Modeling Human Actions through Context

Avelino J. Gonzalez[1]

[1] Computer Science Department, University of Central Florida, Orlando, FL USA, Avelino.Gonzalez@ucf.edu

ABSTRACT. This paper presents and discusses how context is being used to model intelligent human activity – specifically, tactical actions. Tactical behavior involves selection and execution of courses of action that address the current needs of the agent. The discussion centers about the work done in the author's research laboratory that addresses tactical behavior by an agent. A limited discussion about the works of others is also included. Two points of view are discussed vis-à-vis the tactical behavior of an agent: a) from the point of view of a performer of an action (the doer), and b) from the standpoint of one who directs others (agents or humans) to perform the actions (a manager, commander or coach). Additionally, the role that can be played by context in machine learning of tactical behavior is also discussed. This particularly focuses on learning from observation of human performance, known as LfO. LfO has been found to be an effective way for learning agents to learn how to perform certain tasks that are performed by a human and whose actions are observed (i.e., recorded in a time-stamped trace of what actions were taken when).
KEYWORDS. Context-based Reasoning, tactical reasoning, human performance modeling, decision support.

1. Introduction

Over the last 25 years or so, the theme of context in computing has burst upon computer science, mostly as a sub-discipline of AI research. Although by no means exclusively, the most common emphasis has been in context-aware computing [Dey, 2001]. Context-aware computing seeks to recognize the context in which a human user finds her/himself, in ways that can make it readily useful for determining his/her needs in real time, all in the context of an automated personal assistant system. This view of context has led to significant advances, and continues to form the backbone of a highly active research community. However, context plays another, more important role in computing – that of modeling broader human behavior. Tactical reasoning is one of these human behaviors, and is the focus of this paper.

Brezillon and Gonzalez [2014] provide an insight into the state-of-the-art in contextual reasoning and how context can be used to model the physical world and our perception of it. The importance of context in our everyday lives is clear to any reader of this volume. The work expresses how several authors and researchers apply the elusive but quite real concept of context to advantage when modeling or representing real world human activities. This volume contains a broad and deep coverage of the rich subject of context and how it can be used to model everyday tasks and activities typically performed by humans. The chapters presented in Part I "Context in software and systems" show that it is possible to introduce context more intimately within software. This moves our concept of context from an initial view of it as a simple layer between the system and its environment to one in which system and human jointly try to realize a task in a context-sensitive way. In Part II "Context in the computing environment", a key feature of a context-based system is its potential to learn from its interaction with the user and its environment. This feature becomes of paramount importance for mobile systems and ubiquitous computing. The chapters in Part III "Context in an individual human dimension" and Part IV "Context in the collective human dimension" address the other side of the coin - the human dimension of context. Context in computing cannot be considered independent from the human with whom the system interacts. This is because we humans also have our own working context that must be reflected and attended to. Part V "Context in modeling reasoning" and Part VI "Context in representing reasoning" discuss the building blocks of any system that uses context as its main representation paradigm.

Other research threads in context in computing include context-based search in software development [Antunes et al., 2014]. Knowing the context of a search can add new and implicit features

to the search request that can limit the search space and make the search more efficient. Context can assist in NLP [Hung, 2014] because knowing the context of a conversation can help to understand the intended meaning of a speech utterance. This can make a meaningful difference in automatic interpretation of spoken speech, given the general immaturity of automated speech recognition systems. Decision support [Brezillon, 2004; Wienhofen et al., 2014] can also benefit from contextual treatment. Knowing the context of a decision can provide implicit information highly relevant to the decision. The use of context in decision support was pioneered by McDermott [1982] in the classic expert system R1/XCON, used to automatically configure VAX systems. In the problem of diagnosis, a type of decision support, context was used by the GenAID system [Gonzalez et al., 1986] to assist in diagnosing the cause of malfunctions in large, turbine-driven generators. In this latter application, the context was useful in limiting false positive diagnoses when the generator being monitored was in contexts other than full operation, and expectations of sensor readings were significantly different. Context has also been used to advantage in trust evaluation [Liampotis et al., 2014], situational assessment [Gundersen, 2014], and user intent prediction [Kalatsis et al., 2014] among many others. In situational assessment, the relevant aspects of a situation can be different depending on the objectives of the agents in that context.

Somewhat less popular has been the view that context plays an important role in normal human activity. This has been recognized to greater or lesser degrees by a number of cognitive science researchers. In effect, context can be said to be a natural expression of divide-and-conquer, as human knowledge is highly voluminous, and recognition of our current context permits us to only need to bring forward that small portion of our knowledge that is relevant to our present situation (i.e., context), while suppressing the irrelevant rest of our knowledge. One example is that when driving our cars, it is the knowledge about the rules of the road and how to operate one's vehicle that becomes relevant and thus needs to be available for fast and easy access. Knowledge about our jobs, hobbies, family, social plans, upcoming travel, etc. all slide into the background until the context is such that another part of our knowledge is required for a newly-emerging situation. Of course, it is not as simple is as suggested above. We often think about other things while driving our cars, for example, holding conversations with other people at the same time we drive. So, while humans can be in more than one context at the same time (e.g., driving a car and discussing investment strategies over the phone with a financial advisor), it is often a distraction and impairs our ability to work error-free.

See Parker et al. [2013] for a discussion of the role of context in human cognition and how that compares with the use of context in computing. The authors compare selected computational architectures to cognitive paradigms based on key elements of human intelligence. Their objective is to illustrate the similarities and differences between the two viewpoints and highlight the potential effectiveness of context-based computing. Meaningful parallels between the assessment of context in cognition and computation (as found in the literature) are pointed out. These have great implications for both fields of study.

To efficiently and effectively model human activity, context should be somehow incorporated into any model that purports to describe/predict human intellectual activity. This is certainly not a new idea, but one that bears re-emphasizing. This was recognized in the 1990s simultaneously and independently by Turner and by Gonzalez. They devised strikingly similar approaches to modeling human tactical behavior – Turner's Context-Mediated Behaviors (CMB) [Turner, 1998] and Gonzalez' Context-based Reasoning (CxBR) [Gonzalez and Ahlers, 1998; Gonzalez et al., 2008]. Nevertheless, while they are similar, they also do diverge in some meaningful ways. A full discussion of these differences is beyond the scope of this paper; nevertheless, for the sake of completeness, a brief discussion follows.

1.1. *A brief definition of CxBR and CMB and how they compare with each other*

A context in CMB is called a *contextual schema*, or *c-schema* for short. C-schemas contain *descriptive* and *prescriptive* knowledge. The former describes to the system the salient properties of the situation (context) itself. The descriptive knowledge contains expectations about things that can be present in the current situation, as well as items that must be present in the situation if this context is to be in force. It also sets expectations. For example, in the case of a c-schema related to driving an automobile in a city center, an unidentified moving item on the roadway may be identified as a pedestrian because pedestrians can be expected in the context of a city street.

The prescriptive knowledge in a c-schema is, well, prescriptive. That is, it provides suggestions of procedural schemas, which are hierarchical plan-like structures; they are themselves memory structures and their indices can be searched for even better-fitting procedural schemas. Prescriptive knowledge is further separated into *goal directed* and *standing orders.* Goal directed knowledge, as the name suggests, prescribes ways to achieve the goals of the mission. These goals may be assigned priorities by the system author (programmer) or by the system itself as a function of the likelihood of that particular goal being achieved. The standing orders are basic behavioral parameter settings that can be used to help the agent achieve its objective(s).

When an event occurs in the environment, CMB takes three steps to respond to the event: The first, of course, is to *detect the event.* This can be as easy as merely monitoring the environment continually to look for events of interest. Nevertheless, detecting complex events can present a challenge, and often requires that such events be decomposed into simpler events to be properly recognized. Next, CMB *evaluates the importance* of the event. This is context-dependent. This determination leads to the third step – what, if anything, to do about it. Knowing *if /how to respond* is likewise context-dependent. For example, learning of a severe thunderstorm warning while one is at an isolated beach can require immediate action, while the same warning when inside a large building may not require any reaction (other than maybe avoiding windows). If the situation calls for seeking shelter, then the c-schema will suggest how to do that (go to nearest building, get inside a car, or at last resort, kneel down and bend over without touching the ground, preferably in a dry ditch or other such ground depression to minimize one's electric posture).

The CMB system first determines what the goal is for the reaction (what needs to be done). Then the context assessment process begins, where all the contexts (c-schemas) in the system are examined to see which can suggest the best reaction to the changing situation. One or more c-schemas may be selected. If more than one, these c-schemas are merged into one context that now provides the prescriptive actions. If the recently-merged c-schemas contain no conflicting knowledge, then the merger goes through easily and smoothly. However, if it is discovered that the merged c-schemas contain conflicting knowledge, then such conflicts must be resolved. The extreme way of handling such conflicts is to simply discard all pieces of conflicting knowledge. A better way (if possible) to resolve conflicts is to find a halfway point between the conflicting elements. The newly-created merged context now becomes part of the set of contexts of the CMB system and is treated like any other context that already existed. Upon having dealt with the changing situation in this manner, the CMB system continues to monitor the environment for new changes and performs the same process when a new change is noticed.

CxBR is quite similar to CMB is that its *contexts* contain prescriptive knowledge (although it is not called that in CxBR) about what the agent should do when in a particular context, as well as how to do it. In contrast to CBM's c-schemas, contexts in CxBR can be and typically are hierarchically organized. Figure 1 depicts a generalized hierarchy of contexts.

