

# A Composite Architecture for a Realistic Blocks World Domain

**Peter A. Hamilton**

CMSC 677 - Agent Architectures and Multi-agent Systems  
Department of Computer Science and Electrical Engineering  
University of Maryland, Baltimore County  
1000 Hilltop Circle  
Baltimore, MD 21250-0002

## Abstract

I present a domain-specific architecture targeted for a real-world version of the Blocks World domain called “Real” Blocks World. The architecture design attempts to include the best features of several popular Agent Architectures, while mitigating their weaknesses. Agent functionality is considered in light of environment scenarios, single-agent, and multiagent systems.

## Introduction

The concept of an Agent Architecture (AA) (Wooldridge & Jennings 1995) is a central idea in Artificial Intelligence (AI). The primary goal of an AA is to provide a structure or framework for the combined usage of other AI techniques. These techniques often include planning, learning, knowledge representation, natural language processing, data analysis, and data mining, not to mention a host of other sub-fields in AI. The motivation for the study of AAs is drawn from nature. AI was originally directed to mimicking, and ultimately replicating, intelligent behaviors exhibited by human beings and, in general, other living things. These intelligent agents are best represented as intelligent programs, whose internal structures correspond to specific AAs. As such, AAs are often created to showcase, though so far unsuccessfully, complete theories of the mind whereby the entire spectrum of human intelligence can be accurately modeled and, effectively, recreated.

Due to the difficulties in recreating human-level intelligence, AAs are also designed to work optimally in a specific environment or to solve a specific problem. These AAs are usually much more successful and are therefore more well known. Examples include Subsumption (Brooks 1986) in collision avoidance and environment traversal and the Memory-Prediction Framework (George 2005) in image recognition.

---

Style copyright © 2009, Association for the Advancement of Artificial Intelligence ([www.aaai.org](http://www.aaai.org)). All rights reserved.

In this paper I present a domain-specific, composite AA targeted towards a modified version of Blocks World I call “Real” Blocks World. For convention, I define a composite AA to be an architecture that combines and utilizes features from different distinct AAs. This architecture, also known as BSA, combines distinctive features of several well known AAs, including BDI, Subsumption, and ACT-R.

The structure of this paper is as follows. Section 2 covers essential background information concerning the original domain and AAs that served as inspiration. Section 3 presents the BSA architecture and outlines the motivations for the incorporation of the various AA features. Section 4 describes the “Real” Blocks World in terms of the domain layout and the embodiment of BSA as a robot agent.

## Background and Related Work

The architecture design was inspired, in part, by a desire to create an agent that would work well in a realistic version of Blocks World. The source AAs were chosen for the suitability of their respective distinctive features and for their relative uniqueness.

## Blocks World

Blocks World was originally created by Terry Winograd (Winograd 1971) as a natural language system. The most well known form of Blocks World, called Elementary Blocks World (EBW), represents a simple environment composed of a surface and various shaped blocks sitting on the surface or on each other (Russell & Norvig 2003). Simple commands are defined to move blocks around the environment, primarily focusing on stacking and unstacking. The system was used to demonstrate the capabilities of Winograd’s SHRDLU program (Winograd 1971), which could accurately parse and execute queries in the Blocks World language. Query execution corresponds to the movement of blocks into various formations in the virtual environment. Additional applications have expanded Blocks World to represent

a physical system where a robot arm can be controlled to move real blocks (Russell & Norvig 2003). This system is obviously susceptible to real physics rules, adding complexity.

Blocks World is now considered to be a famous planning domain, due to its relatively simple and recursive structure. A Blocks World planner is provided an environment start state and, given facts about the blocks (i.e. size, shape, position, etc.) and rules for moving the blocks, creates a plan to reconfigure the environment into the goal state. As a model for real-world planning, Blocks World is often deemed inadequate due to a lack of expressivity. Since all knowledge of the world is encoded as facts and rules (i.e. symbols), Blocks World is a classic example of symbolic AI, where intelligent behavior is achieved via “symbol pushing”. However, it has been shown that optimal plan construction in EBW is NP-hard (Gupta & Nau 1992), making the development of successful planners an important achievement in AI.

