THE INSTITUTE FOR SYSTEMS RESEARCH

ISR TECHNICAL REPORT 2007-14

LTSA Modeling of the Panama Canal

John Johnson

The Institute for Systems Research

A. JAMES CLARK SCHOOL OF ENGINEERING

ISR develops, applies and teaches advanced methodologies of design and analysis to solve complex, hierarchical, heterogeneous and dynamic problems of engineering technology and systems for industry and government.

ISR is a permanent institute of the University of Maryland, within the A. James Clark School of Engineering. It is a graduated National Science Foundation Engineering Research Center.

www.isr.umd.edu

REU PROGRAM
FINAL REPORT
LTSA MODELING OF THE PANAMA CANAL
By: John Johnson

Problem Description:

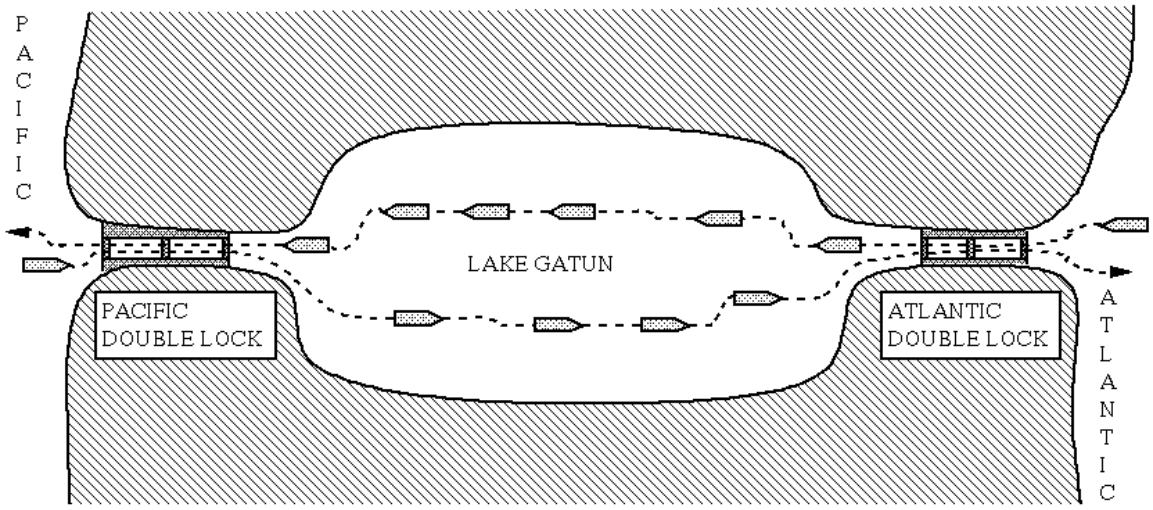
To research, discover, and apply methods of system analysis and control for modeling the Panama Canal using the Labeled Transition System Analyzer Tool.

Groundwork:

The groundwork consisted of gathering information about all parts of the overall project, in an effort to make sure that no time was wasted over the summer's short three months.

The first step was getting to know the Panama Canal. The Panama Canal is a very complicated mechanical and biological system that provides not only the passage across the Americas for shipping, but a relatively large percentage of the Panamanian GDP. The fact is that the Panama Canal is as much dependent on good mechanical controlling as it is on the large ancient forests that surround it, in the Chigres National Park. After gaining ownership of the canal, the Panamanians have been making large strides in monitoring, maintaining, updating, and controlling the complex system.

Diving to the source of the activity at the Panama Canal revealed that the Panamanians are in a race against the shipping industry. The shipping industry is quickly heading towards using only its most mammoth ships, and the world's shipping infrastructure has to accommodate. For this reason, the Panamanians have begun building a new larger channel, which they hope to complete by the hundredth anniversary of the Panama Canal, in 2014.