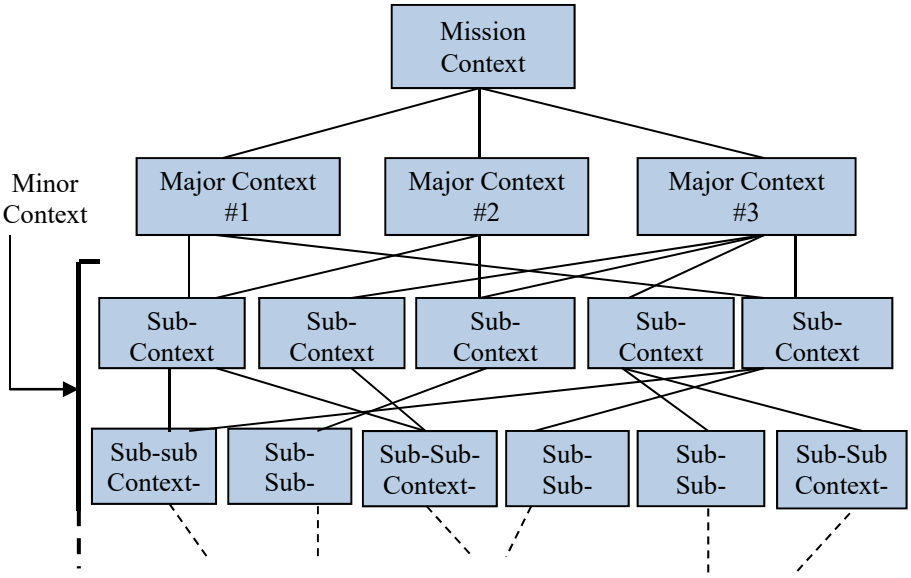

Figure 1. *Hierarchical Context Organization [from Gonzalez, Stensrud and Barrett, 2008]*

A *Mission Context* contains the definition of the mission to be undertaken, its overall objective(s), any applicable constraints as well as any information relevant to the conduct of this mission or task. It also lists all the *Major Contexts* and *Minor Contexts* that are applicable to this mission. These are a subset of all available contexts (the *universe of contexts*), many of which are not be relevant to this mission/task, so they need not be considered. Mission contexts are merely descriptive, to borrow the terminology from CMB. Major contexts and minor contexts, on the other hand, are fully prescriptive and contain two types of knowledge that are critical to how CxBR operates: a) *behavioral knowledge* about what to do when in that context as well as how to do it; b) *situational knowledge* about the environmental conditions necessary for this context to be the most relevant one, and therefore be *active.* An active major context is the sole controller of the agent's actions. Exactly one major context must be active at all times, duly controlling the agent. The "token of activeness", so to speak, is passed among major contexts, depending on the situation and the events occurring in the environment. The difference between major and minor context is simply that of scope - they are virtually identical otherwise. Major contexts represent higher-level, more important situations that persist for a longer time (e.g., driving on a country road); minor contexts represent actions of shorter duration and somewhat lower level (e.g., passing another car on a country road). Both types of contexts are activated and de-activated directly and purposely. Major contexts are activated through *Transition Rules* associated with every major context that monitor the environment continually, looking for events that might trigger activation of their associated major context. Minor contexts, on the other hand, are activated by a major context rather than by events in the environment. Thus, events in the environment only affect minor contexts indirectly, as filtered by the active major context. Minor contexts represent common lower level actions that can be abstracted for re-use by other major contexts.

CxBR does not allow more than one major context to be active simultaneously. However, a major and a minor context may be active at the same time when the minor context is called by the active major context. CMB addresses this issue by merging contexts and de-conflicting any counteracting knowledge in the newly-merged c-schema that may come from the component c-schemas. CxBR avoids the problem of conflicts altogether by not allowing more than one active context. While this makes it easier to execute contexts (no conflicts) and more constructive (contexts are more complete in their prescriptive knowledge), it does require a more thorough design and implementation of the various major and minor contexts to make them complete. This requires a greater effort on the part of the programmer.

The passing of the "activeness" token is called a *transition* between contexts. These transitions are triggered by the transition rules mentioned above. This process is also similar in some ways to how CMB operates, except that CMB does it in a centralized fashion. That is, the control over which context becomes active is done centrally. In CxBR, meanwhile, these decisions are made by the contexts themselves. There are two ways this can happen: the first is the traditional way, where the first transition rule to fire causes the currently-active major context to de-activate and the major context to which the transition rule that fired is associated now becomes active and takes over control of the agent, displacing the old active major context. Each major context is provided with the appropriate transition rules and the "transitionability" (to coin a new term) of each major context is pre-defined in a *context map*. See Figure 2 for an example of a context map.

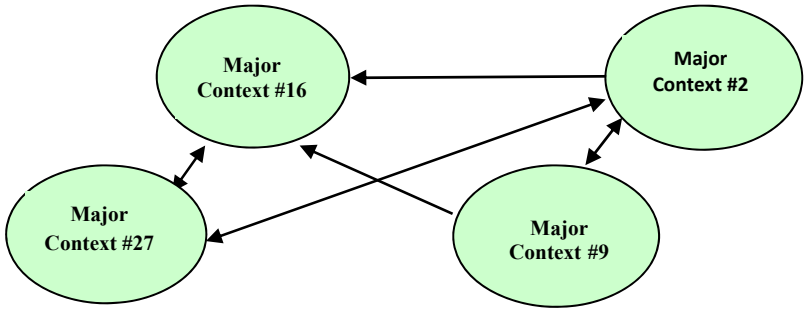

Figure 2. *A Context Map [from Gonzalez, Stensrud and Barrett, 2008]*

This has the advantage of being easily and precisely controlled by the programmer. On the other hand, it imposes on the programmer the need to pre-define all transitions and transition criteria, which can sometimes be burdensome. The second way in which CxBR implements the transition is through a *competing context* concept (see [Saeki and Gonzalez, 2000]). Slightly more similar to how CMB operates, this competitive process identifies the needs of the situation and holds a competition among only those other major contexts deemed to be compatible with such a transition. The major context that can best assist the situation is chosen as the next active major context. While much more flexible and realistic than the first way, it does require that the immediate and specific needs of the situation be identified – something not always easy to do a-priori in a general way. The currently-active context can assist with identifying the needs of the situation, making this a strongly context dependent choice.

Behavioral knowledge relates to knowing how to behave when in a context. While equally important as situational knowledge, it is generally more voluminous. The representational format for behavioral knowledge is not specified by CxBR … on purpose … to provide flexibility to the developer. Most applications have used simple procedural functions to represent behavioral knowledge; others have used production rules [Brown, 1994] while yet others have used embedded neural networks [Sidani and Gonzalez, 2000; Gonzalez, Georgiopoulos and DeMara, 2007]. A discussion about which representational approach works better for what kind of application is beyond the scope of this article. Situational knowledge is almost exclusively represented by production rules.

Contexts also contain an expectation of what is likely to be present and/or occur when the agent is in that context. The behavioral knowledge in a context, therefore, must take such expectations into account, either implicitly or explicitly. My preferred example of a CxBR application is that of an agent that drives an automobile, whether in the virtual world [Gonzalez, Grejs and Gonzalez, 2000] or in the physical world. In this road-driving application, the agent's CxBR knowledge could be decomposed as to the type of road environment it would have to navigate to accomplish its mission of getting from

point A to point B. For example, driving the car in the central core of a city might require some knowledge that would be different from that used in driving through the more pastoral suburbs. In the former, one would expect heavy traffic, many pedestrians, many traffic lights, loud noises, etc. One would need to drive slowly, carefully and always watching for pedestrians crossing in front of the car. In the latter, traffic would be lighter, with fewer traffic lights, but with the possibility of children playing on or near the street. Other contexts in this application would include driving in high speed, access-controlled highways (i.e., freeway, autoroute, autobahn), in a two-lane country road and in a parking lot. Again, these require practices that are different not only from each other but also from the first two contexts discussed.

As an extension of expectation management, a context can also limit what future context could arise from the current context. This has to do with the transition to a new major context as we discussed above. This is because in the real world, not everything happens randomly or miraculously. There is generally a cause-and-effect relationship that can serve as a harbinger of new events about to take place. CxBR takes advantage of this to limit the subsequent contexts to only those to which the current context can possibly transition. For example, if in enemy territory, the possibility of a sudden attack is quite real. So, an agent must expressly look for situations that may lead to ambushes, and quickly react to sudden enemy attacks. However, when located on base at the rear of the battlefield, far from any enemy activity, the possibility of a surprise enemy attack is very low, and the agent need not be actively seeking for ambush situations. Surely this can lead to disastrous surprises, but so it is also in the real world. The restricted list of possible transitions limits the search for possible next contexts, making the determination more efficient. This is in contrast with CMB where all c-schemas and examined to find the ones that address the situation (and are later merged).

Situational knowledge is used to identify when the situation has changed sufficiently so that the currently-active context is no longer valid. When this happens, the active context is de-activated, and another more relevant context becomes activated and able to control the agent. In the automobile driving example, the agent has exited the freeway and is now in a urban center street. This is critical, as the inability to make the transition from a freeway context to an urban center street context could preclude the correct behavior of the agent. Imagine our driving agent cruising down a city street at 75 Miles per hour because it still thinks it is in a freeway. At best, it would get a ticket; at worst, it would get into a deadly accident.

