University of Arkansas – CSCE Department
Capstone I – Preliminary Proposal – Fall 2017

WebDNA Project Proposal

Salvador Sanchez, Zack Fravel, David Darling, Jace McPherson, Jonathon Raney

Abstract - Zack

oxDNA is an extensible DNA simulation and analysis software used by DNA researchers from various fields of study. While existing as a powerful tool for running DNA simulations, oxDNA remains difficult to utilize for users without a computer science background. As a team, we want to bring these tools to a wider pool of users by simplifying the simulation process and building out a web-based user interface.

Along with the standard simulation data, we also hope to build in analysis and visualization tools that share immediately useful information with the user. Our goal by the end of the project is to have made a significant contribution to the field of molecular self assembly by enabling researchers from different backgrounds to have access to these powerful DNA simulation tools.

1.0 **Problem** - Jace

oxDNA is currently clunky in its workflow. There are many isolated steps involved in the data generation and simulation process. With so many distinct, advanced processes, including compiling oxDNA, generating initial system state data using python scripts, converting the output to JSON files for visualization purposes, and running scripts to perform analysis, the simulation software is extremely inaccessible.

Even for advanced users, the entire simulation pipeline is bloated and inefficient. In many cases, the stock analysis scripts (written in Python) do not provide enough functionality to cover all the possible analysis cases. For advanced data generation and analysis, researchers are required to write their own python scripts, which is an overly advanced process. The lack of process unification and simplicity also takes a toll on this useful software's accessibility.

2.0 **Objective** - Jace

The objective of this project is to wrap oxDNA functionality in a simplified user interface that is accessible via the Internet. The problems mentioned in section 1.0 should ideally be solved by the final product. In summary the software must solve the overall issue: accessibility. A unified portal to the oxDNA functionality, as well as easier and fully-functional analysis tools (provided via built-in and user-defined analysis functions) will ensure a smooth user experience.

3.0 Background

3.1 Key Concepts - John, David

Undoubtedly, the most important technology this project will be working with is the oxDNA software, a molecular dynamics and modeling program used to compute complex simulations of nucleic acids. The software includes features for several different types of common simulations. These include: molecular dynamics, Brownian dynamics, and Metropolis Monte Carlo. Additionally, Lennard-Jones interactions, Kob-Andersen mixtures along with multiple patchy particle models can be generated [1]. This rich feature set makes oxDNA very useful for researchers looking to utilize computing power for predicting how molecular systems will interact in a wide variety of scenarios.

In order to implement the required server functionality, Django Python in conjunction with a PostgreSQL database will be used. Django is an open-source web framework for use with Python. It follows the Model-View-Template architecture with the goal of ease-of-use in mind. Django's MVC architecture is designed with an object-relational mapper which mediates between data models and a database [2]. This mapper makes up the model portion. The view portion consists of a web templating system which processes HTTP requests. A regular-expression based URL dispatcher serves as the controller. Django is very flexible as it allows third-party code plugins to be run in a regular project. This enables users to extend the original toolkit to tackle almost any conceivable problem without having to resort to solutions outside of the framework. This flexibility will allow easy conjunction with the prewritten 3D visualizer using 3js which will eventually be used and updated to analyze simulation outputs.

oxDNA produces a trajectory file where all the relevant information to viewing DNA reaction simulation images lies. This info is translated into a pdb or xyz file using a converter provided in the "UTILS" directory. This file can be analyzed in VMD from xyz output format or by UCSF's Chimera from pdb format to produce useful visualizations. Chimera is developed by the Resource for Biocomputing, Visualization, and Informatics at the University of California, San Francisco and provides extensive interactive DNA visualization [3]. Complete documentation and download is free of charge for noncommercial use. Some of its key features include but are not limited to automatic identification of atom types, high-resolution images, molecular dynamics trajectory playback (many formats), and distance and angle plots.