With the world's shipping industry heading towards remodeling and renovation, it set the stage for our project: to use the LTSA tool to demonstrate its ability to create a fail-safe controlling and monitoring system structure. Since this groundwork was done

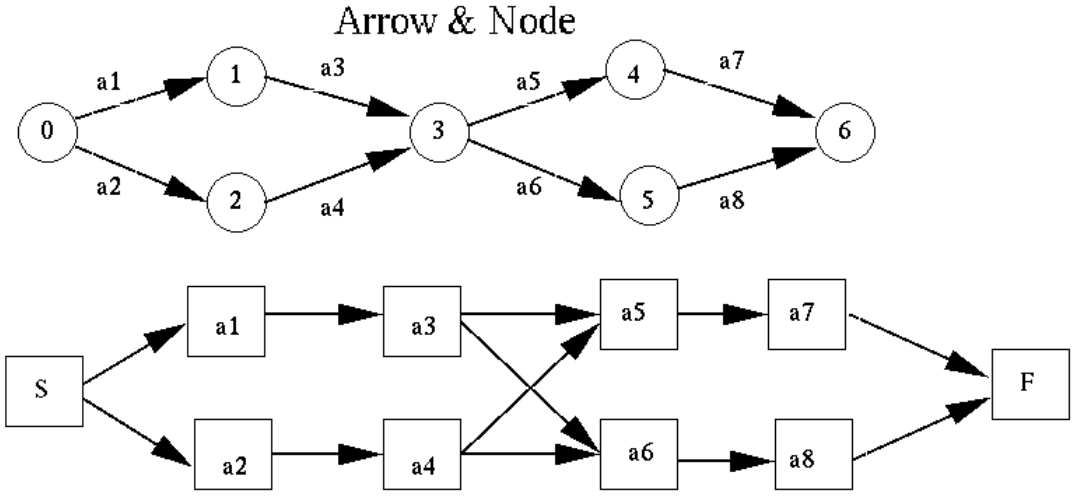
before the program began, as part of an assignment in a technical writing course, it allowed the project to move ahead immediately with confidence at the beginning of the summer. However, to use LTSA to its fullest, over one hundred hours of training, and constant discovery throughout the project would be required.

Learning LTSA:

Learning LTSA (Labeled Transition System Analyzer) was a combination of studying literature, examining example problems, and trial and error. Because it is a program for modeling systems, and systems themselves are very complex, and multi-layered, the learning curve had many stages. Even in the last week of the program, more progress was made in understanding the true nature of the theory and techniques applied in the LTSA program.

During the first two weeks of the program over 50 pages of notes were taken resulting in the understanding that LTSA relied on two key elements: precedence and synchronization. However, in this method of system analysis and control, time is either quantized or eliminated all together.

LTSA creates what are known in the project management profession as arrow-and-node diagrams. Arrow-and-node diagrams show how a project moves from state to state throughout a project and it shows what actions are required to be completed before others, like the counterpart diagrams 'flow charts' or box-and-arrow diagrams. However, there is one dramatic difference between these project management methods of analyzing projects, and the LTSA approach to system analysis; systems are more complex, and continuous.



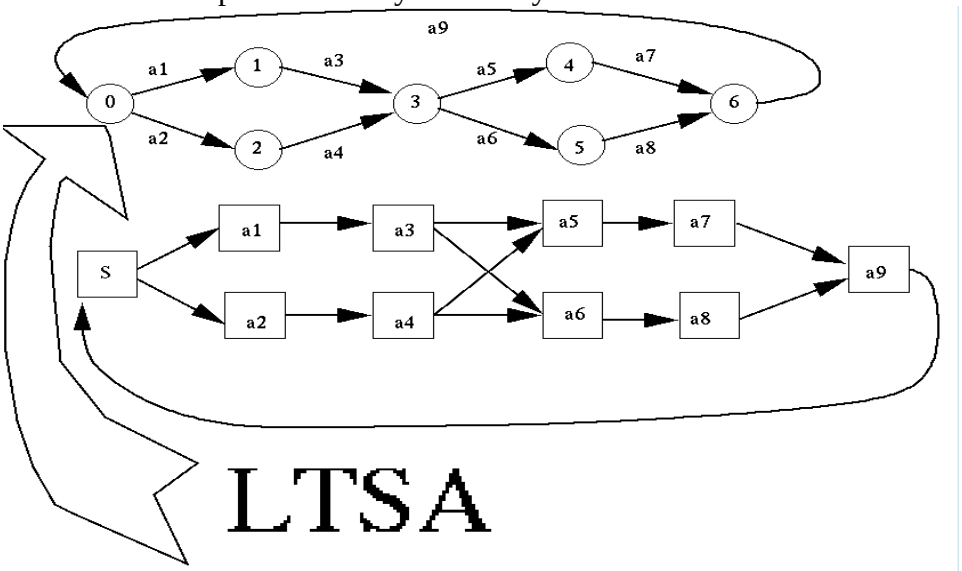
Box & Arrow / Flow Diagram

Systems, like programs, are understood as being cyclical in nature. A common system is a car engine, which repeats its tasks over and over again, sometimes at higher speeds (higher rpms), and sometimes seemingly frozen (when the car is turned off), but the engine is meant to continue to turn and produce power to move the car forward. If a car engine were a project, it would complete one rotation and be finished, because the goal of rotating the engine had been completed, and projects by definition have a start

