Proceedings of the 2016 Winter Simulation Conference
T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, eds.

CASL: A DECLARATIVE DOMAIN SPECIFIC LANGUAGE FOR MODELING COMPLEX ADAPTIVE SYSTEMS

Lachlan Birdsey
Claudia Szabo
Katrina Falkner

School of Computer Science
The University of Adelaide
Adelaide, AUSTRALIA

ABSTRACT

Complex adaptive systems (CAS) are ubiquitous across many domains, such as social networks, supply chains, and smart cities. Currently, the modeling and analysis of CAS relies on adapting techniques used for multi-agent simulation, an approach which lacks several features crucial to CAS modeling, such as agents comprised of other agents, and considering methods for adaptation. Moreover, many existing approaches do not scale well, thus making them difficult to employ in analyzing realistic scenarios. In this paper, we propose the Complex Adaptive System Language (CASL), a declarative language that is able to capture the salient features of CAS while being general enough to be used across multiple domains. CASL facilitates the construction of complex models and our code generation method allows CASL models to be executed on a variety of platforms. We demonstrate the flexibility of CASL by implementing three distinct models, which are then executed using Repast.

1 INTRODUCTION

Complex systems are comprised of many autonomous and interconnected entities, whose interactions can lead to unexpected and emergent properties (Szabo et al. 2014, Mittal 2013, Chan and Macal 2010). Complex adaptive systems are a type of complex system where entities and the environment are encouraged to adapt and interact with each other in order to achieve desired properties (Holland 2006) and provide a more realistic abstraction of real-life scenarios (North et al. 2013, Niazi 2013). Complex adaptive systems have become ubiquitous in domains such as social networks, supply chains, health-care networks, smart-cities and smart-grids, the "Internet of Things", and the Internet itself (Niazi 2013, North et al. 2013, Hasgall 2013, Benham-Hutchins and Clancy 2010). When systems in these domains are modeled as CAS, the importance of entity interaction, adaptation, and influence from the operating environment is highlighted, enabling deeper study focusing on individuals and how their behaviors contribute to higher level properties (Holland 2006, Mittal 2013, Özmen et al. 2013).

Complex adaptive systems must support the concepts of adaptation, modularity, and diversity and contain an environment(s) (Holland 2006). Environments in a complex adaptive system must contain entities and be able to interact with them (Holland 2006). Environments can then act as communication mediums that allow for stigmergical interactions, system wide events, and self-organization. The *adaptation* present in a complex adaptive system refers not only to the individual adaptive processes of entities and environments, but to the adaptive ability of the system as a whole (Holland 1992). *Modularity* in complex adaptive systems dictates that at least one entity or environment must be comprised of sub-entities (Özmen et al. 2013, Holland 2006). These sub-entities process inputs to determine the parent entity's behaviors and actions.

Diversity in a complex adaptive system is crucial as a variety of entities interacting with distinct behaviors and actions can lead to co-adaptation, self-organization, and more accurate results generation (Özmen et al. 2013, Holland 2006, North et al. 2013). We define a complex adaptive system as a system that contains a large number of diverse entities that may have the ability to adapt and/or are comprised of multiple sub-entities. Each agent should be able to interact with the environment it is contained within, and the system should execute for a long period of time to highlight aggregate properties or system level adaptation.

Current attempts to model CAS fall into two main categories, general or domain specific. Each of these provides their own set of challenges. The attempts that rely on a general approach, such as adapting a multi-agent system paradigm, tend to suffer from scaling issues, while also not providing a solid grounding to model individual agent and environment adaptations. Furthermore, they can also only consider a single form of agent representation such as a network or geographical information system, which hinders attempts at creating highly detailed realistic CAS models. Domain specific approaches suffer from fewer scaling issues and are able to provide high-quality representations of their respective entities such as accurate adaptation methods. However, these techniques, amongst others, are unable to be adapted to other domains.

In this paper, we propose a new approach for the modeling and analysis of complex adaptive systems. Our approach relies on a declarative domain-specific language with constraints components from which simulation code is generated, executed, and subsequently analyzed. We advance the Complex Adaptive Systems Language (CASL), which leverages the definition of CAS by providing distinct components which form together to provide a precise definition of an entity. CASL provides features targeted towards the modeling of complex adaptive systems, such as specific adaptation components. Furthermore, by containing constraint components, models constructed using CASL are ensured to be complex adaptive systems models, while also being efficient to create and execute. We demonstrate the use of CASL to implement three models, namely, the Game of Life, a social network, and an emergency department. With our Game of Life and social network models, we highlight the scale capabilities of CASL. For our emergency department model, we highlight the explicit modularity and adaptation capabilities that CASL incorporates.