As the reader can see, CxBR and CMB both address the representation and prescription of intelligent tactical behavior. Although similar in many ways (e.g., contexts and c-schemas contain the appropriate functionality to cause the agent to navigate the context successfully), they do differ in some fundamental ways. One is not better or worse than the other – just different. It is up to the potential user to decide which works better for him/her vis-à-vis the application contemplated.

1.2. *Summary of introduction*

This section discussed a broad perspective of how context has been used to model human intellectual activity in computers. Note that the above statement uses the term “intellectual” in lieu of the word “cognitive”, as I don’t claim to use context to model the human cognitive process, but rather, in modeling applications that use context as the fundamental basis of the model. The focus of this paper is on three specific broad areas of application: 1) modeling tactical behavior directly (agent decides on course of action, and executes it); 2) machine learning of tactical behavior; and 3) modeling command and control (i.e., indirect tactical behavior – agent determines course of action and commands others to execute it). It should be noted that the military connotation of the first and last items on the above list is misleading. Humans undertake tactical behavior every time we get behind the wheel of a car or visit a supermarket with a shopping list in hand. Decisions have to be made quickly to maintain the pursuit of an objective (i.e., a destination, a purchase). While certainly military command

and control is one highly applicable domain, this discussion need not be limited to military operations. As defined here, command and control can also refer to management actions in the context of many types of operations (construction, product engineering, manufacturing, etc.) as well as in competitive team games (e.g., football, basketball). Below is a discussion of using context to represent tactical behavior.

2. Tactical Behavior Representation

My original concept of CxBR was specifically intended for modeling human tactical behavior (directly). That is, the decisions and actions taken by a human to help her/him successfully manage the current situation and accomplish his/her objectives. While tactical behavior is normally associated with team sports and with military operations, the truth is that we execute our own tactical "operations" routinely in our daily lives. This includes driving our automobiles, shopping for food at a supermarket and other such mundane activities.

Thorndike and Wescourt [1984] assert that tactical reasoning involves "*1) assessment of the situation at hand, 2) selection of a plan to most properly address the present situation, and 3) execution of that plan"*. This plan may be an explicit plan formally described and decided upon after discussions among the stakeholders, or it may simply be a set of decisions made instinctively on the spot by the decision maker in the heat of the moment. This depends on the context of situation faced.

Thus, tactical reasoning can be said to involve a time-based decision-making process, where decisions are made continually and sequentially by an agent that seeks to complete a mission or a task over time. Tactical reasoning also involves the agent implementing the actions related to the decisions it made, and these actions will inevitably change the environment, either a lot or a little. Such changes will then have an effect on the agent's subsequent decisions. It can take an arbitrary length of time to complete such a mission or task - from seconds to days. I assume here that tactical reasoning is always made in the context of an environment that may be uncertain or unknown, and possibly hostile. An agent reasoning tactically must move through time and space (physically or virtually), and must make decisions that are more often than not time critical. While some of the actions resulting from decisions it made could be reversible and without incurring any residual effects, most are not.

The situation faced by an agent (human or software), of course, is characterized by the context in which he/she/it is. The context suggests several decisions/actions that can/should be taken, and others that must be taken to ensure one's survival, be it physical, economic or simply in a game. A programmer building an application of CxBR must recognize the potential situations that an agent could face while executing the tactical plan, and define and build an appropriate context for each one.

To properly explain the ideas set forth in this section as well as in Section 1 above, I now describe an example application of CxBR to submarine warfare, where an agent controls a submarine while carrying out a patrol mission. I should note that the submarine warfare tactics described here represent my interpretations of open-source material [Khboshch, 1990; Clancy, 1985, 1986, 1993], and were not reviewed by or obtained from subject matter experts. The mission represented is called **SEARCH-AND-TRACK,** where our agent submarine is to travel to a sector of ocean, look for the presence of enemy submarines, and when one is found, to track it closely from behind. If expressly commanded to do so, it can attack the enemy submarine, as well as defend itself from attacks. When recalled, it is to return to base from the assigned sector. This example is a modified and abridged version of what was originally published in Gonzalez and Ahlers, [1998].

The submarine agent to be described here is controlled via CxBR. The submarine agent represents an intelligent opponent in the context of using it to train a (human) submarine commander. Thus, our submarine agent is labeled as the *opponent submarine*, or ***opsub*.** The submarine agent commanded by

the commander trainee, on the other hand, is referred to as ***ownsub*,** as is the custom in the U.S. Navy. **Ownsub** depends on the human trainee to control its actions, and thus has no intrinsic intelligence. The objective of **opsub** is to assist in the training of the submarine commander controlling **ownsub** by providing him with a credible threat that reacts to his tactics in an intelligent fashion, without the need to assign an expensive and scarce expert to control **opsub** during the training simulation. Therefore, the scenarios described focus on **opsub** and how it behaves automatically in response to the situation.

This example application was implemented in the CLIPS tool [www.clipsrules.net]. Major contexts and sub-contexts were activated in CLIPS by asserting a fact that indicated the activation of the context (mission, major and minor). For example, the active mission context is represented in CLIPS as the fact:

(mission-context **SEARCH-AND-TRACK**)

Likewise, activation of a major context was done by asserting the following fact:

(active-major-context **Sector-Search**)

Activation of a minor context is done with the active-sub-context fact:

(active-sub-context **clear-baffles**)

Retracting these facts, of course, has the effect of deactivating the contexts. A new fact is asserted immediately thereafter, announcing the newly-activated context. The situations faced by the agent are also identified through facts that are posted to the fact base by the simulation part of the system every 5 seconds. The old facts are simultaneously retracted.

Opsub and **ownsub** are implemented as object instances of class SUBMARINE in the COOL object language integrated in CLIPS. Their *static* slots (defined as those whose values will not change during the simulation) define their capabilities. Examples of these are their maximum speed, quiet speed, maximum depth, periscope depth, weapon systems, (e.g., number and ranges of torpedoes and missiles), sensors (e.g., range and types of passive sonar, active sonar, and towed arrays), and Electronic Warfare capabilities (e.g., sonar decoys). Additionally, their *dynamic* slots (those whose values may be updated during the simulation) describe its actual position (i.e., x-coordinate and y-coordinate), depth, heading, and speed, in the course of the simulation, as well as whether the sensing equipment is on or off. Damage assessment information, as well as the supply of weapons and food, are also found in dynamic slots.

The data containing all of the external information relevant to the mission are found as slots in various other class objects. For example, this mission calls for a sector of open ocean to be patrolled as part of the **SEARCH-AND-TRACK** mission, therefore, a sector object is instantiated that contains the points in the ocean that define the area to be patrolled. It may also contain other information related to the waters within that sector, such as the depth, the location of the thermal layer (a layer of water with high temperature gradient called the *thermocline* that has the effect of reflecting sound), and the location of any known underwater geological formation or shipwreck that can affect **opsub**'s ability to navigate that sector.

Weapons (e.g., torpedoes, missiles, mines) and defensive devices (e.g., sonar decoys) are also instances of classes that describe the specific devices. They have member functions that move them towards the target or weapon. These functions use probability functions to decide whether they will successfully destroy their target or not. This probability is based on the presence of evasive maneuvers and deployment of decoys.

When one submarine agent, through its sensors, can become aware of the whereabouts of the other (a critical issue in submarine warfare) depends on a combination of several factors, including the distance, speed, and depth with respect to the other submarine, as well as to the thermocline and the sensors being employed.

Monitoring the simulation for changes in the situation and implementing the resulting context transition is carried out with the help of a pattern-matching, forward reasoning production system. All information that needs to be acted upon immediately upon occurrence is posted in the production system fact base.

If a change of mission is dictated by the external inputs (e.g., new orders from fleet command), then the fact that enables the old mission is retracted from the fact base, and one indicating the new mission is asserted. However, this is not expected to happen often.

The fact pattern that begins with *active-major-context* takes on the name of whichever major context is active at any one time. When a major context is to become activated, the presently active major context is deactivated and considered the *previous-major-context*. This feature introduces the ability to remember past actions when it is important to know what **opsub** was doing previously.

The activation of a minor context that is compatible with the active major context is done through the active major context by posting a fact onto the fact base that indicates that that particular minor context is now active.

When one of the transition rules associated with the active major context determines that its active major context should be deactivated, it first fires a rule that retracts the fact advertising it as the active-context from the fact base. Secondly, it proceeds to send an initialization message to the class object for the new major context to be activated. Which major context to activate is determined by the transition rules. The initialization action of the newly activated major context begins by posting the appropriate active-major-context or active-sub-context fact onto the fact base to enable the new activation. Secondly, it implements any changes in the control variables of the **opsub** agent that may be required by the new major context. In the case of **opsub**, this may indicate a change in depth, speed or heading, or to fire its weapons or its defensive countermeasures.

The **SEARCH-AND-TRACK** Mission focuses around **opsub** patrolling a specified rectangular sector of open sea until told to return to base port. If it detects an opponent (namely **ownsub)**, **opsub** is to track it from its rear (called its *baffles*) until either it loses contact with it or **opsub** is recalled by fleet command (in reality, the instructor). The objective of this mission is not to destroy **ownsub**, but rather to keep it under surveillance. Nevertheless, unprovoked attacks by **opsub** upon **ownsub** are permitted, but a higher authority (i.e., the instructor) must order them. If counter-detected by **ownsub**, **opsub** is to break contact and evade the opponent. If attacked, it is to try to evade the in-coming weapons and then counter-attack. **Opsub** must always seek to avoid counter-detection by **ownsub** as well as its own destruction.