and a finish. However, thankfully, the car is a system that completes and repeats its built-in projects to continue to produce something of value.

The question then is why the LTSA program uses the arrow-and-node method of analysis, when the box-and-arrow method is so widely used and understood. The answer lays in the difference of the two techniques, and the importance of states to system analysis.

The difference between arrow-and-node diagrams and box-and-arrow diagrams is that the arrow-and-node diagrams show the states, and label the arrows with the actions that are required to move to the next state in a project; where as the box-and-arrow diagrams show the precedence of the actions with the arrows, and the actions within a project are contained within the boxes. States are not shown in box-and-arrow diagrams, and states are important with system analysis.



States are like snapshots of a system, or project, that can be categorized in what has occurred, and what has yet to occur. One can think about the simple system of a light that is either on or off. When the “on” action has occurred, and the “off” action has not occurred yet, then the light is in the “ON” state (LTSA uses capitals to differentiate between STATES and actions). But more importantly, states are the most succinct and efficient way to analyze a system.

Since systems are known to have concurrent behavior, there are many different paths to reaching a state, just as there are many different paths to getting to the top of a mountain. If one had two light switches and wanted them both to be turn on, there would be two paths consisting of two actions each, for turning both light switches, but only one state of both being turned on. If one had three lights, and desired them all to be on, there would be six different paths of three actions, not including paths where lights are turned off, to reach the “ALLON” state. Since computation has its limits, and wasting time is not desired either, a state-focused, arrow-and-node analyzing method makes the most sense in this approach to systems modeling.

Understanding just how LTSA works took the entire summer, but most of the discovery was made through the process of actually creating the canal model. Since progress had to be made, after two weeks of study and investigation, the process of creating a solid canal model began.

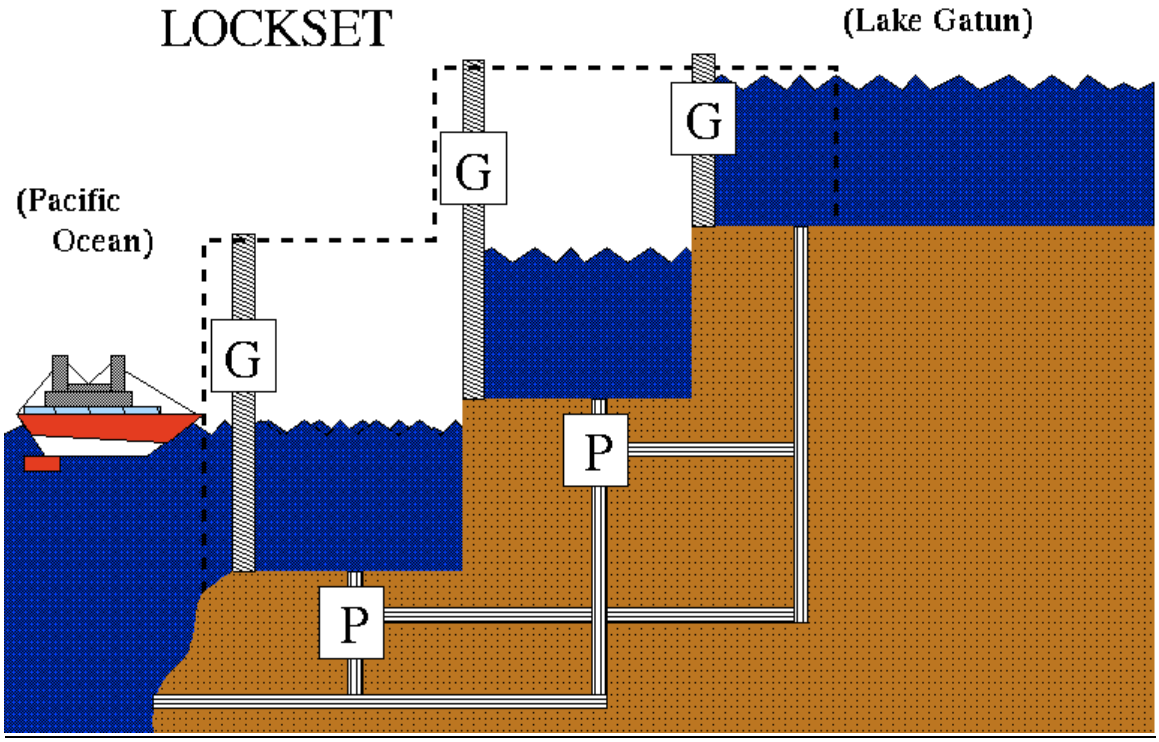
Creating the Canal:

Starting off, an example was needed to give guidance on how to approach the creation of a lockset. Locksets amount to a temporary one-way path that raises or lowers ships from one altitude to another. Within the material that was used to prepare for creating the canal model, there is an example program of a one-way bridge, which seemed a promising launch pad.

The SingleLaneBridgeFair example, from Jeff Magee and Jeff Kramer's book, Concurrency: State Models & Java Programming, came with the LTSA program, and was used as a guide of how the structure of control could be maintained for a small single-lock system.

The example program used simple explanations of how cars use a one-lane bridge when there are two opposite traveling convoys of cars on a two-lane road, both needing to use it. The conversion was simple, the bridge is renamed lock, and the action "enter(bridge)" becomes "enterlock" and so forth. There were, however, some differences when it came to the monitor (the overall controller of the use of the bridge/lock).

The single lock monitor ended up needing fewer variables, or monitoring parts, than the single lane bridge monitor. With the bridge, fairness is based on allowing multiple cars to use the bridge in the same direction, when it's efficient to do so. In the lock model the bridge was either in use or not, which allowed for a smaller monitor with one less variable to interpret. Also our model required fewer restrictions on the processes, since the ships could pass each other after using the lock (where as cars must remain in a single file—that's one less line of code).



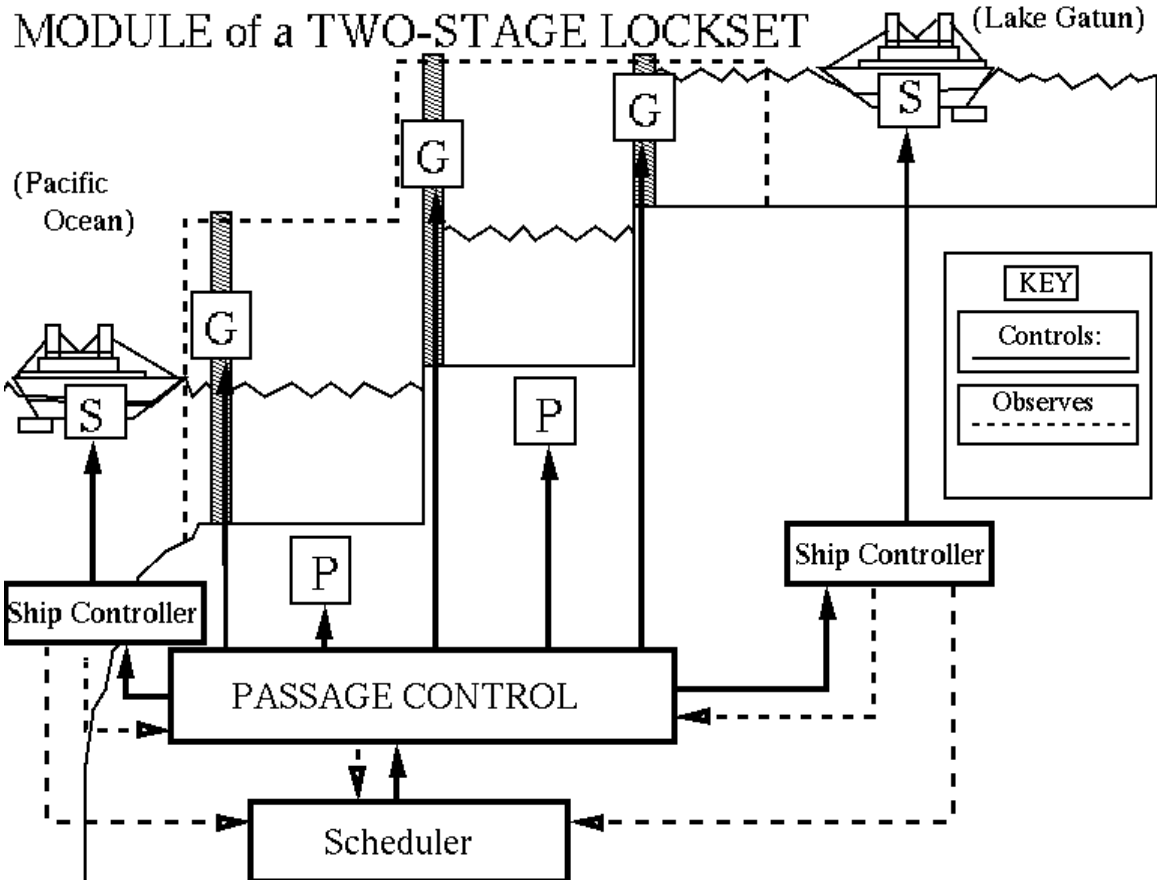
Bringing the Locks to Life:

Confident with a successful first model, the next steps would realize the brilliance and difficulties of a bottom-up programming process.

The first model was successful, but incomplete. The next step was to add in pumps, and gates, with different procedural controls for moving east to west (moving to a higher level), and moving west to east (lowering a ship to 'sea level').

The pumps and gates were made from simple two-step processes. The process GATE (processes are also labeled in capitals) consisted of "opengate" and "closegate," and PUMP contained "pumpup" and "pumpdown". These building blocks allowed us to make multiple GATEs and PUMPs using "tags" like east:GATE and west:GATE for the single lock system, and then low:PUMP, high:PUMP and middle:GATE for the double lock system (a double lock requires a third middle gate). Tags are part of the power of the LTSA program, and using them cut down on the time and the number of lines of code needed to complete a practical model, but later they would provide some hurdles to overcome.

Now the single and double lock systems had life, with the gates and pumps putting the real-life elements into the systems; however these systems needed to be harnessed, and therefore a controller was needed.



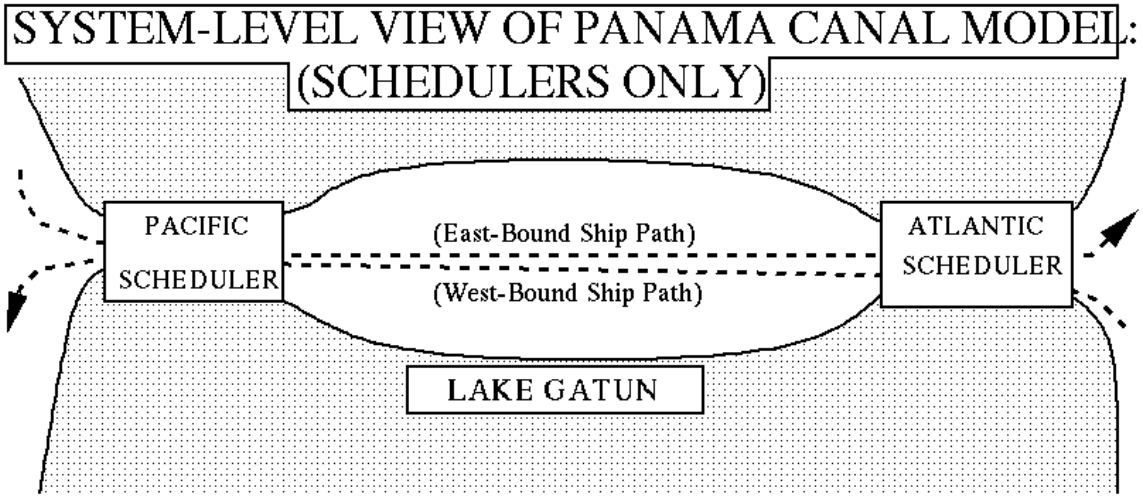
Controllers are used to make sure that each independent concurrent process works in synchronization with the others to perform the tasks of moving a ship up or down through the locks. The controller for each lock system was built with multiple starting conditions, on the basis that once one starting condition was met, a series of actions would be initiated to perform the EAST-WEST or WEST-EAST passing procedure.