2 RELATED WORK

As CAS become an increasingly important field of study, tools, methodologies, and frameworks are needed to create models and simulations of CAS. There are three main paradigms for the modeling and simulation of CAS, namely, complex networks, discrete-event simulation, and agent-based modeling. Complex networks are able to capture the interactions between entities and exemplify the massive scale at which some systems operate, including scale-free networks (Niazi 2013, Mittal 2013). However, complex networks can only represent homogeneous entities, e.g. entities that have different parameters but share the same behaviors. Discrete-event simulation is able to capture the heterogeneity, scale, and a top-down view of the system during execution by using system states, as well as behavior evolution of the entities (Mittal 2013, Chan and Macal 2010, Banks et al. 2005). Discrete-event simulation is not entirely suitable for CAS modeling as it requires events, transitions, and states to be defined prior to execution, which is highly susceptible to state space explosion, even with a moderately sized CAS. While agent-based models are able to capture the heterogeneity, interactions, simplicity of entities, and non-linearity of a CAS, they cannot easily accomplish ultra-large scale simulations, and it is non-trivial to provide a top-down view of the system (Niazi 2013, Mittal 2013, North et al. 2013).

Although several authors have modeled CAS using existing modeling software such as Repast (North et al. 2013) and NetLogo (Tisue and Wilensky 2004), no specific modeling tools have been developed. North et al. (2013) provide a series of methods to use Repast for CAS modeling and simulation. Niazi (Niazi 2013) outlines how two existing paradigms, namely, complex networks, and agent-based modeling, can be used to model CAS in particular fields such as social sciences, biology and computer science. Several

attempts at modeling domain specific CAS have been achieved by leveraging existing frameworks. Niazi and Hussain (2011) developed a model of a wireless sensor network designed to be used inside a CAS model, using NetLogo for both the CAS and wireless sensor model. Özmen et al. (2013) utilize ABM to model a collaboration network. Cioffi-Revilla et al. (2012) model a complex adaptive social system. They achieve this by creating evolving agents and deploying them to an existing model, and defining evolutionary parameters suited to the model. Boulaire et al. (2015) designed a tool to handle large-scale agent based simulation that utilizes agents that are both modular and are involved in many interactions. Works from Mittal and Holland have focused on modeling CAS using more theoretical approaches. Mittal (2013) enhances existing discrete-event simulation formalisms to model and simulate CAS. Holland (1992) proposed a language, Echo, to model CAS. Smith and Bedau (2000) determine that Echo is not capable of modeling CAS correctly as it cannot capture the diversity of hierarchically organized aggregates and therefore is unable to display emergent properties. Although not tools specifically for modeling CAS, Aydt et al. (2012) and Picone et al. (2012) designed tools for modeling and simulation aspects of smart-cities. Aydt et al. (2012) designed a simulation of a smart-traffic system capable of using external data sources. Picone et al. (2012) constructed a discrete-event simulation that focuses on the differing scales of sub-systems in a smart-city by considering how mobile entities affect the overall traffic. Raunak and Osterweil (2013) designed tools for modeling and simulating emergency departments using discrete-event simulation that focuses on constraints and resource requirements of entities.

Several languages have been created in the modeling and simulation community. Nanoverse (Borenstein 2015) is a high-level constraint-based language for biological cell simulation, that relies on agents existing in a singular environment. The generated Nanoverse code is designed for use in the Nanoverse framework. While Nanoverse requires very little code to construct a full model, it is bound to biological simulation and only considers simple agents. A more domain agnostic modeling language, 3APL (Dastani and Meyer 2005), provides model designers a DSL and a Java based framework to create multi-agent system models where the agents follow the Belief-Desire-Intention model of cognitive behavior. 3APL explicitly defines certain blocks such as `GOALBASE` and `CAPABILITIES`, which together form the total cognitive abilities of each agent. However, 3APL has strict limitations as each agent model is required to follow BDI logic and is unable to perform complex actions without the use of external plug-ins or Java calls. Another framework for developing multi-agent system simulations called JADE (Bellifemine et al. 1999) is more suited to complex adaptive system simulation as it is more flexible, as JADE is a Java library, and each agent can use a variety of different functionalities. Furthermore, JADE agents explicitly require behaviors to be defined, similar to 3APLs blocks. However, JADE suffers drastically from scaling issues as each JADE agent requires individual threads (Lorig et al. 2015). Repast and NetLogo provide languages for modeling based on the LOGO language, named ReLogo (Ozik et al. 2013) and NetLogo respectively. Both ReLogo and NetLogo are DSLs designed for agent-based modeling and they provide few restrictions beyond this goal. However, due to these languages being more general, the amount of code required for complex adaptive system modeling becomes quite substantial, even for simple models.