This **SEARCH-AND-TRACK** mission has nine major contexts and eight sub-contexts associated with it. The first five (5) major contexts described below are found in the *plan* to execute the **SEARCH-AND-TRACK** mission. The plan is the list and sequence of major contexts that are minimally necessary to carry out the mission assuming no contretemps. The other four (4) major contexts represent tactics that may be used by **opsub** in response to abnormal circumstances, but are not necessary in this mission. The major contexts, with a short description, are as follows:

a) **Transit-To-Sector:** This context sets course and speed to get **opsub** to its designated sector.

b) **Sector-Search:** Causes **opsub** to search the sector until it finds **ownsub** or is recalled home.

c) **Maneuver-Into-Position:** Once **ownsub** has been found, this context puts **opsub** into position to track **ownsub** by "getting on its baffles".

d) **Target-Track:** Causes **opsub** to follow the detected **ownsub** wherever it goes.

e) **Transit-Home:** Guides **opsub** back to its home base.

f) **Attack:** Gives **opsub** the ability to fire its weapons at **ownsub** when ordered to do so.

g) **Escape:** Causes **opsub** to move away from **ownsub** as quickly as possible, in a direction opposite to **ownsub's** bearing.

h) **Under-Attack:** Will cause **opsub** to evade the attack by maneuvering and releasing decoys.

i) **Counter-Attack:** Causes **opsub** to attack the aggressor **ownsub** after its own threat has ended.

The minor contexts used in this prototype are the following:

1) **sprint-and-drift:** Used when there is a need for **opsu**b to get to a destination fast, yet, also be aware of the surrounding environment.

2) **lie-low:** Causes **opsub** to go very quiet until **ownsub** has passed.

3) **get-behind:** Causes **opsub** to get behind **ownsub**.

4) **keep-quiet-and-shoot:** Activated when, upon initial detection of an attack, the distance is judged to be too short for a successful escape.

5) **communicate:** Causes **opsub** to go to periscope depth to send and/or receive a message.

6) **clear-baffles:** Causes **opsub** to suddenly and radically change direction to ensure that no one is behind its baffles.

7) **approach:** Gets **opsub** into position to attack **ownsub**

8) **immediate-attack:** causes **opsub** to immediately launch weapons at **ownsub**

An example scenario was carried out on the prototype in which **opsub** sprints and drifts to the assigned sector and carries out as sector search. After an arbitrary period of time, it detects **ownsub** and gets behind its baffles to track it. It tracks it for an arbitrary period of time, clearing its own baffles periodically. Upon being commanded by fleet headquarters, it attacks **ownsub** and then attempts to escape. **Ownsub** launches a counter-attack and **opsub** attempts to evade it. Depending on the success of each attack (decided on a probabilistic basis), **ownsub** and/or **opsub** are either destroyed or escape. If **opsub** escapes, it is told to return home, and it does. Figure 3 shows a time-line describing the sequence (but not the duration) of events, both at the Major-Context and the Sub-context level.

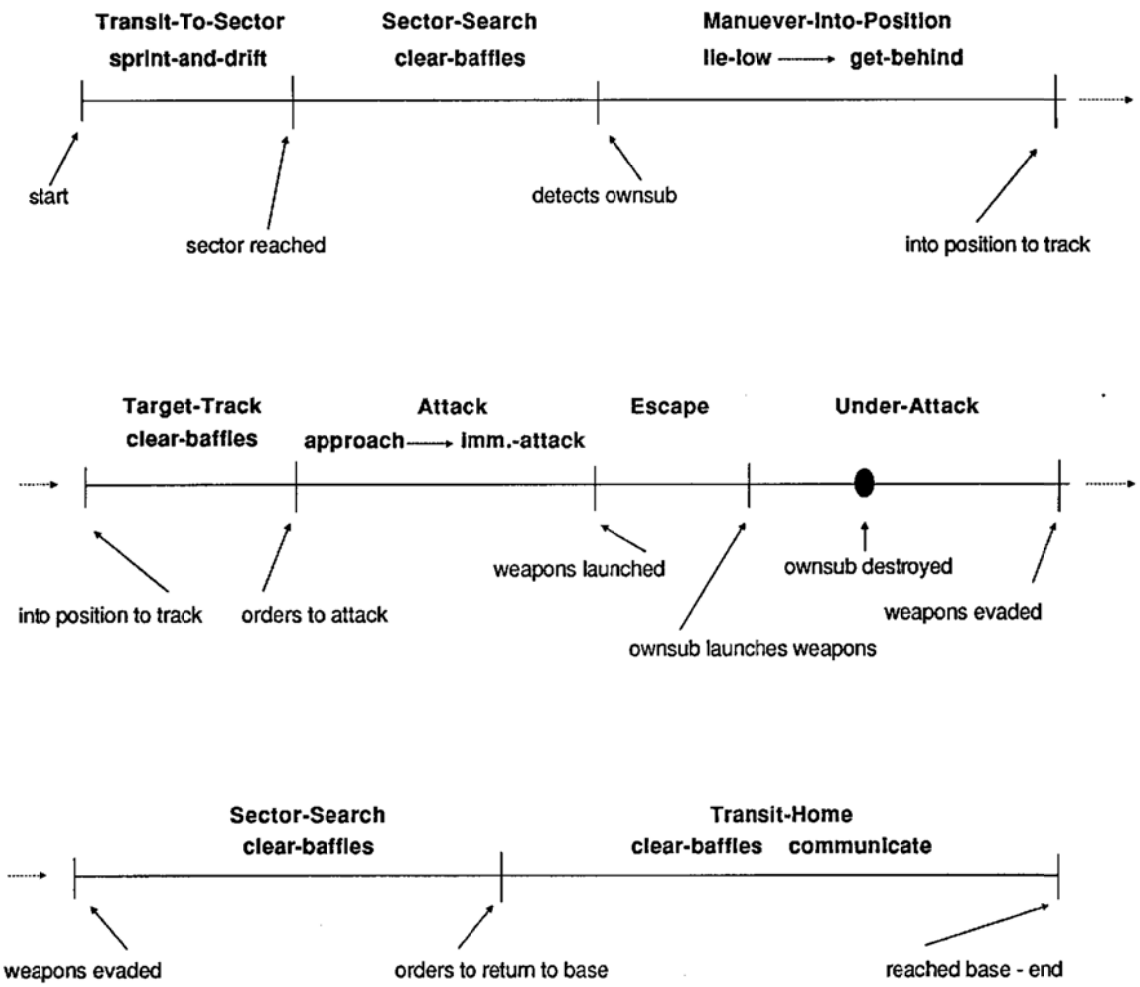

Figure 3. *Prototype Scenario Time-line [From Gonzalez and Ahlers, 1998]*

3. Machine Learning

Machine Learning has of late been the backbone of AI research. The techniques developed in the 1980s as knowledge discovery in data bases have now been renamed and transformed into data mining/big data, with a major component in machine learning. Classification tends to be the task of choice in this interpretation of machine learning. Researchers continually search for new and more efficient algorithms that can make the learning of classifications more efficient and more accurate, over different types of very large and different data sets. However, learning how to perform tasks that are normally done by humans has been largely (although far from totally) forgotten. This section focuses on how context, specifically CxBR, can be used to assist the task of an agent learning how it can perform activities and behaviors normally done by humans.

I and several of my students have focused on the use of contextual reasoning to facilitate the problem of machine learning. Our work has generally revolved around the idea that humans learn how to do many tasks merely by observing how others do it. This is as true for young children learning to play baseball as it is for college students learning advanced mathematical proofs. We refer to this as *learning from Observation of Human Performance*, or simply *Learning from Observation* (LfO). LfO shares some similarities with *Learning from Demonstration* (LfD), a growing area of research in robotics, where robots are taught how to perform tasks by having them observe a human expressly demonstrate the task. The differences are that in LfO, no effort is made to specifically demonstrate. Rather, a human is observed doing the task in a natural environment, where she is observed unobtrusively (i.e., no interruptions and no questions asked). In fact, the human "actor" need not even

be aware that she is being observed as she performs the task, and may not necessarily be a willing participant. LfO is, therefore, a more general form of learning than LfD, and permits learning from unwilling participants, such as game opponents or military enemies. In our work, we have used context as a way to divide and conquer the learning problem. Now the learning algorithm only needs to learn actions that are relevant to a specific context, as well as what (environmental and/or internal) conditions are necessary to realize that the agent is facing that particular context. There are many forms of contextual reasoning that can be applied to learning problems, but we choose to use CxBR.

The first approach (in our lab) was by Sidani in the mid-1990s [Sidani, 1994; Sidani and Gonzalez, 2000] and it involved the use of recurrent neural networks combined with CLIPS rules (see www.clipsrules.net) to build a hybrid system. The simulated agent (an automobile driver) had no a priori knowledge about driving. It had to learn everything on the job by merely watching a human drive the simulated car, with no possibility to ask questions. The automated learning involved training several neural networks to guide a simulated automobile through a set of traffic lights and intersections. The system was called a hybrid because the CLIPS rules were not learned automatically, but rather, were created by a human to identify the context which the car driving agent faced and therefore, which specific neural networks to execute. Thus, during operation, the CLIPS rules would determine the context, and the context-appropriate neural network would control the agent. The system worked well in a limited environment (no other automobiles, although it did include pedestrians trying to cross the street).

