no failure-related facts to include).

We assume that $exec(\,)$ always succeeds, and that if necessary the programmer will check whether the action was actually successful by explicitly testing its desired goal condition. This is exemplified by the $!\,move(R, F, T)$ achievement goal in Section 2, where $?\,at(R, T)$ checks whether the $navigate(R, F, T)$ action was successful. One property of the described encapsulation is that looking for motion plans and then executing them and/or applying the associated symbolic facts are one atomic operation—no other step can be interleaved to occur between those steps. This ensures that a motion plan found while evaluating an action's precondition cannot be invalidated by an interleaved step while the action is being executed.

Once all geometric predicates have been encapsulated as described, we may then use their corresponding achievement goals from within plan-rules. Since we cannot include them in context conditions (logical formulae) they can instead be placed as the first steps of plan bodies. This allows such achievement goals to be ordered so that the ones having the most computationally expensive geometric predicates are checked only if the less expensive ones were already checked and they were met.

5 Symbolic Abstractions of Geometric Elements

There are certain elements in the geometric representation that are worth abstracting out into their corresponding symbolic entities so that they may be exploited by the agent. Our first abstraction is a user-defined surjection from a subset of the geometric bodies (defined as a sequence of boundary points, for example) onto a subset of the constant symbols occurring in the agent. This allows multiple bodies—such as the individual pieces of a Kuka arm—to simply be identified by a single constant symbol such as $k1$, but also for certain geometric bodies (e.g. an unknown box on the floor) and symbolic constants (e.g. the name of a customer) to be ignored. Indeed, while every rigid body is crucial for geometric reasoning, it does not necessarily need a

[3] For simplicity we omit the last parameters of INT$(p, \boldsymbol{v})$, which may be $null$ constants.

corresponding symbolic representation, and likewise, every constant symbol occurring in the agent does not necessarily represent a geometric body.

Our second abstraction is represented by logical literals, whose ground instances are obtained and applied via the function $post(\,)$. Formally, these literals are a consistent subset of $2^{P \cup \bar{P}}$, where $\bar{P} = \{\neg p \mid p \in P\}$ and P is the set of ground instances of predicates occurring in the agent, obtained by replacing each predicate's vector of n terms with an arbitrary vector of n constant symbols. Thus, while these literals will only mention predicate symbols that occur in the agent, they might mention constant symbols (objects) that do not occur in the agent. This leaves room for discovering new, previously unknown objects "on the fly". For instance, if the agent senses from one of its RFID readers that there is a new object on the tool rack, the agent might then look up the tag's associated globally unique electronic product code (EPC) on the web, recognise the object as a certain type of gripper, and assign it with the new symbol $gr7$. This might then become associated with new symbolic facts returned by $post(\,)$, such as $gripper(gr7)$, $inToolRack(gr7)$ and $near(gr7, gr1)$.

One useful domain-independent predicate inferable from the geometric representation concerns pairs of bodies that are "attached" to one another in the geometric model. For example, suppose that the vision test in the testing station builds a detailed 3D model of the partially assembled hinge on the pallet carrier, and then checks that it was assembled correctly. If this test fails because a part (e.g. one of the leaves) is absent in the partial hinge, facts such as $att(pc, leaf1)$ and $att(leaf1, retainer)$, indicating which pairs of parts are nonetheless successfully attached to each other in the partial hinge, will enable the agent to reason about which parts will move together when the pallet carrier is transferred onto the conveyer belt. Formally, a possible definition of $att(o, o')$ for two objects o, o' is the following (we use A, A' and $C^A, C^{A'}$ in C to respectively denote their bodies and configuration spaces): $att(o_1, o_2)$ holds if there is a $k \in \mathbb{R}$ and $m \in \{1, \ldots, 3\}$ such that for any two poses $(a_1, a_2, a_3, a_4) \in C^A$ and $(a'_1, a'_2, a'_3, a'_4) \in C^{A'}$, we have $a_m = a'_m + k$, i.e. at least one degree of freedom of one of the objects is constrained by the other. Other useful domain-independent predicates include $vol(o, v)$ and $coll(o, o')$, where the former is the volume v of object o calculated from its geometric representation, and the latter indicates that there is a pose in which o and o' (e.g. the two arms) will collide; formally, $coll(o, o')$ holds if there exist bodies A, A' associated respectively with objects o, o', and poses $c \in C^A$ and $c' \in C^{A'}$ such that $A(c) \cap A'(c') \neq \emptyset$, i.e. when A, A' are transformed and 'placed' into world W, at least one of their points overlap. Such a fact might eventuate in the agent taking precautions to ensure the tool rack is only used by one arm at a time.