Our proposed language is more suitable for modeling complex adaptive systems in several respects. Firstly, CASL directly utilizes the definitions of complex adaptive systems to provide component blocks that directly implement each entity's adaptation, interaction, and modularity features, among others. Secondly, it allows the implementation of CAS conceptual models in a more straightforward way by providing a distinction between the key features of a CAS, such as adaptation processes. Thirdly, the code generated by our DSL takes into account scale issues for the target simulation software. Finally, despite being a DSL, CASL is a domain agnostic modeling language, as it is capable of being used to model any CAS.

3 PROPOSED APPROACH

This section presents an overview of our approach and modeling language. Our CAS modeling and analysis framework consists of several components, which include the CASL modeling tool and code generator, the simulator, and an observation tool. Our proposed language, CASL, provides constructs for designing models to capture the salient features of a CAS such as adaptation and modularity, along with other complex systems features such as interactions and behaviors.

3.1 Framework Overview

Our proposed framework consists of the CASL modeler, the code generator, a simulator, and an observation tool. Once a model is constructed in the CASL modeler, code is generated only if all the required constraints have been adhered to. The generated code is then executed in the simulator, which may require initialization parameters which can be provided by a configuration XML file. The observation tool is comprised of several modules that analyze various features of a CAS such as aggregation, runtimes, interactions, and domain-specific features. The observation tool is designed to be extensible to allow for new metrics to be added, that may either be designed specifically for the current simulation or for a more domain-agnostic purpose. Figure 1 presents a diagram of our framework architecture.

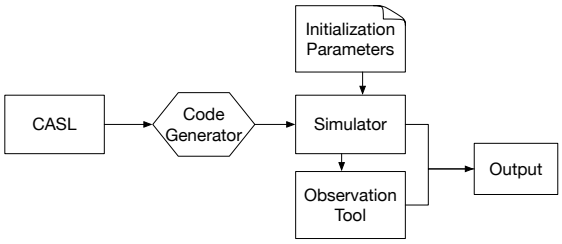

Figure 1: Framework Overview.

3.2 CASL

Constructing models of complex adaptive systems is a sizable task, even for systems with unsophisticated entities. Furthermore, CAS models tend to contain large numbers of entities which requires simulation software that can scale satisfactory. Typically, model designers must use a fully featured development library such as Repast or MASON (Balan et al. 2003) in conjunction with a general purpose language like Java or C++, or use a domain specific language (DSL) such as 3APL or JADE. Neither of these approaches are ideal for complex adaptive system modeling, as the former requires the CAS model to be constructed using the rules of the library, while the latter constricts the model designer to follow a strict set of rules which may also lead to a model that doesn't fulfill the criteria of being a CAS.

CASL aims to heavily reduce the amount of system expert effort required to model a system, while also producing code that is tailored to the target simulation software, and increase the readability of the code. CASL itself is a declarative DSL with constraint activated components. These constraint activated components determine how the system and entities should be modeled, for example, if the CAS contains only system level adaptation by way of aggregation such as in the Flock of Birds model, explicit adaptation methods should not be allowed. By providing these constraints in CASL, models can be constructed in a way that focuses on how the entities and system affect each other.

In CASL, each system consists of multiple entities that are either agents or environments. Agents must exist in environments, while environments can either be unique or exist in other environments. This allows for agents (or other environments) to interact directly with their containing environment. CASL is divided into three sections, namely, a *SYSTEM* section, an *ENVIRONMENT* section, and an *AGENT* section. This

distinct separation is similar to blocks in 3APL and JADE. Figure 2 shows the relationship between the agent, environment, and the system sections.

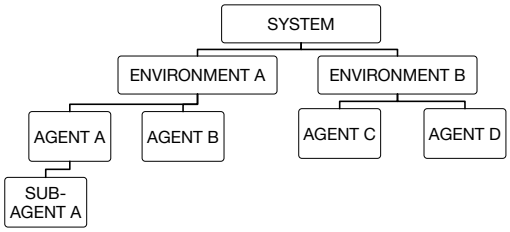

Figure 2: Example of a CASL model structure with five agents and two environments.

The *SYSTEM* section contains model initialization functions and system-level parameters such as initial system configurations, whether the system is open or closed along with how entities may enter or leave the system, and an enumeration of the agents and environments in the model, as well as simulation parameters such as termination conditions. The *SYSTEM* section also allows for modification of certain constraints, namely, *diversity*, *adaptation*, and *modularity*. The diversity constraint requires a heterogeneous system, i.e., multiple agent types. The adaptation constraint requires at least one entity to possess an adaptive process. Finally, the modularity constraint requires at least one entity to be constructed from other entities.

The *AGENT* and *ENVIRONMENT* sections allow for agents and environments to be defined respectively. Each agent and environment contains components to define parameters, functions, behaviors, interactions, adaptation processes, and subsystems. The use of these separate components allows us to define agents or environments as a combination of components, while also providing a clear distinction between the various features of a CAS entity. For example, an entity's behaviors may trigger different interactions, or an entity's sub-entities may trigger an adaptation. *Parameters* contain all the operating and state parameters of the entity, such as spatial position, life status, and busy status. The *functions* component contains the definition of initialization functions as well as any helper functions. In addition to the functions component, CASL also provides macros to access common simulation functions such as random number generation, calculating an entities nearest neighbors, and list filtering among others.