Fernlund (see [Fernlund et al., 2006]) made significant improvements on Sidani's work by introducing Genetic Context Learning (GenCL), which learned tasks and behaviors, also in the automobile driving domain, by using a trace of human driving performance as the fitness function with Genetic Programming. It produced much better results than Sidani's work because it was able to learn everything - behavioral as well as situational knowledge. It also displayed a significant ability to generalize, although at the cost of significant more computation time. It also required non-trivial manual contextualization of the human performance data for it to serve as the fitness function. This contextualization involved identifying contexts by clustering the data points that were somehow similar, and looking at the entire trace to see when and how often such clusters of similar data occurred. This process, therefore, involved manually clustering observed data to identify patterns of similar behavior that suggested a single context, and then (also manually) partitioning these detected contexts to determine where else they occurred in the observed trace of the human actor's actions. For example, all sections of the trace that dealt with a traffic light were considered part of the traffic light context. Those trace segments that involved intersections likewise belonged to the intersection context, etc.

Trinh's COPAC system [Trinh and Gonzalez, 2013] sought to automate the above contextualization process that was done manually by Fernlund. This involved discovering contexts from the trace data by clustering the data that were alike into one context, and then automatically determining where else this pattern may have occurred in the trace. This yielded good results and greatly simplified the use of GenCL.

Stensrud [Stensrud and Gonzalez, 2008] used ART networks to identify the high-level behaviors used in a discreet game (poker) as it was being observed by the learning agent (i.e., the ART network). In the process, he made contributions to learning the transition criteria between contexts in poker.

Johnson [Johnson and Gonzalez, 2014] addressed the much more difficult problem of learning how teammates work together on a collaborative task by observing their behaviors unobtrusively. She also used contexts to break down the different behaviors observed. She used a version of Case-based Reasoning to learn and store the learned behaviors. Her system, called COLTS, worked well in problems of simple and intermediate complexity, but failed when trying to learn soccer team play from

observation of RoboCup soccer games. This was a result of the high complexity of the cases learned and subsequently used for the more complex tasks in playing soccer.

Lastly, Stein [Stein and Gonzalez, 2011], used NeuroEvolution to learn tactical behaviors from observation, and then enhance the agent's performance through experiential learning (reinforcement learning). Stein applied a NeuroEvolution algorithm that he developed called PIGEON-Alternate directly to four problems (i.e., testbeds) in a context-free manner; that is, he presented the entire trace of time-tagged human performance to the PIGEON-Alternate algorithm to create a monolithic agent. Two human test subjects, code-named V*iolet* and *Orange,* were used as actors to execute the tasks in simulations and generate the traces. In all cases, the actors were given a description of the task to be accomplished and were given 20 minutes of free play with the simulation to acquaint themselves with the simulation and the control devices used. Then they were asked to accomplish some specific tasks in a limited time mode –30 to 60 seconds. Their performances were recorded and the resulting traces containing these data were used to train the agents through PIGEON-Alternate. Different agents were trained for each actor. He tested his work in four testbed problems, ranging from the simple (an agent learning to chase a fleeing agent) to the difficult (an agent learning to off-load cargo containers from a ship with a crane). The first three applications worked well. While far from perfect, these agents were able to learn these tasks well when compared to the actor from whom they learned. The *similarity factors* (a measure of the similarity between the actions of the agent trained compared to those of its human trainer) all ranged from almost 61% to 86.5%. The resulting *performance ratings* (a measure of how well the agent performed a task when compared to perfection) were even more impressive (75% to 90% for the first two tasks, and 30% to 52% for the third and more complex one – driving a car), as the agents were sometimes able to generalize and perform the task better than their human actors (with the exception of the car driving application). The fourth application, however, failed dramatically in its performance rating, as the trained agents for the two human test subjects achieved a performance rating of only 6.6% in this testbed.

Keep in mind that in his original work [2011], Stein did not contextualize the observed data, nor in any way used context in his approach. To address this failure in the fourth testbed, he subsequently modified his overall system (called Falconnet, which incorporates PIGEON-Alternate as its machine learning algorithm) to include contextualization, and found that the agent was then able to successfully learn the complex task of operating a crane to off-load ships. See [Stein and Gonzalez, 2014] for a full description, but this is briefly described next for the sake of completeness.

Stein contextualized the overall problem into different contexts faced by the crane operator agents. He used these contexts to in effect, divide-and-conquer the problem - learn how to act when in each context and learn how to transition among contexts - the essence of CxBR. He used various methods to implement CxBR, and when the performance rating calculations were done in the contextualized Crane application, the results now ranged from an average low of 37% for the worst of these methods to an average high of 62% for the best performing method. The human operators had scored 77% performance rating in the same task.

When compared to the 6.6% performance rating by the agents when using the original context-free approach, it is evident that an agent structured in a context-centric manner was able to perform significantly better than a context-free agent trained with the same PIGEON-Alternate machine learning algorithm. In fact, using the best performing method of contextualization, it was able to perform close to the human actor from which it learned. The use of a context-based structure to accomplish a divide-and-conquer strategy served its purpose effectively for this task.

It is clear from the above-cited works that contextualization (specifically, CxBR) can help significantly in learning tasks when the objective is to learn human behaviors and actions. The usefulness of context in machine learning is particularly brought out by the research of Stein discussed

above. While there have been many published works about learning from observation (too many to review here) and even more publications about contextual reasoning, only the above works combine the two. See Johnson [2014] for a more extensive discussion of context and how it can assist in machine learning.

4. Command and Control Agents

The third major area of application to be discussed here entails representation of command and control agents. These agents, as mentioned above, determine a high level course of action and issue commands to subordinates to execute those actions, as opposed to doing them themselves. I note again that these need not be commanding a military operation, but could be management agents that manage a large-scale project that includes many collaborators. The behaviors of command and control agents can also be said to be tactical, as it meets all the criteria outlined in section 2 above for tactical behavior.

While we have not had the extensive experience in applying contextualization to command and control problems that we have had in using CxBR for (direct) tactical reasoning or machine learning from observation, the guiding concept is that managers/commanders also face situations, albeit at a higher level of abstraction. Unlike the actual performers of a task, the commanders/managers are not physically involved in the tasks; however, they control those that directly perform the tasks, and probably have just as much at stake. Like the performers, they also face different situations that can be likened to contexts, and there may be accepted ways of handling such situations/contexts once they are recognized and correctly identified.

Our work on this subject entailed managing a design and construction project for a sounding rocket. Of course, this was an imaginary project that made several artificial assumptions about the rocket's specifications, cost and delivery requirements. Nevertheless, it provided a good testbed to evaluate our basic ideas. See Gonzalez et al., [2010] for details about this work. However, here is an abridged description of that work.

We built a project manager agent (called *PM-Agent*) that was charged with ensuring that the design of the rocket be done according to the customer specifications, on time and within budget. The specifications, the timeline and the budget are reflected in the Mission Context. The authors specifically made use of a NASA simulation system [NASA, 2003] that can simulate the launch of a specifically-designed rocket, and determine whether the rocket as so-designed will be successful in reaching the specified altitude while hauling the specified payload before falling back to the ground.

The design of the rocket in this application was actually more like a specification of values for several attributes. The NASA program required that the following components be designed (specified):

- Engine and body material
- Structure of rocket, including the number of stages and the thickness of the fins
- Control System

The design decisions were made by *design agents* (DAs) that, with their experience and knowledge, could make the best decision for the stated requirements. Design agents could be either human or software. In their work, the authors assumed that they were human.

PM-Agent has access to all "postings" of design activity made by the various human design agents onto a *Global Fact Base*. The PM-Agent reviews and interprets the designs of the various DA collaborators in the project and determines in which context the overall project finds itself. In cases where it needs expertise to make these decisions, PM-Agent can rely on *Auxiliary Agents* (called, creatively enough, AAs). These can also be human or software, but were software for this work.

The authors define three Design Agents and five Auxiliary Agents. The DAs contributed the design of the engine, the material of the rocket body, the design of the rocket structure, and the selection of an appropriate control system. Each of these design agents had to decide on a limited number of design parameters, each of which has relatively few possible choices. These agents are the Engine Design Agent (DA-1), the Structural Design Agent (DA-2), and the Control Design Agent (DA-3).

The five Auxiliary Agents included the NASA RocketModeler simulator (AA-1), a Scheduling Agent (AA-2) and a Cost Agent (AA-3). AA-1 (the rocket simulator) determines the technical validity of the rocket design. It does this by incorporating the design choices into a simulated rocket that is virtually launched. If the simulated rocket attains the specified altitude with the appropriate payload, it is considered to be an acceptable design. AA-2 determines whether any delays are expected as a result of a design or of an external event. The cost agent (AA-3) is in charge of determining the financial impact of design decisions and/or of delays on the project cost. In addition to the main AAs defined above, AA-0 will compute the initial cost and schedule plans as well as maintain the official schedule and cost estimates. PM-Agent could only make changes to the expected cost or the schedule through AA-0. Lastly, AA-4 was considered to be a problem solver loosely based on constraint-based algorithms that when given all the constraints involved in making decisions, it selects the values for the various variables that meet the criteria defined as constraints. If the system is over-constrained (i.e., no solution is possible), AA-4 informed PM-Agent of this. These AAs are called by PM-Agent when it deemed necessary. While DAs are human for this experiment, the AAs are software agents in their prototype. However, only the PM-Agent was based on CxBR.