3.2 Related Work - David

The foundational research for DNA self-assembly systems was originally carried out by Erik Winfree. Winfree developed a model for artificial, self-assembling systems called the Tile Assembly Model which he introduced in his 1998 thesis [4]. This model could be split into two versions, namely the abstract Tile Assembly Model and the kinetic Tile Assembly Model (aTAM and kTAM respectively). Between the two versions of the model, the key differences lie in the fact that the aTAM version generally ignores realistic errors and provides a more high-level approach, while the kTAM version accounts for errors and provides means to analyze errors that can happen in reality. Because of this accounting for errors, the kTAM model has been

used in actual lab experiments to predict the ways assemblies will form. Winfree's research into these types of models showed that the systems could be classified into the field of algorithmic self-assembly.

oxDNA itself was originally based around the research of T.E. Ouldridge, J.P.K. Doye, and A.A. Louis, who introduced the coarse-grained DNA model [5]. oxDNA has since been expanded by multiple research groups to be a framework for simulating and analyzing DNA and RNA. However, oxDNA is not the only molecular dynamics simulation software currently available. Another nucleic acid simulation software is Nanoscale Molecular Dynamics (NAMD), created by researchers from the University of Illinois at Urbana-Champaign; This software features scalable simulations that can utilize hundreds of processing cores in order to model massive molecular systems [6]. Similar to oxDNA, in order to correctly utilize NAMD requires extensive technical knowledge of the subject as well as a solid grasp of programming concepts. These requirements are generally too demanding of researchers not from a computer science background and causes unnecessary delays in getting simulations up and running. The development of a user-friendly graphical interface to oxDNA should mitigate many of these issues and attract more researchers to the software.

4.0 Design

4.1 Requirements and Design Goals - Jace

The culmination of the WebDNA software should accomplish the following:

- Provide the user with an interface for generating input data to a simulation environment.
- Allow users to control the execution parameters of a simulation. This includes, but is not limited to: simulation type, initial seed, time steps, and temperature. These parameters are based on the generic/simulation options
- Display simulation progress and provide a visualizer to view the current state of the simulation.
- Provide a visualizer to render the system state at different points in time.
- Provide an analysis pipeline editor. Such an editor would allow users to perform analyses on the output of the simulation, mapping and/or reducing system states to other forms of analysis data. See below for more details.

One of the highlights of the aforementioned requirements is the *analysis pipeline editor*. This serves as a solution to the cumbersome analysis process that DNA researchers undergo every time they want to extract or calculate new information from simulation results. As mentioned in section 1.0, current analysis solutions require manually writing python scripts to analyze execution data. The *analysis pipeline editor* would give users the ability to visually pipe the raw simulation output through a series of scripts such that they can quickly process simulations meaningfully. See Figure 1 for an example of such a pipeline.

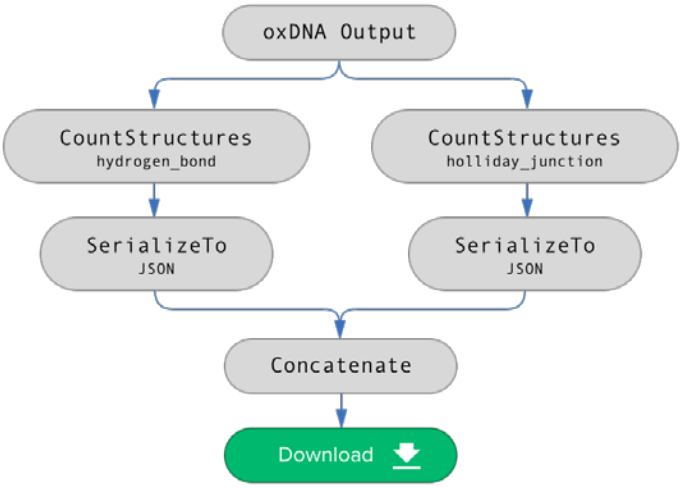

Figure 1. Mockup of the *analysis pipeline editor*. The WebDNA pipeline editor will provide default analysis nodes, as well as custom user-defined nodes.

The variety of analyses that may be performed is great, so it is conceivable that no visual editor can cover all the cases researchers may need to execute. In this case, it is useful to provide a "custom script" operator, where a researcher who is savvy with Python can manually perform a step of the pipeline without having to rely on built-in tools. The semantics of this functionality imply script sandboxing, since their custom scripts will be executed on our servers (which is a blatant system vulnerability if not properly safeguarded).