When describing a controller, one must choose which direction the ships are being elevated above sea level. In the following description, the ships traveling westbound ships are being elevated, and the eastbound ships are descending. This controller is used for the Atlantic-side lockset.

The four starting conditions for a lock are east[ID].acquire, reseteast, west[ID].acquire, and resetwest. The conditions east[ID].acquire, and west[ID].acquire are attained when the water levels inside the locks are at the right levels for passage either from east-west (water level is low in both locks) or from west-east (water level is high in both locks). The reseteast, and the resetwest conditions are for passing east to west or west to east when the levels are not correct and the water levels have to be reset before passage procedures can commence. All procedures either feed into EAST-WEST passing procedures or WEST-EAST passing procedures, once the water level is correct. Through shared actions, which must be executed at the same time between the controller and either a gate or a pump, each action of pumping and opening/closing the gates occurs in the correct order.

Composing at a System Level:

The System Level composition of connecting two locksets together to form a canal went through many completed stages, not all of which were reader friendly. Using the tagging system, like when creating multiple gates and pumps, the entire lockset system could be uniquely copied and slightly altered to create an accurate canal system; everything but the ships.



One could not simply tag two unique copies of the original lockset created. The double lock system with gates and pumps could be tagged, since the gates and pumps would still operate the same way, but the controllers and the schedulers needed to be copied and altered.

The new controller had to be redesigned to pass the eastbound ships down through the lockset and the westbound ships up through the lockset on the Atlantic side, which is the opposite, or mirror effect of the original controller. Thanks to the design of the original controller, this meant switching the two sets of controls for east and west passage, and switching the "pumpups" to "pumpdowns" and vice versa. Changing the controllers would later spawn the idea of truly modularizing the canal model, for easier conversion, since getting the conversion right was simple in practice, but became hard to debug due to nomenclature and a non-streamlined structure.

The Scheduler was very simple to convert, since its purpose is just to read the scenario and decide which ship to pass through. After using copy and paste, and changing the name of the scheduler, the only step to full conversion was switching the water level variables, which determine fairness, and whether reset procedures are required. In the new scheduler, when a west bound ship would say it had cleared the lock, the scheduler would switch its water level monitor to up, since west bound ships move up through the mirrored lock, as oppose to the original, which would lower westbound ships down to sea level.

The ships also required special attention in building the canal model, since they not only needed to be able to get through the canal, but also needed to go through the locksets in the correct order.

The first hurdle was programming the ships to interact with both locks. Inserting a method called SHIP_PROG, standing for "ship program," did this, a middle step in creating the two convoys of ships (eastbound and westbound). SHIP_PROG had to be tagged as an intermediate step, otherwise there would be four total sets of convoys interacting with one lock each, as opposed to two sets that interact with both locks in series. However the "interacting in series" part of that definition was only attainable through processes called "physical properties."

Physical properties are part of the modeling system, and are built in to enforce real life restrictions into the model. The physical properties for the east- and westbound ships were different, since they entered the canal system from different sides, but in both cases, physical properties (can't use they as they refers to ships) kept the ships from conceptually jumping over a lockset into Lake Gatun and descending through a lockset before ascending through one. This sounds preposterous and it is; in real life such controlling systems are discarded as being unnecessary, but in a programmed model, the physical properties must be installed.