## BDI

BDI (Bratman, Israel, & Pollack 1988), which stands for Beliefs, Desires, and Intentions, is an architecture that focuses on providing a technical, quantitative equivalent to its defining concepts. BDI functions primarily by building goal and plan hierarchies, where goals and plans are composed of subgoals and subplans respectively. Goals, and subgoals, are viewed as *desires*, while plans, and subplans, are viewed as *intentions*. *Beliefs* are simply facts of the world currently considered to be true.

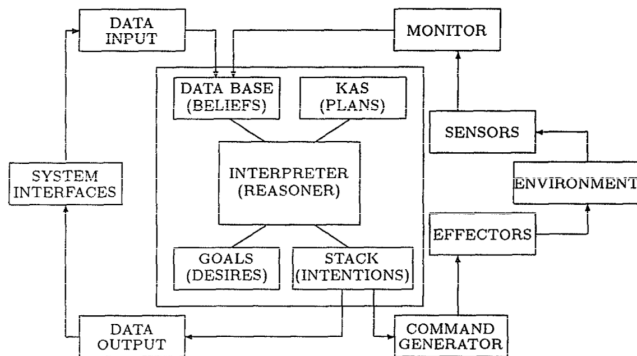


Figure 1: The BDI architecture, specifically visualizing the structure of the Procedural Reasoning System (Georgeff & Lansky 1987).

BDI applies a plan filter to the current set of goals, filtering out suggested plans that violate the goal set. Plan filtration promotes plan stability and addresses the often cited resource-bounds problem by actively minimizing the set of plans under consideration at any one time. The planning protocol is intuitively more flexible

when dealing with smaller plans, since less time is spent building the plan. If changes later occur, the shorter plan is not entirely dependent on the world state, meaning it can be saved for reuse. In general, plan construction intentionally creates gaps where pre-existing, flexible subplans can be inserted. Backup subplans can also be collected to allow for quick plan modification if subplans currently in use turn out to be inadequate.

The Procedural Reasoning System (PRS) (Georgeff & Lansky 1987), shown in Figure 1, provides the main structure for BDI. An internal database stores world facts, while another, known as a Knowledge Area (KA), stores plans as sequences of goals and subgoals. The KA defines elementary plans for basic goals, allowing more complicated plans to be easily reconstructed when needed instead of storing the entire plan sequence. It can also contain metalevel plans, which effectively regulate the formation of subplans in BDI. The PRS interpreter coordinates the flow of data between different KAs, the fact database, the current active set of goals, and physical system control, allowing for smooth control of the general architecture.

## Subsumption

Subsumption (Brooks 1991) attempts to recreate intelligent behavior by modeling the course of evolution. A subsumption system is composed of control layers, each made up of one or more individual modules. Lower-level layers emulate more fundamental behaviors (e.g. collision avoidance). Rather than enforcing a flow of sensory data through filters and on to a central processor, each layer of modules acts in parallel with regard to the rest of the system, having direct access to the sensory data as it comes in. Each module itself is independent of the functioning of the other modules and, in actuality, is usually unaware of their existence. A module focuses only on its internal representation of the world and recommends actions, encoded as action signals, accordingly.

The intermingling of the action signals from all modules results in a “chaos of actions” that Brooks claims results in intelligent behavior. However, an organizational structure, known as the action hierarchy (Brooks 1986), links the different modules together. The action hierarchy can be thought of as a system of virtual wires on which action signals are sent for execution. The action hierarchy also allows for a hard-coding of module hierarchy, which can be done by running virtual wires from distinct modules into one wire. Certain modules can be given priority when sending action signals, allowing them to override lower-level behavior without the lower-level modules having to account for upper-level module intervention.

Each module can send and receive action signals, queuing incoming signals as they are processed. Tangentially, the concept of forgetfulness can be easily mod-

eled here by the dropping of action signals when an input queue fills up.

## ACT-R

ACT-R (Anderson 1996), which stands for Adaptive Control of Thought - Rational, is a cognitive architecture that attempts to embody a complete theory of the mind. The main idea behind ACT-R is that intelligence, and specifically consciousness, is solely composed of knowledge units and simple processing capabilities applied to them.