There are also geometric elements that are too 'fine grained' to be modeled as symbolic elements, such as absolute x and y coordinates, and orientations of objects in 3D space; doing so may well lead to an explosion in the symbolic state space [3]. Moreover, as pointed out in [2], there are also relevant symbolic facts that do not depend on a pose, such as the number of products assembled so far and the weight of a new part. These facts can be managed directly by the agent, for example by directly sensing the environment.

In the situation where there was no viable motion plan when the precondition of an action-rule above was checked, the facts applied by $post()$ instead "describe" the reason. To this end, two useful domain-independent predicates, inspired by [3], are $obsSome(k2, canMov, k1, t1)$, indicating arm $k2$ obstructs at least one trajectory of the task $canMov(k1, t1)$, and likewise $obsAll(k2, canMov, k1, t1)$. The agent could exploit such information by, for instance, moving arm $k2$ out of the way.

6 Conclusions and Future Work

We have presented an approach to interfacing BDI agent reasoning with geometric planning in a way that uses some of AgentSpeak's existing constructs and semantics. We have also shown how interesting abstractions can be extracted from the detailed geometric model and then exploited during agent reasoning. We intend to study these abstractions further, e.g. how to compute $obsSome(k2, canMov, k1, t1)$, and to formalise the integration by extending the operational semantics of AgentSpeak.

Acknowledgements. We thank Elkin Castro, Amit K. Pandey, and the reviewers for their feedback. Felipe thanks CNPq for support within grant nos. 306864/2013-4 under the PQ fellowship and 482156/2013-9 under the Universal project programs. The others are grateful for support from the Evolvable Assembly Systems EPSRC project (EP/K018205/1), and the PRIME EU FP7 project (Grant Agreement: 314762).

7 References

1. Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Proceedings of the MAAMAW Workshop*, pages 42–55. 1996.
2. Lavindra de Silva, Amit Kumar Pandey, and Rachid Alami. An interface for interleaved symbolic-geometric planning and backtracking. In *IROS*, pages 232–239, 2013.
3. Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *ICRA*, pages 639–646, 2014.
4. F. Lagriffoul, D. Dimitrov, A. Saffiotti, and L. Karlsson. Constraint propagation on interval bounds for dealing with geometric backtracking. In *IROS*, pages 957–964, 2012.
5. E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *ICRA*, pages 4575–4581, 2011.
6. E. Plaku and G.D. Hager. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *ICRA*, pages 5002–5008, 2010.
7. Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. Semantic attachments for domain-independent planning systems. In *ICAPS*, pages 114–121, 2009.
8. Andre Gaschler, Ingmar Kessler, Ronald P. A. Petrick, and Alois Knoll. Extending the Knowledge of Volumes Approach to Robot Task Planning with Efficient Geometric Predicates. In *ICRA*, 2015. To Appear.
9. Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *IJRR*, 32(9-10):1194–1227, 2013.
10. Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

11. Nikolas Antzoulatos, Elkin Castro, Lavindra de Silva, and Svetan Ratchev. Interfacing agents with an industrial assembly system for “plug and produce”. In *AAMAS*, pages 1957–1958, 2015.