3.2.1 Defining Entity Behaviors

The behaviors, interactions, adaptation, and subsystems components are the most crucial as they define the entity life-cycle in the overall model. Each of these components can be triggered by an input, time, an interaction, after a state change, or instantly from another component. By using any combination of these components, any entity in a complex adaptive system can be implemented. The behaviors component describes an entities' behaviors, where a behavior is a process that may affect only the entity calling it. For example, in the Game of Life model, where a Cell should change its life state, or in the Flock of Birds model, where a Bird should move to in the next step. Behaviors can never be in the form of an interaction, however, they may trigger a particular interaction. Each behavior contains rules for a behavior type, a trigger time, optional input parameters, as well as the behavior action itself. A behavior type determines if the behavior contains no interactions, an interaction to an agent, an environment, or both. This allows the designer to instantly verify the logic of the particular logic. The trigger for a behavior determines when the behavior action should be performed, either instantly, at the end of the step, or in a number of steps. The optional inputs for a behavior may be used if the behavior requires further information, such as a value acquired from an interaction. For example, the behavior rules `[SELF][DELAYED]` indicate that the behavior will only affect the entity itself and that it will be performed at the end of the current step.

3.2.2 Defining Entity Interactions

The interactions component is the most crucial for the entities in a CAS model as it describes ways that an entity can interact with others. Interactions between entities can be a *query* or a *communication*. *Query* interactions occur when one entity simply polls another for information by reading a state or value, without the other entity requiring knowledge of the interaction. *Communication* interactions occur when an entity interacts with another and passes a message or signal. All agents are required to have at least one interaction, while an environment can only have an interaction if the environment is of a physical type. Figure 3 shows the interaction component for a Cell in the Game of Life model. The interaction type is a query, as no other entities are required to acknowledge the interaction, and the interaction is triggered every one step. The final line of the component shows that the adaptive process, `changeLife`, is to be triggered.

```
interactions:{
  countAliveNeighbors[QUERY][STEP(1)](): {
    var List:neighborsList = CASL.GRID.GetNeighbors[Cell](1);
    var int:aliveNeighbors = CASL.COUNT[neighborsList](Alive);
    ADAPTATION.changeLife(aliveNeighbors);  };  };
```

Figure 3: Interaction component for a Cell in the Game of Life model.

3.2.3 Defining Entity Adaptation

The adaptation component describes the adaptive processes that the entity may undergo, where an adaptive process is a procedure that causes the state or certain parameters of an entity to change. While similar to a behavior, adaptive processes have the goal of optimizing some features to improve performance. These processes may be implicit, an evolutionary algorithm or genetic programming, or a cognitive approach. Implicit adaptation is used for imparting relatively simple adaptive processes such as determining the life state of a Cell in the Game of Life. Evolutionary adaptation allows for various evolutionary methods such as a genetic or evolutionary algorithm to be used as an adaptive process. Cognitive approaches such as the Belief-Desire-Intention model or a neural network can be implemented as an entities adaptive process. Furthermore, the adaptation component allows these processes to be driven by another model or a separate engine, such as MIDCA. By allowing adaptive processes to contain external features, entities in CASL can be implemented in the most suitable fashion. Each adaptive process contains rules for the type of adaptation, if there is a delay on the adaptation performing, optional input parameters, as well as the process itself. The type of adaptation determines what methods of adaptation the process is to use, such as implicit, evolutionary, or cognitive. In the case of the latter two types, CASL will check if a suitable package or process is being used. Adaptive processes can either be triggered instantly, or at the end of the simulation step. The optional inputs for an adaptive process may be used if the process requires further information such as values generated from a behavior. For example, the rules `[IMPLICIT][NONE]` indicate that the adaptive process relies on implicit adaptation and that there is no delay once it is triggered.

3.2.4 Defining Sub-entities

The subsystems component allow for definition of sub-entities that effects the entity, such as biological cells in an organism. The sub-entities can interact with the entity and other sub-entities in the same entity but cannot directly communicate with other entities. This allows for the creation of models where entities can be broken down into functional components or sub-processes which can enable deeper understanding of such entities. For example, a trader entity in a stock market simulation would be comprised of a variety of sub-processes that determine if the entity should make a trade or not. The *ENVIRONMENT* section has an additional component that allows the model designer to set the main attributes of the environment, namely whether the environment represents a physical environment, such as a building, or a virtual environment,

such as a wireless sensor network (Niazi and Hussain 2011). This distinction is important as it determines if the contained agents are stored according to a two or three-dimensional location, such as a position inside a building, or a less tangible representation such as the tie strength between users in a social network. In addition, each environment can be described as implicit or explicit, with the implicit representing an environment type similar to typical MAS simulation, and the explicit type requiring the environment to have interactions and behaviors. This differentiation is crucial as it determines if and how agents can interact with their containing environment, as well as if the simulator needs to consider the environment to be an entity that requires extra processing.