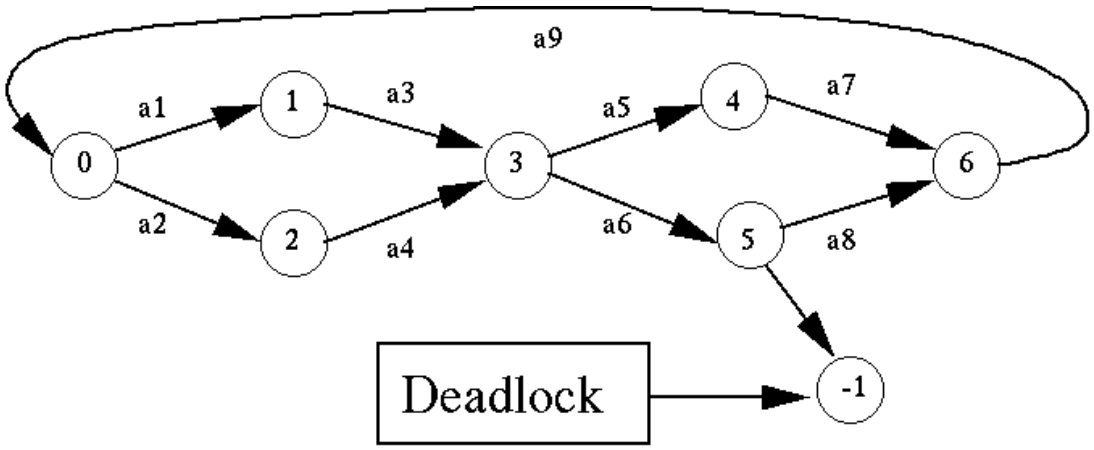
Proofing the Canal:

Finished with the creation of the canal model, it was necessary to prove that the model worked. This meant proving not only that the model never locked up (reached a state from which it could not move on to any other states), but that the model never allowed any catastrophes to occur, and that when run, all ships that desired to pass through the canal could do so.

When the full canal model was compiled completely there were 148,274 different states. Normally, with small programs, one can check one's program in LTSA by viewing the graphical form of LTSA and searching for deadlocks, or progress errors (progress

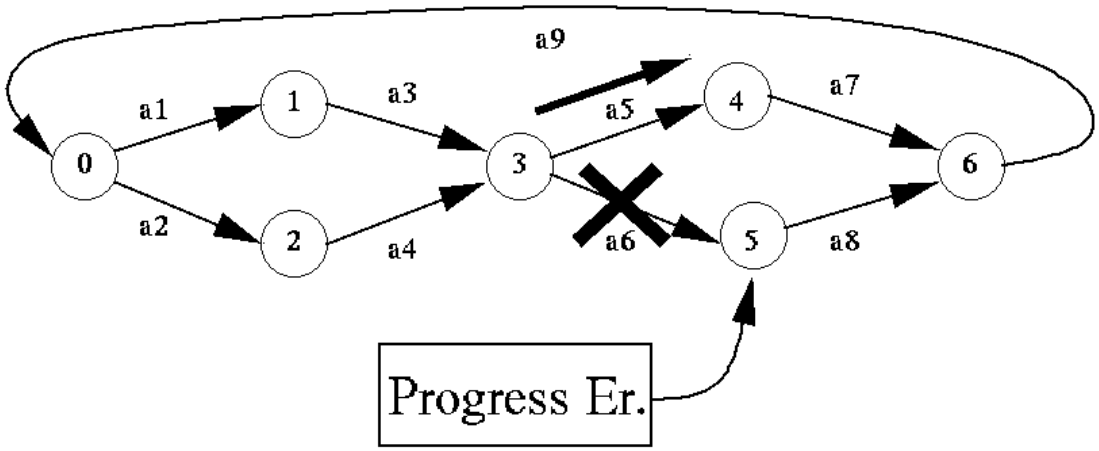
errors amount to finding out whether a ship will actually make it through the lock every time). However, with an eighth of a million states, a different and more powerful method was needed to prove that the canal model would work every time, in all situations.

LTSA has the methods for combing large models for deadlocks, safety specifications, and progress processes, in LTSA's checking feature. Because of the shared actions used in LTSA to make models with synchronization, single processes alone can have deadlocks, but when composed in a system with a monitor or controller, the deadlock is unattainable, and therefore eliminated. For this reason LTSA was built with a checking feature to see if a deadlock occurs within a large system. This becomes extremely useful when checking for realistic properties of a model, and ensuring that catastrophes do not occur.



After eliminating all deadlocks within a system, the system can then be checked for progress errors, or errors in achieving desirable states. LTSA does not allow you to check for progress errors unless deadlocks are eliminated, since a deadlock is a type of progress error in itself. In the most basic terms, if a system has deadlocks, it doesn't function, but even if a system functions, it may not function properly, and that's where progress checking picks up the slack.

A progress error occurs when a desirable state is unattainable, due to the composing of multiple systems together. When controlling a system, one wants to make sure that the system progresses from start to finish, and completes its mission every time the system is cycled, no matter what scenarios are influencing the system. The road system in a neighborhood is built so that every household can exit and enter the neighborhood using the road system. Although not all roads in a neighborhood will connect to one another, there is a path to/from every household.