Knowledge units are categorized into two types: procedural and declarative. Procedural knowledge maps a state of the agent to an action, or new state; such a mapping is called a production rule. Declarative knowledge, representing simple facts about the world, are organized like entities in a database system, having attributes and basic relationships. These entities are called chunks and can be created by production rules or by direct analysis of sensory input.

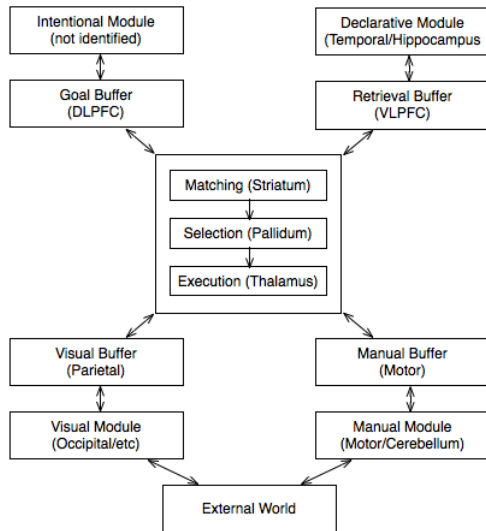


Figure 2: The ACT-R architecture, with specific modules mapped to the regions of the brain that would handle the corresponding functionality. Adapted from a diagram in (Anderson *et al.* 2004).

The architecture itself is composed of three generic parts: a set of modules, a set of buffers, and a pattern-matcher (Anderson *et al.* 2004). An augmented version of this setup can be seen in Figure 2. Modules contain both production rules and chunks. Buffers provide an interface to the various architecture modules; the current content of all the buffers constitute the current state of the agent. The pattern-matcher analyzes the contents of the buffers and matches the agent state to a production rule, which then fires. This changes the state of the agent, causing the modules to load different

knowledge units into the buffers. The pattern-matcher again analyzes the agent state, selects another production rule, and the rule fires. This sequences of events effectively guides the agent through its current task and, in general, through its environment.

## The BSA Architecture

The BSA architecture, named for its reliance on BDI, Subsumption, and ACT-R, combines some of the more novel features of each source AA while attempting to augment total functionality. For an overview of BSA, see Figure 3.

### The BDI Core

The core structural layout of BSA corresponds primarily to that of BDI, in that there is a central interpreter that manages and interfaces between collections of facts, goals, plans, and metaplans. The knowledge database contains all of the empirical knowledge of the agent. This includes an environment map along with data representations of objects and, possibly, other agents in the environment.

It is necessary for the goal, plan, and knowledge databanks to be seeded with initial goal and plan elements, since the unsupervised learning process for deriving these from sensor data is beyond the current scope of BSA. Since goals, subgoals, plans, and subplans are so closely linked in traditional BDI, it is incredibly important for the initial seeding process of the goal and plan sets to be coordinated. If the BDI interpreter attempts to develop a plan for a goal that is not represented in the plan set, it will be incapable of functioning properly. For this paper, we assume that this initial seeding process is done properly and no fatal errors arise upon startup.

Given a goal, the interpreter will search through its set of plans, pruning as necessary, and will compose a set of viable plans and subplans, defined according to the BDI convention in terms of goals and subgoals. As this planning process takes place, the interpreter will hand temporary control over to the subsumption motion controller, providing guidance by passing on the relevant situational data associated with the previous active plan.

### Subsumption Rationale

The subsumption architecture, including basic collision avoidance, random wandering, and directed exploration, acts as the main motion controller. Once a plan is processed by the BDI interpreter, the listed physical actions associated with the plan are provided to the subsumption module and allow it to handle the low-level physical interactions with the environment.

Subsumption also acts as the agent’s reflex module, allowing it to take over quickly if immediate action is needed to protect the agent. However, the BDI interpreter is allowed to provide direct input to the actuators, overriding the action signals of the motion controller. An example of where this is useful will be provided later.