3.2.5 Implementation

CASL currently generates Java code for use in the Repast Simphony Suite (North et al. 2013). Repast Simphony is a widely used simulation tool that provides the basics of simulation, such as a time and schedule management, initialization settings, along with a simple visualizer, amongst other features. Repast Simphony is capable of handling medium scale simulations, however, scale issues become an obstacle once several thousand agents are present, mainly due to the single-threaded nature of Repast simulations (Lorig et al. 2015). The CASL DSL was implemented using the Xtext plug-in for Eclipse (Bettini 2013). In addition to creating the DSL rules, Xtext provides an Eclipse instance where CASL can be written utilizing standard IDE features such as code linting and auto-completion. Furthermore, the Eclipse instance allows for constraints to be enforced prior to any generation of code. These features further allow for CASL models to be constructed relatively easily. This also enables distribution of a version of Eclipse that is used to construct CASL models. One of the main benefits of CASL, delivered by most DSLs and code generators, is that the generated code can be optimized for the targeted simulation software (Bernstein et al. 2015). This provides a high confidence that the model created will execute as efficiently as possible with the targeted simulation software and thereby mitigate idiosyncrasies within the simulation software.

4 EXAMPLES

Several modeling languages are designed for use with specific domains in mind such as DEUS and Nanoverse. This implies that the languages contain domain-specific features which prevent them from being applied to other domains, thereby reducing their overall applicability. In this section, we demonstrate the flexibility of our approach by implementing three distinctly different models, namely the Game of Life (Gardner 1970), a social network (Birdsey et al. 2015), and a model of a hospital emergency department.

4.1 Game of Life

Conway's Game of Life (Gardner 1970) is a classic model that exemplifies agent-based modeling. The Game of Life model consists of a two-dimensional grid of cells, and each cell has two possible states, alive or dead. At each simulation step, each cell can become alive from dead or vice-versa depending on the states of its local neighbors. To model this as a CAS, we consider the life state changes of each Cell to be an implicit adaptation. However, the Game of Life model does not exhibit modularity or diversity. Furthermore, each Cell exists in a single implicit environment, which has no properties. While the Game of Life doesn't contain all of the identified CAS properties, using CASL to implement the Game of Life highlights the brevity of CASL and how the separation of interaction and adaptation processes allows for simple model design. The CASL code used for each Cell is shown in Figure 4. The total amount of code used for the CASL implementation of Game of Life is only 84 lines.

```
parameters: {
  var bool:Alive = false;
};
behaviors: {
  changeStateToDead[SELF][DELAYED]():{
    self.Alive = false;  };
  changeStateToAlive[SELF][DELAYED]():{
    self.Alive = true;  };
};
interactions: {
  countAliveNeighbors[QUERY][STEP(1)]():{
    var List:neighborsList =
        CASL.GRID.GetNeighbors[Cell](1);
    var int:aliveNeighbors =
        CASL.COUNT[neighborsList](Alive);
    ADAPTATION.changeLife(aliveNeighbors);
  };  };
adaptation: {
  changeLife[IMPLICIT][NONE](var int:n):{
    if (self.Alive) then
      if (n >= 2 && n <= 3) then
        BEHAVIOR.changeStateToDead();
      endif;
    else
      if (n == 3) then
        BEHAVIOR.changeStateToAlive();
      endif;
    endif;
  };  };
```

Figure 4: A Game of Life Cell implemented in CASL.

4.2 Social Network

The social network model considers several communities of advocate-led networks. Each agent follows the model presented in Birdsey et al. (2015), where each advocate follower visits the social network after a certain number of steps and may see a new post from the advocate. If the follower sees a post from the advocate, the follower may respond to it, depending on how interested they are in the posts content. The follower may instead just make an original post that could be on the communities topic, again, depending on how interested the follower is in the topic. However, even if the follower does not take notice of the new advocate post, they may just make an original post anyway. We extend this to a scenario with multiple disconnected communities. Each Community consists of a single Advocate with a number of Followers, and a Consensus for the Community, i.e., the communities' opinion on a particular topic. The Consensus forms the adaptive process for the Community environment and is triggered at the end of each simulation tick. Each Community's Consensus adapts according to how their respective users post and interact with the community advocate. Figure 5 presents a function from the Behavior component of a Follower.

```
makeAPost[SELF][INSTANT](): {
  if (FUNCTION.determineInterest()) then
    INTERACTION.makeNewPostToComm(self.comm);
    self.userState = USERSTATES.ON_TOPIC;
  else
    INTERACTION.makeNewPost();
    self.userState = USERSTATES.NOT_ON_TOPIC;
  endif;  };
```

Figure 5: A Follower Behavior function used to determine the type of post to make.