The implementation of the remaining key requirements is highly dependent on the nature of web application development. For example, all simulation configurations must be stored on a central server, then served to the user so they can view and modify the configurations. This data flow is also necessary for simulation visualizations, which are stored on the server and served to the user, rather than performing intense computations on the user's end. Section 4.2 covers the implementation details for these broad features.

4.2 **High Level Architecture** - Salvador and Jace

As mentioned in section 4.1, the server and client will be performing distinct and separate tasks in order to fulfill the end-user requirements and functionality. Essentially, the client is a "pretty terminal" view on the server, granting a limited view of oxDNA execution configurations and output data. The server is responsible for handling requests for custom executions, performing standard parameterized oxDNA simulations, as well as executing utility scripts, both built-in and custom-uploaded by users. To further explain, we will cover the server architecture in section 4.2.1 and the client architecture in section 4.2.2.

4.2.1 Server Architecture

The server needs to provide request endpoints to perform the following actions (the list is not comprehensive, but highlights the main requests clients will make):

1. Create and save simulation configurations and parameters.
2. Fetch previously saved simulation configurations and parameters.
3. Begin the execution of an oxDNA simulation based on a previously saved configuration.
4. Fetch visualization data (i.e. system state data) for a currently executing or previously executed oxDNA simulation.
5. Request the execution of previously made sandboxed python scripts on generated input files.
6. Request the execution of premade/custom scripts on output data. The server should provide a suite of generic analysis scripts that users can chain together to obtain meaningful, customized results.

The server will be running with Python using Django. The Django server will be capable of redirecting HTTP requests to python functions that can either render web pages or perform one of the actions listed above.

Data will be stored in a PostgreSQL database. The following data types will be implemented:

- **User**: Contains standard user data, such as email, name, password hash, salt for the hash, etc. Users also possess Projects.
- **Project**: Contains simulation execution configuration data and references to current/past Jobs. Also contains output analysis pipeline data in the form of a linked list of python Scripts.
- **Job**: Contains information about a currently executing or previously executed simulation using the “oxDNA” executable. Jobs need to store much of the same information as a Project.
- **Script**: Contains information about a script that should be scheduled for execution by the server. Elements of this table need to know what Python script to execute, all parameters to that script, what files to use as input, where to save output, and if there are any subsequent Scripts that need executing.

As project development continues, more tables will need to be implemented, and the structures of these tables may change from this original proposal, but these four tables represent the most important aspects of the server’s data storage.

4.2.2 Client Architecture

Our front end solution to the oxDNA simulation software will be a web page that allows the user to create Projects. Projects, from a user standpoint, allow users to isolate distinct configurations and simulation results into manageable groups. Project creation/execution involves three main components: the *input configurator*, the *simulation visualizer*, and the *output pipeline*.

With respect to the user experience and application layout, the project components will be split into two distinct pages per project; one page will contain only the *input configurator*, while the second page will contain both the *simulation visualizer* and the *output pipeline*. This is because running a simulation is a big, time-consuming job, and should therefore be treated functionally as a “job submission” rather than a quickly adjustable visualization. This is not to say that a

simulation could not be cancelled, adjusted, then restarted, but rather that a two-page layout emphasizes the magnitude of the work being done.

The *input configurator* will be segmented into two categories: *required* and *optional* settings. Required settings include the following:

1. Sequence Input: A module that will generate a file of character-based sequences of DNA (or RNA) strands as the primary input or accepts a file/manual text input.
2. Generator Script: A module specifying the “generator”, i.e. a Python script that will generate topology and configuration data from the sequence input (Required Setting 1).
3. Generic Options: A module that asks for generic options for the simulation. These are based on the “Generic Options” listed in the oxDNA documentation [5].
4. Simulation Options: A module that asks for custom simulation-specific options. These are based on the “Simulation Options” listed in the oxDNA documentation [5]. Some options have prerequisites for other option states before the original option can be set.