Regarding the BSA architecture from a subsumption point of view, the BDI PRS, augmented with the ACT-R buffers, acts as an advanced upper-level module that handles goal and plan coordination, a capability that the original model of subsumption could not efficiently emulate.

### Knowledge Organization with ACT-R

Sensors and actuators, the architecture’s main interface with the environment, are represented by individual modules like those used in ACT-R, accompanied by individual buffers whose contents at any given time step define the state of the agent.

The ACT-R architecture is also used as a model for data representation. Specifically, the plans used by the BDI interpreter are actually ACT-R production rules. Empirical data regarding the world is represented by many individual chunks. The attributes of chunks have also been extended to apply to production rules, where rule attributes match with agent goal and state configurations. This allows the BDI interpreter to take world knowledge, the agent state, and the current goal set and match them to whole or partial plans. The resulting plans are then used to update the goal and plan sets, adding or removing goals and plans as needed, as well as the knowledge database.

### The Flow of Agent Actions

Once the agent activates in its environment, its sensors activate and pass data to both the subsumption motion controller and the sensor buffer. Subsumption allows the agent to initially wander through its environment, avoiding collisions where appropriate. As the agent moves, situational data provided by the sensors is constantly being sent to the BDI interpreter, which links the individual situational chunks to form an elementary environment map. If left to wander long enough, subsumption alone will drive the agent to explore areas it has not, or at least cannot remember, visiting. However, this will most likely not occur due to the intervention of the BDI interpreter.

During the initial wander phase, the BDI interpreter is updating the agent state based on the sensor buffer contents. Once links are found between the agent state, the initial goals, and the initial plans, the interpreter will begin to form plans to try and satisfy the goals. The requisite actions tied to the intermediary steps of

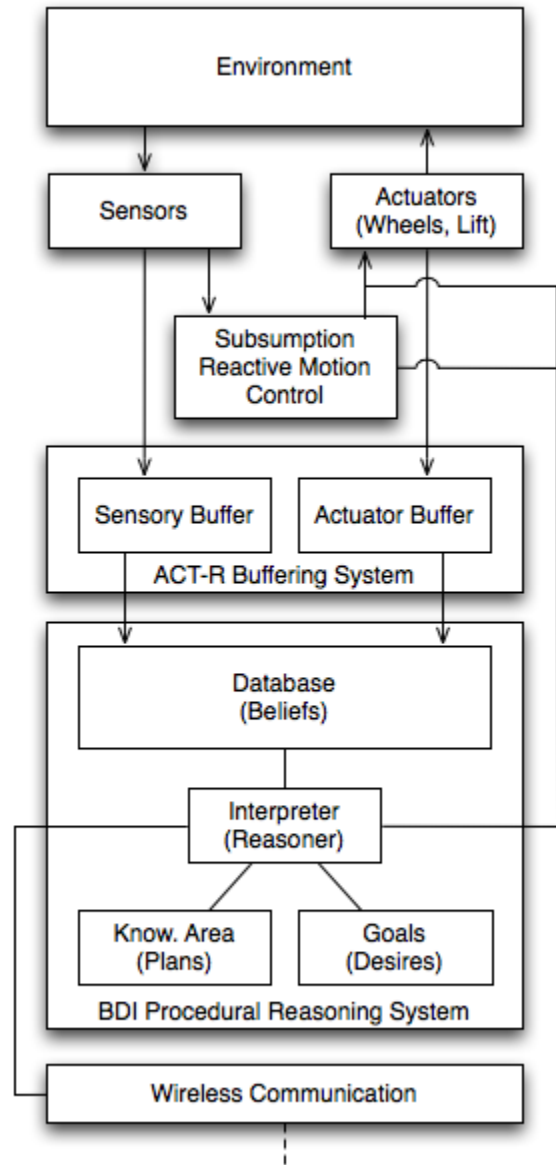


Figure 3: The core of the BSA architecture is an adapted version of BDI’s PRS. The modified PRS acts as an independent planning layer for the original Subsumption architecture, with the ACT-R buffering scheme simplifying the process of determining the agent’s current state. A wireless communication module is included to allow for the formation of multi-agent systems

the plan are then sent to override the wander actions of the subsumption motion controller.