4.3 Emergency Department

Our Emergency Department model considers the flow of patients across a year following the Australian Department of Health emergency triage guidelines (Australian Government 2013). At each step, a number of patients enter the Emergency Department where they have their triage category determined by a Nurse and are then placed into a priority queue to wait for the next available Doctor. When a Doctor begins examining a Patient, the Doctor may take a sample from the Patient which has to be processed by the Pathology department. Once the Pathology department has finished processing, the results are returned to the Doctor, who then determines the course of treatment for the patient.

This model highlights two main benefits of CASL. Firstly, each agent exists within an environment, which provides a closer likeness to the real world system. Secondly, by treating the PathologyDepartment as being comprised of multiple agents, CASL allows this to be implemented easily. Figure 6 presents

an adaptive process for the EmergencyDepartment environment to determine the optimal roster for the following day.

```
EnhancePersonnelSchedule[IMPLICIT][NONE](): {
  if (self.PatientArrivalQueue.size() > maxPatientsWaitingForNurse) then
    FUNCTION.addNurseFromRoster();
  else
    FUNCTION.removeNurseFromRoster();
  endif;
  var int:currentPatientsOverdue = FUNCTION.CountOverduePatients();
  if (currentPatientsOverdue > maxOverduePatients) then
    FUNCTION.addDoctorToRoster();
  else
    FUNCTION.removeDoctorFromRoster();
  endif;
  if (self.pathology.ProcessingQueue.size() > maxPathologyQueueSize) then
    FUNCTION.addPathologyTechnicianToRoster();
  else
    FUNCTION.removePathologyTechnicianFromRoster();
  endif;  };
```

Figure 6: An EmergencyDepartment Adaptation function.

5 EXPERIMENTAL ANALYSIS

To highlight the capabilities of CASL with respect to scale and adaptation, we performed several experiments across our three example models. For our Game of Life, and social network models, we performed experiments to show how CASL generates moderately scalable models. For our Emergency Department model, we performed experiments to exhibit how CASL can be used to create simple and practical adaptive methods at an environmental level.

5.1 Scale

For two of our models, namely, the Game of Life, and social network models, we performed several experiments to highlight the scaling capabilities of the code that CASL generates. Figures 7 and 8 display the runtimes for our Game of Life and social network experiments respectively.

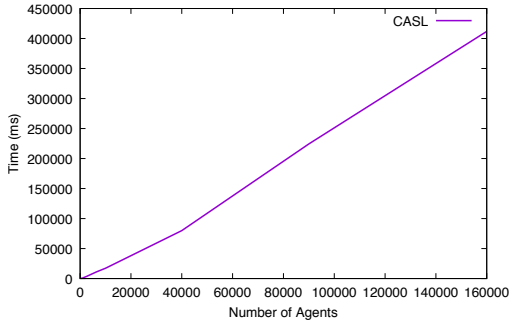

Figure 7: Runtimes for the Game of Life model.

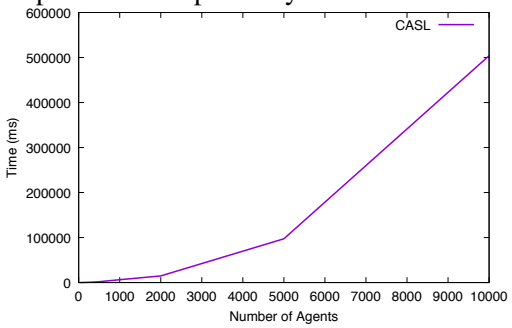

Figure 8: Runtimes for the social network model.

For our Game of Life model, we performed experiments with Cell populations ranging from 225 Cells to 160,000 Cells. Each experiment ran for 1,000 steps with 10 replications. For our smallest configuration at 225 Cells, we attained a runtime of 173 milliseconds to execute, while with a population of 160,000 Cells, the runtime was 411 seconds. As the Game of Life is a relatively trivial model, the runtime increase is fairly linear. However, beyond 160,000 cells we are subject to various Repast and Java.

For our social network model, we performed experiments with varying numbers of Communities and Users per Community. Each experiment is executed for 2,500 steps, and configurations range from 2 Communities with 20 Users each, to 10 Communities with 1000 Users each, with 10 replications of each

experiment. Our smallest configuration, which contained a total of 40 agents, took 67 milliseconds to execute 2500 steps, while our largest, which contained 10,000 agents, took approximately 8 and a half minutes to execute. However, beyond 10,000 agents we were again subject to Repast and Java limitations.