Optional settings refer to any non-critical input-phase parameters that the user may want to make. These “parameters” are limited to modifications of the generated topology and configuration data created based on settings in the “Generator Script” setting (Required Setting 2):

1. Custom Scripts Module: This allows users to run pre-defined scripts (bundled with the oxDNA software on the backend) or run their own sandboxed Python scripts on the generated input data. Essentially, the user is allowed to transform the boilerplate generated topology and configuration files with these pre-defined/custom scripts.

The settings and scripts modified in the *input configurator* are saved to their respective Project associated with a user account. Figure 2 on the next page shows a mockup interface for the *input configurator*.

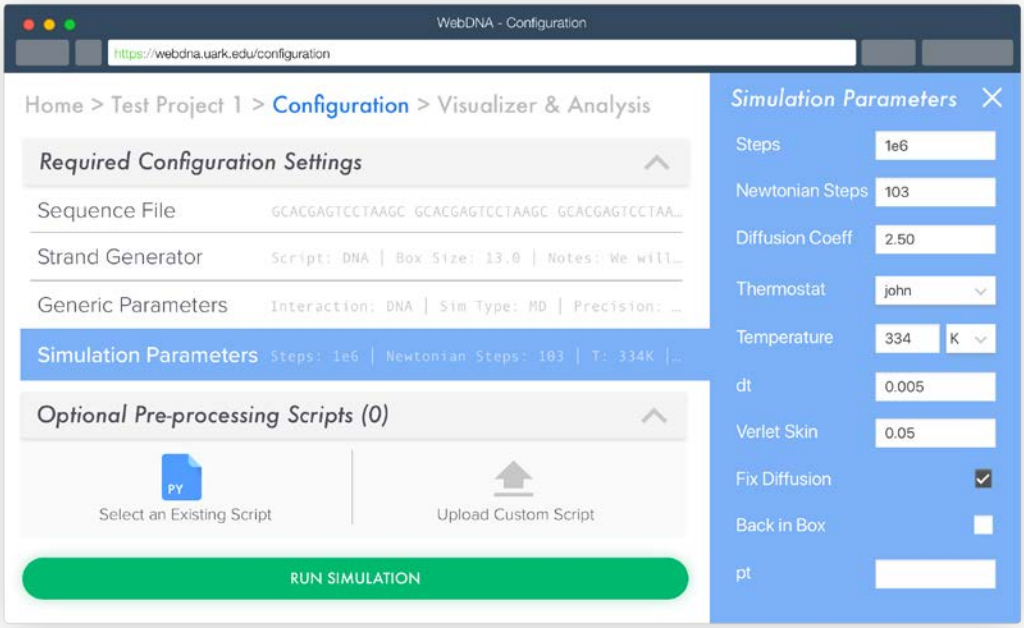

Figure 2. A mockup of the *input configurator*, the first of two pages for all projects.

The next major component of a Project relates to the *simulation visualizer*. There are a couple semantics relating to simulations for projects in WebDNA, listed below:

- Simulations are slow, so the visualizer should display simulations as they are being calculated in near-real-time.
- All simulations — finished or not — should allow backward scrubbing to previous points in the simulation, if they were configured to save the system state throughout the simulation.

The visualizer will be based on the current implementation of a basic browser visualizer for oxDNA output files. This visualizer was written using a simple Python HTTP server, so it will need to be migrated to function with our Django HTML renderer. In addition, the current visualizer uses Three.JS to render the DNA system state using WebGL.

The final component, the *output pipeline*, will allow the user to pipe system states through user-defined analysis programs, which are created using a drag-and-drop style pipeline interface. Figure 1 in section 4.1 gives a good visual for what this pipeline will look like. The requirements for this pipeline are not entirely clear, as there are so many potential use cases. Therefore, the pipeline will give access to the scripts in the oxDNA UTILS directory, as well as allow users to upload custom scripts (which will be sandboxed as mentioned in section 4.1).