Figure 4 illustrates the conceptual relations between the agents that participate in their prototype. All events related to the project were posted in the Global Fact Base. PM-Agent need only monitor the GFB to learn what new events affect the project.

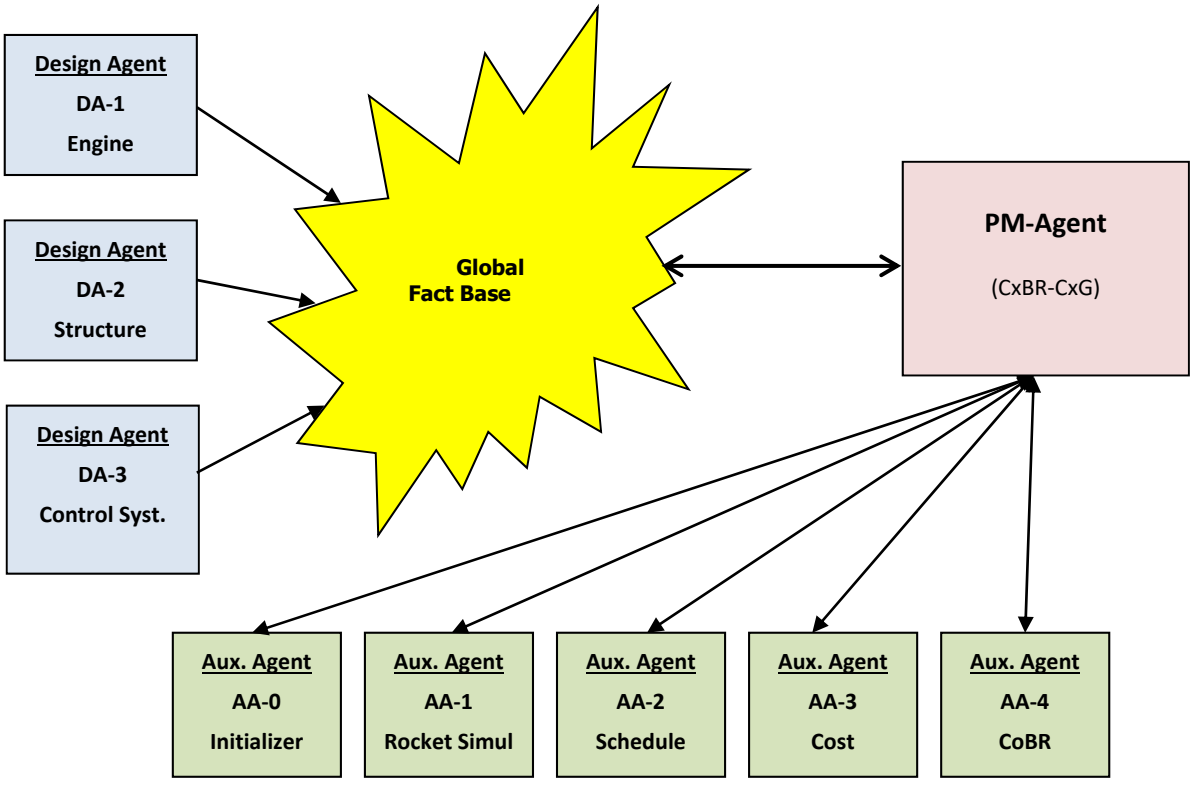

Figure 4. *Conceptual Block Diagram of Participating Agents in Our Prototype*

PM-Agent examines the GFB on a regular basis (e.g., once per day). However, for the purpose of practicality in demonstrations, the authors set it to once every few seconds to simulate several months in the space of a 10 minute simulation. Upon retrieving the relevant information from the GFB, PM-

Agent assessed the situation. Its transition rules decided which new major context should become active, if any. If different from the currently-active major context, this newly-selected major context becomes activated and the current one de-activated. The major context contains the action appropriate for the context and executed this action through the execution framework. It will continue to do so until the situation is resolved, or until the next sampling cycle if there are no problems to address.

PM-Agent had at its disposal two corrective action operators that can take corrective measures when the situation became conflictive (i.e., the budget and/or the delivery date is/are exceeded). In a robust, real-world application one would expect that there be many more such corrective action operators. Those used in this work were the following:

– **Corrective action operator #1**: Costs could be reduced by trimming staff. This could only be done twice during the lifetime of the project.

- The first time, reduction of up to 10% of total projected cost
- The second time, reduction of up to 5% of total projected cost

– **Corrective action operator #2:** Component delivery time could be reduced by paying overtime to workers to accelerate delivery

- The relation is set to $500/day of reduced delivery time.
- Can only be done for a maximum total reduction of 30 days in the life of the project being simulated.

When faced with a conflict to resolve, agent AA-4 first attempted to find a solution using either the above corrective action operators, or through a design change. If all attempts to solve the problem failed, then an impasse was declared by PM-Agent and the **Impasse** major context became activated. This indicated that the situation had become over-constrained, and turned the process over to a higher authority to either 1) authorize an extension to the project deadline, 2) increase the maximum allowable cost (the budget), 3) mandate a design change on the DAs that results in lower cost and/or faster delivery, or 4) declare the project to have ended in failure. These all required human involvement.

Each major context reflected a situation that PM-Agent faced and must resolve. The major contexts contained functionality to do what PM-Agent could do to return the project to normal status. The major contexts used in this application were:

– Normal
– DesignChange
– ExternalEvent
– Impasse

These contexts introduce expectations as well as procedures on how to act in these situations.

Normal major context**:** If all is normal, then the context is **Normal**. The authors define normalcy to be that the specifications have been met by the current design, the rocket is projected to be delivered on time and within budget, and no evidence to the contrary exists in the GFB. In other words, **Normal** can be defined to be the absence of any problems. In this major context, PM-Agent continues to monitor the situation every computation cycle, or if that is too frequent, then every **n** computation cycles. Sampling the situation once per day is thought to be sufficient for this prototype. The **Normal** major context is also the default context as well as the initial context. The simulation always begins with **Normal** being active. Furthermore, when nothing else emerges as a problem, PM-Agent returns to this context. When **Normal** acts as the initial context, PM-Agent calls agent AA-0 as the active sub-context to initialize the schedule data structure and calculate the projected final cost of the project. A design

change or an external event that reports a change in delivery time or increase in cost can cause the project to become not normal. Therefore, upon detecting either a design change or a new external event in the GFB, PM-Agent will transition, respectively, to **DesignChange** or **ExternalEvent** to examine and determine the effect of the new situation.

DesignChange Major Context**:** When a design change is posted into the GFB, PM-Agent transitions into this major context, where it deals with a design change and its effect on the project. Within this major context, the authors define a minor context called CxG-1 that steps through a process to determine whether there are any ill effects from this design change and resolve the problem if there are. CxG-1 determines the effect of the design change and its effect on the cost and schedule. If it discovers any problems, it will seek to resolve them through AA-4.

ExternalEvent Major Context: This major context is activated by PM-Agent when there is a new external event posted in the GFB. By definition, external events only affect the schedule and cost of the project, and not the design. When activated, **ExternalEvent** major context executes the minor context CxG-2. This minor context first calls AA-2 to determine whether there is any negative effect on the schedule. If not, then it calls AA-3 to see whether there is any negative effect on the cost. If neither of these returns indication of negative effects, then the **Normal** major context is re-activated. If, on the other hand, either one returns indication of negative impact, then AA-4, the problem-solver agent, is called to arrive at an acceptable solution. If system is not over-constrained and an acceptable solution can be found by exercising the corrective action operators, then this is done and the **Normal** major context is re-activated with these new values. However, if the situation is found to be over-constrained, then the **Impasse** major context is activated.

Impasse Major Context: This major context is activated by PM-Agent when the situation is over-constrained. In this major context, an extension of the mission timeline or increase in the authorized budget is required in order to proceed. Otherwise, the project must be cancelled. The **Impasse** major context can also become activated if a design conflict is irresolvable within the PM-Agent's range of authority. Little action is executed in **Impasse**, other than a call to upper management for their intervention. The minor context CxG-3 reflects this logic.

The system was tested extensively by the authors. Their tests introduced several realistic situations in a fictional project that would have to be resolved by a human project manager. Their focus was to see whether, given the knowledge and functionality found in the contexts provided to the PM-Agent, the latter could address the situation in a way that a human project manager would do. While the authors admit that it does not represent a real-world case study application to an actual project, it did subject the prototype to realistic situations normally faced by a PM. The decisions made by PM-Agent were compared to those made by a human project manager under the same circumstances. If the same or similar, the test was deemed a success; otherwise, it was deemed a failure.

The tests cases consisted of eight relatively easy (but not trivial) tests, five more of increased difficulty but within the expected capabilities of PM-Agent, and finally, two that presented situations well beyond PM-Agent's capabilities. All but the last two were successfully addressed by PM-Agent. The last two were not, as expected by the authors, who claimed overall success in the project. The last two failures did provide them with insights into the operation of PM-Agent and its context-based structure.

The underlying basis of this work was that a tool able to enhance a manager's *situational awareness* would be an advantageous asset for a manager to have. Situational awareness [Endsley, 2000] is the idea that one should quickly and accurately become aware of the situation (context) in which one is placed. Such quick and effective awareness of the situation can make a manager much more effective in handling emergencies, especially in situations where events happen quickly and decisions to address

these events must be made equally quickly (e.g., military operations, firefighting, law enforcement, and team games among many others).