5.2 Adaptation

For our Emergency Department model, we performed experiments using Environmental adaptation to determine the optimal number of Doctors, Nurses, and Pathology Technicians that should be working across the next 24 hours. The adaptation process determines the new roster based on the number of Patients not seen after by a Doctor after a 4 hour time limit, how many Nurses were not needed, and how many pathology samples still need analysis. Table 1 presents our experimental results.

Table 1: Emergency Department Roster Adaptation Results.

Initial Roster (Doctors/Nurses/Technicians)	**Average Patients per hour**	**Final Roster (Doctors/Nurses/Technicians)**
7/15/4	1.00	3/3/3
7/15/4	2.00	4/3/2
7/15/4	3.00	8/5/3
8/5/3	3.00	9/6/3

These results highlight the ability to easily create an adaptable component, in this case the staff roster, for a model which can also be easily expanded. If the model needs other factors, such as Doctor seniority or medical technologies, CASL allows for these to be added rather easily. Furthermore, the entire Emergency Department model can be placed inside a larger model, such as one for an entire hospital, with little effort.

5.3 Discussion

For our three models, the total amount of CASL code written amounts to under 1000 lines, with the largest requiring 466 lines and the smallest needing only 84. However, as we use Repast Simphony as our target simulation software, the models are unable to scale well, with a Game of Life model consisting of 225 agents taking only 173 milliseconds, while one with 160,000 agents takes approximately 7 minutes. While many small scale CAS models can be executed using Repast, to fully utilize CASL and gain a deeper understanding of these systems, a simulation tool capable of handling large scale models is necessary. Our social network model suffers from similar scale issues, with a population of 10,000 agents taking approximately 8 and half minutes to execute. However, our social network model consists of multiple distinct communities, each containing a number of users. CASL easily allows for alterations to our model, such as if the communities require different types of users, cross-communication, and adversarial agents, thus lending to a more realistic social network model. Our emergency department model showcases an environmental adaptation component. While the adaptive process we implemented is fairly simple and doesn't consider many factors, in models that contain more detailed agents it would be trivial to create more accurate adaptation methods that are capable of generating results that translate into real-world actions.

6 CONCLUSION

Complex adaptive system modeling and simulation provides significant challenges as they contain a large number of interacting complex entities. We propose a new modeling language, CASL, and framework, targeted specifically for the modeling and simulation of complex adaptive systems. CASL provides distinct components for each entity which considers how they interact, behave, adapt, and how subsystems affect them. By providing these distinctions, entities can be modeled in a way that ensures they have essential properties. Moreover, these separate components, and the relationships between them, can provide a deeper understanding of how entities operate individually and in their respective systems. The components also

allow for entities to be constructed relatively easily as focus can be placed on each component individually, and that the constraint components ensure that the entities contain the correct properties for the particular CAS model. In addition to these three models, CASL has also been used to model birds flocking and an ant colony, further showcasing the flexibility of CASL and our framework.

Our future work is three-fold. Firstly, we seek to enhance CASL and the framework to be able to handle simulations with large amounts of complex, modular agents. This would enable modeling and simulation of more real world complex adaptive systems. Secondly, we aim to expand our collection of observation modules which will allow for greater insight into complex adaptive systems. Thirdly, we intend to implement a graphical modeler to further increase the efficiency of creating CAS models while also reducing the bar to be able to construct models for simulation.

REFERENCES

Australian Government 2013. "Department of Health | Emergency Triage Education Kit". http://www.health.gov.au/internet/main/publishing.nsf/Content/casemix-ED-Triage+Review+Fact+Sheet+Documents.

Aydt, H., M. Lees, and A. Knoll. 2012. "Symbiotic Simulation For Future Electro-mobility Transportation Systems". In *Proceedings of the Winter Simulation Conference*, edited by C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, 149 – 161. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Balan, G., C. Cioffi-Revilla, S. Luke, L. Panait, and S. Paus. 2003. "MASON: A Java Multi-Agent Simulation Library". In *Proceedings of Agent Conference on Challenges in Social Simulation*.

Banks, J., J. S. Carson II, B. L. Nelson, and D. M. Nicol. 2005. *Discrete-Event System Simulation*.

Bellifemine, F., A. Poggi, and G. Rimassa. 1999. "JADE–A FIPA-Compliant Agent Framework". *Proceedings of PAAM*:97–108.

Benham-Hutchins, M., and T. R. Clancy. 2010. "Social Networks As Embedded Complex Adaptive Systems". *The Journal of Nursing Administration* 40 (9): 352–6.

Bernstein, G. L., C. Shah, C. Lemire, Z. DeVito, M. Fisher, P. Levis, and P. Hanrahan. 2015. "Ebb: A DSL for Physical Simulation on CPUs and GPUs". *arXiv preprint arXiv:1506.07577*.