## “Real” Blocks World

The original inspiration for the BSA architecture, “Real” Blocks World is defined as a real-world equivalent to the Blocks World domain. In this case, we will focus primarily on a warehouse environment where BSA is embodied by a robot known as the warehouse grunt.

### The Warehouse Layout

The warehouse environment preserves most of the features of the original Blocks World. The base surface is simply the warehouse floor and all objects on the surface are crates being stored in the warehouse. Crates are cube shaped and can be stacked on top of other crates. Crates are delivered to the warehouse at one of several loading docks and must be stored until they are requested for shipment. If a desired crate is buried underneath other crates, those crates must be removed before the target crate can be moved. An example layout for a warehouse is provided in Figure 4.

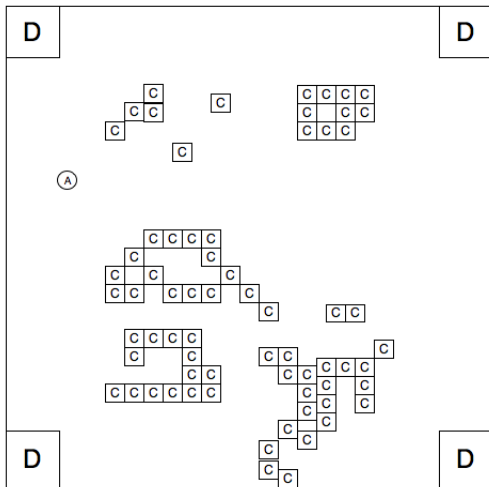


Figure 4: An example warehouse for the “Real” Blocks World domain. The four corners, in this instance, represent loading [D]ocks. Individual [C]rates are scattered around the environment. A single [A]gent is sent to manage the flow of goods through the warehouse for delivery.

### The Warehouse Grunt

The warehouse grunt is a forklift robot that utilizes the BSA architecture to effectively navigate the warehouse environment and manage receiving, storing, and delivering crates to the various loading docks.

The grunt will start out with a single goal, which directs it to move to a loading dock to check for delivered crates. After a brief period of initial random walking,

the BDI interpreter will generate a local map. Drawing on seed data in the knowledge database, the agent will “know” that it can reach a loading dock by following a wall of the warehouse. Doing so will, eventually, lead it to one of the loading docks in Figure 4. During this initial exploration, the BDI interpreter is extending its local map. It is also creating chunks to represent warehouse objects, including crates and docks. Once reaching a dock, the agent will be informed that it must store a crate. Storing a crate is not too difficult and can simply be coded to imply that the agent should place the crate next to another crate.

Once the agent is done storing all of the provided crates, it may continue to random walk for a time, exploring its environment. However, after a certain amount of time, the BDI interpreter will once again direct the agent to a dock. The agent may need to store more crates, if the dock is active, or it may need to go fetch a crate from the crates already stored. The retrieval process involves backtracking through its partial environment map to the last known location of the crate. When the agent first activated, it started to build its environment map from its wakeup position, noting it as point (0,0) in a Cartesian representation of the warehouse. All objects the agent encounters are defining, positionally, in terms of this origin point, regardless as to whether it maps evenly within the overall warehouse space.

Once arriving at the crate’s location, the agent can utilize basic production rules (e.g. If target crate under object, move upper object) to manipulate stacks of crates and, retrieving its crate, make its way back to the loading dock.

This process of fetch and deliver will continue indefinitely as long as the agent is functional and the warehouse is in use.

### Performance Scenarios and Examples

Examining some of the lower-level details of the BSA architecture as it performs demonstrates the usefulness of the different AA features.