4.3 Risks - Salvador

Risk	Risk Reduction
Learning curve for the Custom Scripts Module	Creating an easily accessible help page about a nearly intuitive scripting language.
Unmanageable codebases	Develop a framework and organization for source code beforehand, as well as an agreement on a rigorous style guide.
File upload security risk	Prescreening of all files.

4.4 Tasks - Zack

1. Get a better understanding the inputs and outputs of oxDNA along with Python/Django.
2. Develop an understanding of the Chimera software and JS for possible implementation.
3. Create web-based front end and database setup.
4. Add functionality to inject the generated input files into oxDNA on the front end.
5. Add output file analysis functionality, such as hydrogen bond count.
6. If time permits, find a suitable visualization method for further output analysis.
7. Document the final project.

4.5 Schedule – Zack

Tasks	Dates
1. Familiarize team with oxDNA and Python/Python-based web development (Django).	11/5 - 1/16
2. Learn more about visualization tools (JS/Chimera).	11/5 - 1/16
3. Create web-based front end and database setup.	1/16 - 2/13
4. Add oxDNA simulation functionality to front-end.	2/13 - 3/6
5. Add the ability to perform output file analysis on the webDNA simulation interface.	3/6 - 3/27
6. Integrate visualization tools for oxDNA simulations using Javascript visualizer and/or Chimera.	3/27 - 4/17

7. Put together final written documentation meant for end-user.	4/17 - 5/1

4.6 **Deliverables** – Salvador and Zack

- Design Document
    - Outline front end functionality, database schema, and scope of analysis
- Website
    - Python/Django
- Data Analysis
    - Output documents
    - Work in existing Python scripts into front end analysis
    - Sandboxing of custom data analysis python scripts
- Visualization tools
    - Javascript
    - Chimera
- Final Documentation
    - Provide a detailed document meant to be read by a non-computer scientist that provides instruction on the use of the WebDNA front end for oxDNA simulations.

5.0 Key Personnel

Dr. Matthew Patitz – Dr. Patitz performs research in self-assembling systems, and has published a number of papers related to theory, software, and experimental aspects of self-assembling systems [7].

Salvador Sanchez – Sanchez is a senior Computer Science and Mathematics major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Programming Paradigms, Software Engineering, and Operating Systems. Front end development with Zabbix monitoring legacy PHP code, making server metric reporting pages for a Cerner systems engineering team.

Zack Fravel – Fravel is a senior Computer Engineering major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed a coursework in Embedded Systems, VLSI Synthesis, Low Power Digital System Design, Programming Paradigms, as well as independent study of Abnormal & Evolutionary Psychology.

David Darling - Darling is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Software Engineering and Programming Paradigms. He completed internships at Entergy Arkansas creating automated tools to streamline the design process for new electrical transmission structures.

Jace McPherson - McPherson is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has worked software development/engineering internships at Metova and Google, as well as researched and developed software to control multiple 3D printer robots to perform a single print cooperatively.

Jonathon Raney - Raney is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Programming Paradigms and Operating Systems. Currently undergoing further study in Genetics and Bioinformatics courses.

5.1 Facilities and Equipment - Zack

There isn't much hardware that we need for our purposes currently. Down the road it would be useful to own a few machines with powerful GPUs to perform quicker simulations. On the software side however, one very useful potential purchase could be licenses for a SVN GUI client such as SmartSVN or Versions. Versions costs $59 per license [8]. SmartSVN costs $99 per license but offers a small discount when purchasing multiple [9].

6.0 References

[1] oxDNA Features, https://dna.physics.ox.ac.uk/index.php/Features
[2] Django Documentation, https://docs.djangoproject.com/en/1.11/
[3] UCSF Chimera, https://www.cgl.ucsf.edu/chimera/
[4] Self Assembly, http://self-assembly.net/wiki/index.php?title=Main_Page
[5] oxDNA Documentation, https://dna.physics.ox.ac.uk/index.php/Documentation
[6] NAMD, http://www.ks.uiuc.edu/Research/namd/
[7] UofA Directory, https://csce.uark.edu/directory/index/uid/patitz/name/Matthew-Patitz/
[8] Versions, https://versionsapp.com/
[9] SmartSVN, http://www.smartsvn.com/