The authors state that the research succeeded in creating a project manager agent that was situationally-aware. It was able to see the external events and design changes and apply whatever corrective action operators existed to bring the project design, cost and schedule within original objectives, if possible. If the problem was over-constrained, then it declared an impasse and called for external intervention on the part of the customer or upper management. In general, it was deemed successful.

5. Notable Applications of Context-based Reasoning by Others

A few noteworthy applications of CxBR have been done by researchers outside of our laboratory. Two of these have been in simulation-based control; one in smart buildings; and the most notable one, in a self-driving automobile that participated in DARPAs Urban Challenge competition in 2007. This entry finished in 7th place in that competition. This section briefly describes these applications.

5.1. *Contextualizing the Environment Controls in Smart Buildings*

Diagnosis and control of complex electrical/mechanical systems can often require deep expertise to manage them well. This is normally the case for environmental control in large buildings where temperature and humidity of the ambient air needs to be adequately controlled without needless use of energy. For example, a computer server room will need to be maintained at a cooler temperature than other parts of the building at all times. A restaurant kitchen in the same building will require a larger cooling effort because of the heat being generated in the cooking process, but only when it is open for business. Therefore, maintaining the appropriate temperatures throughout a multi-purpose building is important as well as difficult to do. The use of context can be helpful in this regard, as it can set the temperature and humidity requirements for each of the rooms, depending on its context of the room and/or of the entire building (e.g., an unoccupied school building on a weekend day requires less cooling (or heating) than during daytime on a school day).

This application merits mention here because it is quite an interesting and innovative application of CxBR. It is not an example of tactical reasoning as discussed above. Instead, it is a type of control behavior that can predict the needs of the building by its current and future context and adjust the heating, ventilation and air conditioning (HVAC) controls to suit the context of the building (or parts thereof).

Likewise, knowing the context of a system failure can also help identify its cause and provide repair resources quickly and efficiently. For example, a dead battery in a car that has spent the night outdoors in cold weather will have drastically different diagnostic implications than an equally dead battery that discharged while the car was being driven at highway speeds.

The authors of this project [Fazenda et al., 2012] used CxBR as the control scheme for a building environment control system. They define several blocks of knowledge that must be known a priori in order to identify the context in which the building or a room therein finds itself. These blocks of knowledge include 1) architectural features of the building, such as the location of the rooms with respect to the sun entering the building, the size of each room, he type of furniture, the wall and insulation material used in its construction; 2) Building systems (elevators, HVAC, sensors, actuators, etc.); 3) The function of the building (school, office building, factory, residence, hotel, etc.); 4) Activities of occupants (sedentary work, factory workers, surgery, active congregation (pep rally) or athletic event; and 5) Electric and gas rates if the cost of running a building needs to be controlled.

They also define other blocks of knowledge that must be acquired, mostly by running the building control system over a period of time to gather information and draw some conclusions with these data. In a way, one could make the argument (the authors do not) that this is akin to context-based machine learning.

Once the knowledge becomes available, the CxBR building control system (in the case, the ambient environment controls) is set up. Ambient environmental control is but one of many services s_i in S [Fazenda et al., 2012]. The control function for this service is

Control of $si = \Gamma(C^t active)$

Where $C^t active$ is the active context path that is currently controlling the building's ambient services. The term path is used by the authors because in this application, all the knowledge in the context hierarchy path from root to leaf is included in the applicable knowledge required to run the control system. While the authors do not report putting such a context-based smart building control framework in actual practice, they state that such a framework should be built and leave it as a future task.

5.2. *Helicopter Control Model*

Hu and Liu [2007] used CxBR to build a model of ASW helicopter control. The authors claim that decision trees and queuing theory are not applicable to this domain because of the asynchronous and random nature of events that happen in helicopter ASW. The authors use the basic concepts of CxBR but add an atomic action level. This is the lowest level sub-context that prescribes an atomic action that cannot be further sub-divided (e.g., move forward). The original definition of CxBR certainly allows for such atomic action contexts but does not explicitly define them.

The authors use the traditional CxBR hierarchy, but they use the slightly different labels of *task contexts* and *sub-task contexts* to refer to major and minor contexts. Another modification they introduce to the original CxBR is how they structure their transition rules. Their transition rules can be matched either exactly or uncertainly. An exact rule is defined in their paper by the traditional first order predicate logic calculus rules. This is a slight and unimportant modification of the transition rule concept espoused in the original CxBR, where less rigid production rules were used as transition rules. Where the authors' work differs from the original CxBR definition is that if an exact match is not relevant when transitioning to another major context, neural networks are used to assist in matching rules inexactly. The use of neural networks (or any other technique, for that matter) is certainly supported in the highly flexible definition of CxBR. This allows for the generalization capability of neural networks to come into play when deciding when and to what other major context to transition the control of the agent. However, the structure, the type, the training and specific implementation of these neural networks are not explained by the authors. Nevertheless, the authors report a prototype simulation system, but no test results are included in their paper.

Figure 5 below is recreated from their paper [Hu and Liu, 2007]:

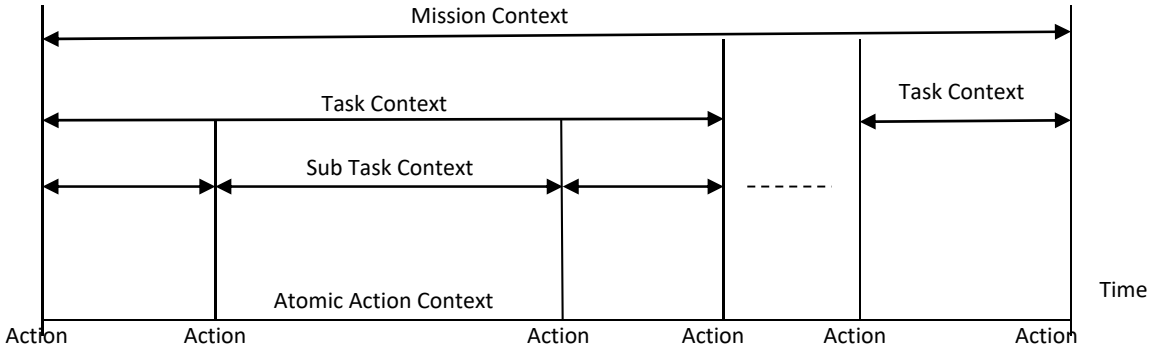

Figure 5. *CxBR hierarchy as described in Hu and Liu [2007] (Recreated from Hu and Liu [2007] without permission)*

5.3. *Use of CxBR in Autonomous Automobiles – The DARPA Urban Grand Challenge*

Another notable application of CxBR, this time in the physical world, was implemented by Patz et al. [2008] as the control structure for the University of Central Florida entry in the DARPA Urban Grand Challenge competition held in late 2007. The car entered by the University of Central Florida College of Engineering and Computer Science, called the Knight Rider, successfully made it past the early elimination rounds over the course of 18 months to make the semi-finalist field of 35 competing teams. The 35 semi-finalists convened in Victorville, CA, at the site of a de-commissioned military base during the last two weeks of October 2007. The 35 contestant teams were put through a series of qualifying tests. From these tests, the field of 35 was reduced to 11 finalist teams. The UCF Knight Rider team was one of these 11 finalists. The final on-site competition was held on November 3, 2007 at the same site and the winners were announced the next day.

As per Patz et al, "The overall urban driving objective as defined by DARPA was to demonstrate an autonomous robot's ability to complete a series of driving missions in traffic, over the course of 6 h, while obeying California driving rules, utilizing a moderate level of a priori information associated with the road network, but being expected to deduce any missing information."

The robot automobile was highly complex and several software modules were used by the authors to implement its autonomy: *laser data processing, vision data processing, sensor fusion, intelligence, planning,* and *control* modules. Context-based Reasoning was utilized in the intelligence module, and within that, in the *tactical* component of the intelligence block diagram, also called the *core AI* module. The authors combined the higher level functional decomposition feature of CxBR with a hierarchical state machine that was able to provide a step-by-step sequence of actions to be executed within each context. This hybrid framework proved very successful in providing the tactical high level control of the Knight Rider the autonomous vehicle during the final event. Unfortunately, a GPS system malfunction caused the car to be the fifth vehicle retired from competition, thus placing it in 7th place out of the many contestants that participated. See Patz et al. [2008] for details of this project.

5.4. *Use of CxBR for Simulated Military Command and Control*

The last external work to be described involves an application to military command and control. This investigation and resulting prototype was authored by Lovlid et al [2017]. In their work, the authors built a CxBR system that reflected the expertise of a battalion commander on a (simulated) military mission. In this paper, the authors apply this idea in a hierarchical multi-agent system of command agents, where the agents' actions are to command and coordinate subordinates, send reports to their superiors, and communicate with other agents at the same level. The work focuses on how contexts and actions can be defined for these higher level command agents and how the contexts and

actions for the different command agents are related. The proposed method is implemented in a prototype and tested for a hierarchy of command agents that are interpreting and planning an operational order at a battalion level and carrying it out in a computer generated forces environment.

The prototype multi-agent system incorporates a hierarchy of command agents, where one agent is created for each military unit in the simulated confrontation. Each agent reflects the behavior of the commanding office of that unit and his staff. A task is examined by a command agent upon receipt, and it is decomposed into sub-tasks, which are then assigned to its subordinates, just as the original task was handed down to it by its superior officer. At the lowest level, the command agents that occupy these positions assign the appropriate sub-tasks to units under its command. These bottom level units then actually carry out the action. However, these action units are in a commercial CGF system, which then execute the tasks in the simulation environment.