Bettini, L. 2013. *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing Ltd.

Birdsey, L., C. Szabo, and Y. M. Teo. 2015. "Twitter Knows: Understanding The Emergence Of Topics In Social Networks". In *Proceedings of the Winter Simulation Conference*, edited by L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, 4009–4020. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Borenstein, D. B. 2015. "Nanoverse: A Constraints-Based Declarative Framework For Rapid Agent-Based Modeling". In *Proceedings of the Winter Simulation Conference*, edited by L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, 206–217. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Boulaire, F., M. Utting, and R. Drogemuller. 2015. "Dynamic Agent Composition For Large-scale Agent-based Models". *Complex Adaptive Systems Modeling* 3 (1): 1–23.

Chan, Wai Kin Victor, Y.-J. S., and C. M. Macal. 2010. "Agent-based Simulation Tutorial-Simulation Of Emergent Behavior And Differences Between Agent-based Simulation And Discrete-Event Simulation". In *Proceedings of the Winter Simulation Conference*, edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, 135–150. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Cioffi-Revilla, C., K. De Jong, and J. K. Bassett. 2012. "Evolutionary Computation And Agent-based Modeling: Biologically-inspired Approaches For Understanding Complex Social Systems". *Computational and Mathematical Organization Theory* 18 (3): 356–373.

Dastani, Mehdi, M. v. B. R., and J.-J. C. Meyer. 2005. "Programming Multi-Agent Systems in 3APL". *Multi-Agent Programming*:39–67.

Gardner, M. 1970. *The Fantastic Combinations of John Conway's New Solitaire Games*. Mathematical Games.

Hasgall, A. 2013. "Digital Social Networks As Complex Adaptive Systems". *Vine* 43 (1): 78–95.

Holland, J. H. 1992. *Adaptation In Natural And Artificial Systems: An Introductory Analysis With Applications To Biology, Control, And Artificial Intelligence*. A Bradford Book.

Holland, J. H. 2006. "Studying Complex Adaptive Systems". *Journal of Systems Science and Complexity*.

Lorig, F., N. Dammenhayn, D.-J. Müller, and I. J. Timm. 2015. "Measuring and Comparing Scalability of Agent-Based Simulation Frameworks". *Multiagent System Technologies* 9433:42–60.

Mittal, S. 2013. "Emergence In Stigmergic And Complex Adaptive Systems: A Formal Discrete Event Systems Perspective". *Cognitive Systems Research* 21:22–39.

Niazi, M., and A. Hussain. 2011. "A Novel Agent-Based Simulation Framework for Sensing in Complex Adaptive Environments". *Sensors Journal* (2): 404–412.

Niazi, M. A. 2013. "Complex Adaptive Systems Modeling: A Multidisciplinary Roadmap". *Complex Adaptive Systems Modeling* 1 (1): 1.

North, M. J., N. T. Collier, J. Ozik, E. R. Tatara, C. M. Macal, M. Bragen, and P. Sydelko. 2013. "Complex Adaptive Systems Modeling With Repast Simphony". *Complex Adaptive Systems Modeling* 1 (1): 3.

Ozik, J., N. T. Collier, J. T. Murphy, and M. J. North. 2013. "The ReLogo Agent-Based Modeling Language". In *Proceedings of the Winter Simulation Conference*, edited by R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, 1560–1568. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Özmen, Ö., J. Smith, and L. Yilmaz. 2013. "An Agent-based Simulation Study Of A Complex Adaptive Collaboration Network". In *Proceedings of the Winter Simulation Conference*, edited by R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, 412–423. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Picone, M., M. Amoretti, and F. Zanichelli. 2012. "Simulating Smart Cities with DEUS". In *Proceedings of the Fifth International Conference on Simulation Tools and Techniques*, 172–177.

Raunak, M., and L. Osterweil. 2013. "Resource Management for Complex, Dynamic Environments". *IEEE Transactions on Software Engineering* (3): 384–402.

Smith, R. M., and M. Bedau. 2000. "Is Echo A Complex Adaptive System?". *Evolutionary Computation*.

Szabo, C., Y. M. Teo, and G. K. Chengleput. 2014. "Understanding Complex Systems: Using Interaction As A Measure Of Emergence". In *Proceedings of the Winter Simulation Conference*, edited by A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, 207–218. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Tisue, S., and U. Wilensky. 2004. "Netlogo: A Simple Environment For Modeling Complexity". *International Conference on Complex Systems*:1–10.

AUTHOR BIOGRAPHIES

LACHLAN BIRDSEY. Lachlan's email address is lachlan.birdsey@adelaide.edu.au.

CLAUDIA SZABO. Claudia's email address is claudia.szabo@adelaide.edu.au.

KATRINA FALKNER. Katrina's email address is katrina.falkner@adelaide.edu.au.