**Internal data representation:** There are several obvious environment objects that will be represented by chunks. Crates, loading docks, other agents, and even the warehouse floor can be adequately represented. Several sample schemas for these chunks are shown below:

```

Crate(ID - No., X - coord, Y - coord, On - ID - No.)
Surface(ID - No.)
Agent(ID - No., X - coord, Y - coord, BaseX -
coord, BaseY - coord, Has - crate, Looking - for -
ID - No.)
Dock(ID - No., Isactive, X - coord, Y - coord)

```

Example chunks corresponding to the schema are also provided:

*Crate*(52, +45, -23, 1)  
*Surface*(1)  
*Crate*(14, +45, -23, 52)  
*Agent*(6, +9, -3, -23, -34, *Yes*, 2)  
*Dock*(2, *Yes*, +250, -100)

Like in a database system, each chunk has a uniquely identifying ID code that distinguishes it from every other object. A crate also has X/Y coordinates, plus an indicator which lists the ID number of the object the crate is on. In the case of the first crate, it is on the warehouse floor. There can only be one surface in the warehouse in the current example, so there will only be one surface chunk. The second crate is actually on top of the first crate. The only agent in the system has a current position and its original starting location. The starting location will be useful when interacting with other agents. The agent chunk also has attributes indicating if it is carrying a crate and what object ID number the agent is currently searching for. As it turns out, the agent is carrying a crate and is searching to deposit it at Dock 2.

The general chunking structure turns out to be quite useful in representing basic knowledge about the “Real” Blocks World domain.

**Picking up a crate:** After locating the position of a target crate relative to itself, the warehouse grunt must move any crates that block the target crate. Interestingly, picking up a crate involves moving into contact with it to lift it with the forklift. However, the basic properties of the subsumption motion controller will avoid collisions and, if one occurs, cause the grunt to back away from the crate it is trying to lift. By providing the BDI interpreter direct override to the actuators, the problem of action loops involving collisions is solved.

## The Warehouse Taskforce

An obvious extension of the single agent paradigm is to group individual agents in a multiagent system. In this case, multiple grunts cooperating to manage the warehouse form a warehouse taskforce. When encountering other agents, an agent must determine the other agent’s starting location so that all positional data exchanged between the two can be converted accurately. Additionally, if one agent is looking for an object it has not yet seen, it can query the knowledge banks of its neighbors and, if they know where it is, use their positional data to augment its own databanks and complete its task.

## Conclusion

I have described a composite agent architecture that combines unique features from the BDI, Subsumption, and ACT-R agent architectures and have additionally defined a specific domain in which the architecture functions well. The structured nature of the “Real” Blocks World domain allows the internal structure of BDI and ACT-R to organize and plan agent actions that allow it to function intelligently in its domain while using subsumption to fine tune motion control.

## References

- Anderson, J. R.; Bothell, D.; Byrne, M. D.; Douglass, S.; Lebiere, C.; and Qin, Y. 2004. An integrated theory of the mind. *Psychological Review* 111(4):1036–1060.
- Anderson, J. R. 1996. Act: A simple theory of complex cognition. *American Psychologist* 51:355–365.
- Bratman, M. E.; Israel, D. J.; and Pollack, M. E. 1988. Plans and resource-bounded practical reasoning. *Computational Intelligence* 349–355.
- Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* 2(1).
- Brooks, R. A. 1991. Intelligence without representation. In *Artificial Intelligence*, volume 47, 139–159.
- George, D. 2005. A hierarchical bayesian model of invariant pattern recognition in the visual cortex. In *In Proceedings of the International Joint Conference on Neural Networks. IEEE*, 1812–1817.
- Georgeff, M. P., and Lansky, A. L. 1987. Reactive reasoning and planning. In *In Proceedings of the 6th National Conference on Artificial Intelligence*, 677–682. AAAI Press.
- Gupta, N., and Nau, D. S. 1992. On the complexity of blocks-world planning. In *Artificial Intelligence*, volume 56, 223–254.
- Nilsson, N. J. 1980. *Principles of Artificial Intelligence*. Tioga, Palo Alto: Morgan Kaufmann Publishers.
- Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Upper Saddle River, New Jersey 07458: Prentice Hall, 2nd edition.
- Winograd, T. 1971. *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language*. Ph.D. Dissertation, Massachusetts Institute of Technology.
- Wooldridge, M., and Jennings, N. R. 1995. Intelligent agents: Theory and practice. *Knowledge Engineering Review* 10:115–152.