The tests were carried out as a series of simulated military operations. The authors pre-determined what output the system should correctly provide and compared that with the prototype's actual output for the same set of conditions. They found that in all cases, the prototype produced the expected results. See Lovlid et al [2017] for details on this work.

6. Conclusion

It has been known in the context research community that context can play a very important role in how the real world and human actions in it are modeled – from representation of basic situations and actions for tactical agents to assisting with learning these situations and actions automatically from observation of human performance. This paper reviewed and discussed how context (specifically, Context-based Reasoning) was used to address three general problems in AI research: a) tactical agent behavior; b) context-based machine learning from observation; and, c) context-based command and control behaviors. The first and third problems are similar in that task performers and those who might direct the task performers as part of a more complex task have several things in common. The second problem involves learning what humans typically learn – how to do something – and learn it by observing other humans perform the task or actions. The paper focuses on the work done in the author's research lab, plus some a few notable applications of CxBR by researchers un-connected to our research group.

References

Antunes, B., Furtado, B. and Gomes, P. "Context-Based Search, Recommendation and Browsing in Software Development". In Brezillon, P. and Gonzalez, A.J. (Eds.), Context in Computing – A Cross-Disciplinary Approach for Modeling the Real World, New York: Springer. 2014. 45-62.

Brezillon, P. and Gonzalez, A.J. (Eds.). Context in Computing – A Cross-Disciplinary Approach for Modeling the Real World, New York: Springer. December 2014.

Brézillon, P. "Representation of Procedures and Practices in Contextual Graphs". *The Knowledge Engineering Review*, 2004. 18(2): 147-174

Brown, J.C. (1994) "Application and Evaluation of the Context-based Reasoning Paradigm". Master's Thesis, Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL, July, 1994

Clancy, T., Red Storm Rising, New York: Berkley Books, 1985.

Clancy, T., The Hunt for Red October, New York: Berkley Books, 1986.

Clancy, T., Submarine: A Guided Tour Inside a Nuclear Warship, New York: Berkley Books, 1993.

Dey, A.K. "Understanding and Using Context". *Personal and Ubiquitous Computing.* 2001. 5(1): 4-7.

Endsley, M.R. "Theoretical Underpinnings of Situational Awareness: A Critical Review". In Endsley, M.R. & Garland, D.J, (Eds.), Situation Awareness and Measurements New York: Lawrence Erlbaum and Associates. 2000.

Fazenda, P., Carreira, P. & Lima, P. (2012). "Context-based reasoning in smart buildings". *Proceedings of the First International Workshop on Information Technology for Energy Applications.* 923: 131-142.

Fernlund, H., Gonzalez, A.J., Georgiopoulos, M. and DeMara, R.F. "Learning Tactical Human Behavior through Observation of Human Performance". *IEEE Transactions on Systems, Man and Cybernetics – part B*. 2006. 36(1): 128-140

Gonzalez, A.J., R.L. Osborne, C.T. Kemper and S. Lowenfeld. (1986) On-line diagnosis of turbine-generators using artificial intelligence. *IEEE Transactions on Energy Conversion*, EC-1(2), 68-74

Gonzalez, A.J. and Ahlers, R.H. "Context-based Representation of Intelligent Behavior in Training Simulations". *Trans. of the Soc. of Computer Simulation*. 1998. 15(4): 153-166.

Gonzalez, A. J., Georgiopoulos, M. and DeMara, R. F. "Maintaining Coherence among Entities' States in a Distributed Multi-Agent System". *Journal of Defense Modeling and Simulation (JDMS)*. 2007. 4(2): 147-172.

Gonzalez, A.J., Stensrud, B.S. and Barrett, G. "Formalizing Context-Based Reasoning - A Modeling Paradigm for Representing Tactical Human Behavior", *International Journal of Intelligent Systems*. 2008. 23(7): 822-847.

Gonzalez, A. J., Tsuruta, S., Sakurai, Y., Nguyen, J.V., Takada, K. and Uchida, K. "Using Contexts to Supervise a Collaborative Process*", International Journal of Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM)*. October 2010. 25(1): 25-40

Gonzalez, F. G., Grejs, P. and Gonzalez, A. J. "Autonomous Automobile Behavior through Context-based Reasoning". *Proceedings of the International FLAIRS Conference*, Orlando, FL. 2000.

Gundersen, O.E. "The Role of Context and its Elements in Situation Assessment". In Brezillon and Gonzalez (Eds.) Context in Computing: A Cross-disciplinary Approach for Modeling the Real World, New York: Springer. 2014. 343-358

Hu, Z-t. and Liu, L. (2007) "The research of ASW helicopter ACGF construction based on CXBR." *International Conference on Computational Intelligence and Security Workshops, 2007 - CISW 2007.* pp. 132-135.

Hung, V. "Context and NLP". In Brezillon and Gonzalez (Eds.) Context in Computing: A Cross-disciplinary Approach for Modeling the Real World, New York: Springer. 2014. 143-154.

Johnson, C.L. and Gonzalez, A.J. "Learning Collaborative Behavior by Observation", *Expert Systems with Applications*. 2014. 41: 2316–2328

Johnson, C.L., "Context in Machine Learning". In Brezillon and Gonzalez (Eds.) Context in Computing: A Cross-disciplinary Approach for Modeling the Real World, New York:Springer. 2014. 113-126.

Kalatzis, N., Roussaki, I., Liampotis. N., Kosmides, P., Papaiaoannou, I. and Anagnostou, M. "Context and Community Awareness in Support of User Intent Prediction". In Brezillon and Gonzalez (Eds.) Context in Computing: A Cross-disciplinary Approach for Modeling the Real World, New York: Springer. 2014. 187-204.

Khboshch, V. A., Submarine Tactics, Voyenizdat, 1989, translation in 1990 by the Foreign Broadcast Information Service.

Liampotis. N., Roussaki, I., Kalatzis, N., Papadopoulou, E., Goncalves, J.M., Papaiaoannou, I. and Sykas, E. "Context-Sensitive Trust Evaluation in Cooperating Smart Spaces". In Brezillon and Gonzalez (Eds.) Context in Computing: A Cross-disciplinary Approach for Modeling the Real World, New York: Springer. 2014. 187-204.

Løvlid, R.A., Bruvoll, S., Brathen, K. and Gonzalez, A.J. "Modeling the Behavior of a Hierarchy of Command Agents with Context-based Reasoning", *The Journal of Defense Modeling and Simulation: Applications, Methodology and Technology*. DOI: 10.1177/1548512917702832 | First Published on-line on April 7, 2017

NASA - National Aeronautics and Space Administration. (2003). RocketModeler. version 1.2 – undated.

McDermott, J. (1982) R1: A Rule-based Configurer of Computer Systems. *Artificial Intelligence*. 19(1), 39-88.

Parker, J.E. and Hollister, D.L. "The Cognitive Science Basis for Context." in Brezillon and Gonzalez (Eds.) Context in Computing: A Cross-disciplinary Approach for Modeling the Real World, New York: Springer. 2014. 205-219.

Patz BJ, Papelis Y, Pillat R et al. A practical approach to robotic design for the DARPA urban challenge. Journal of Field Robotics 2008; 25: 528–566.

Saeki, S. and Gonzalez, A. J.*,* "The Competing Context Concept for CGF's: Experimental Results", *Proceedings of the Inter-service/Industry Training Systems and Education Conference* (I/ITSEC), Orlando, December 2000

Sidani, T.A. “Learning Situational Knowledge through Observation of Expert Performance in a Simulation-based Environment”, doctoral dissertation, Computer Engineering, University of Central Florida, 1994.

Sidani, T.A. and Gonzalez, A.J. “A Framework for Learning Implicit Expert Knowledge through Observation”. *Transactions of the Society for Computer Simulation*. 2000. 17(2): 54-72

Stensrud, B. S. and Gonzalez, A. J., “Discovery of High-Level Behavior from Observation of Human Performance in a Strategic Game”, *IEEE Transactions on Systems, Man and Cybernetics, Part B*. June 2008. 38(3): 855-874.

Stein, G., and Gonzalez, A.J., ”Building high-performing human-like tactical agents through observation and experience”. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*. 41(3), 792-804. 2011

Stein, G. and Gonzalez, A.J. “Learning in Context: Enhancing Machine Learning with Context-Based Reasoning”, *Applied Intelligence.* 2014. 41(3): 709-724.

Thorndike, P.W. and Wescourt, K.T. “Modeling Time-stressed Situation Assessment and Planning for Intelligent Opponent Simulation. Technical Report PPAFTR-1124-84-1. 1984. Office of Naval Research.

Trinh, V.C. and Gonzalez, A.J. “Identifying Contexts from Observed Human Performance”. *IEEE Transactions on Human Machine Systems*. 2013. 43(4): 359-370.

Turner, R.M. “Context-Mediated Behavior for Intelligent Agents”, *Int’l J. of Human Computer Studies*. 1998. 48(3): 307-330.

Wienhofen, L.W.M., Preuveneers, D., Toussaint, P.J. and Berbers, Y. ”Event Quality Awareness for Contextualized Decision Support in e-Health Applications”. In Brezillon and Gonzalez (Eds.) Context in Computing: A Cross-disciplinary Approach for Modeling the Real World, New York: Springer. 2014. 237-254.