LTSA has a built-in checker that walks through all the transitions between states, in every combination possible, to make sure that all actions are performable if the system were run under every possible condition. This does not equate to every possible state, since states are eliminated through controlling processes, just that in some order every action will occur. The desirable states can be described with progress descriptions adding to the checking feature through specific progress checks. The only things not covered in a progress check are undesirable states, and for those one needs to use "properties."

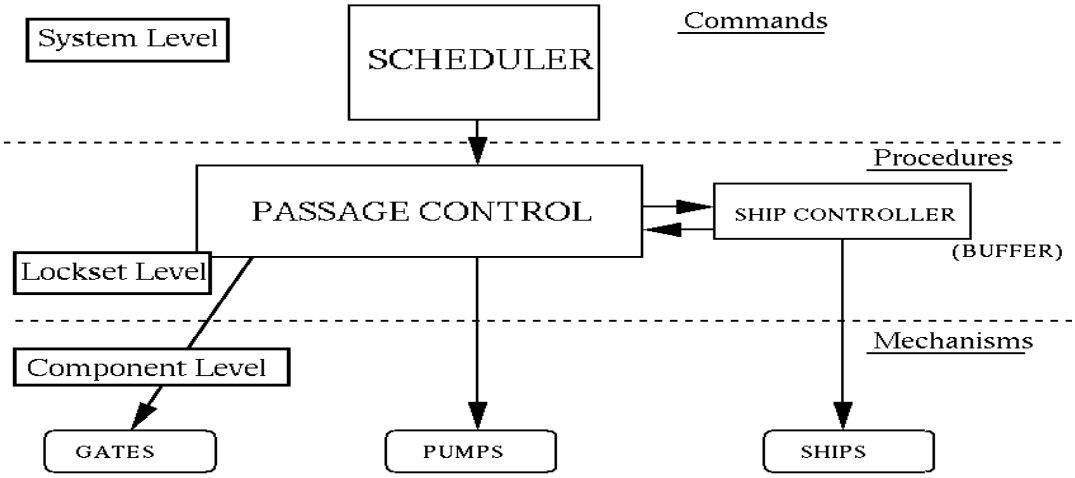
Properties are constant monitors on the system, which ensure that actions do not occur in the wrong order under any circumstance. This means that undesirable states can be checked for, which for the canal model equated to ensuring that floods do not occur within the locksets.

Properties are deterministic, and describe the exact sequence of events, which must always occur in an order. It is not a controlling mechanism, but a checking system, which ensures that the controlling system is working properly. As the last stage in proofing the canal model, and the most complicated due to its deterministic nature, in a system with fairness that can handle all non-determined scenarios, properties were the hardest step in the checking system to complete. However, after learning and discovering more techniques for writing in LTSA, the properties were defined, and the system was proven to be completely fail-safe.

Modularizing Methods:

Modularizing the canal model was Dr. Austin's idea, in order to make the model more acceptable and usable by others in the systems engineering field. This was the last alteration to the canal model, after proofing and testing other methods of control. The realization is that the controlling structure in the lockset could be made universal, easily manipulated, and viewable from multiple depths of system analysis.

Modularizing involved making the lockset simpler in description, and streamlining the controller. The gates were changed from "{east, middle, west}.GATE" into "{low, middle, high}.GATE," making the process of creating a "mirror" second lockset quick and simple in description. Since the "high.GATE" will always be the gate on top, and "low.GATE" will always be the gate on bottom, the controller could be streamlined.



Streamlining the controller meant two things: making the controller have universal ascending and descending procedures, and making the controllers easier to edit. The universal ascending and descending procedures (ASCEND and DESCEND) were simple to make after the locksets' nomenclature had been changed, using the LTSA techniques learned throughout the summer. The next part of streamlining was making the controller more easily edited for conversion, and it was done with the knowledge that the editing would only need to occur to the ship commands, since they are the only foreign objects in the system. Therefore, the SHIPCONTROL controller was created, to be separate from and interact with the PASSAGECONTROL, which handle the gates and pumps.

The final product of modularizing the canal, was a model and controlling system that could be used to create a three, four, or more lockset enabled canal system within one hour of starting with the original modularized lockset.

Interpretation:

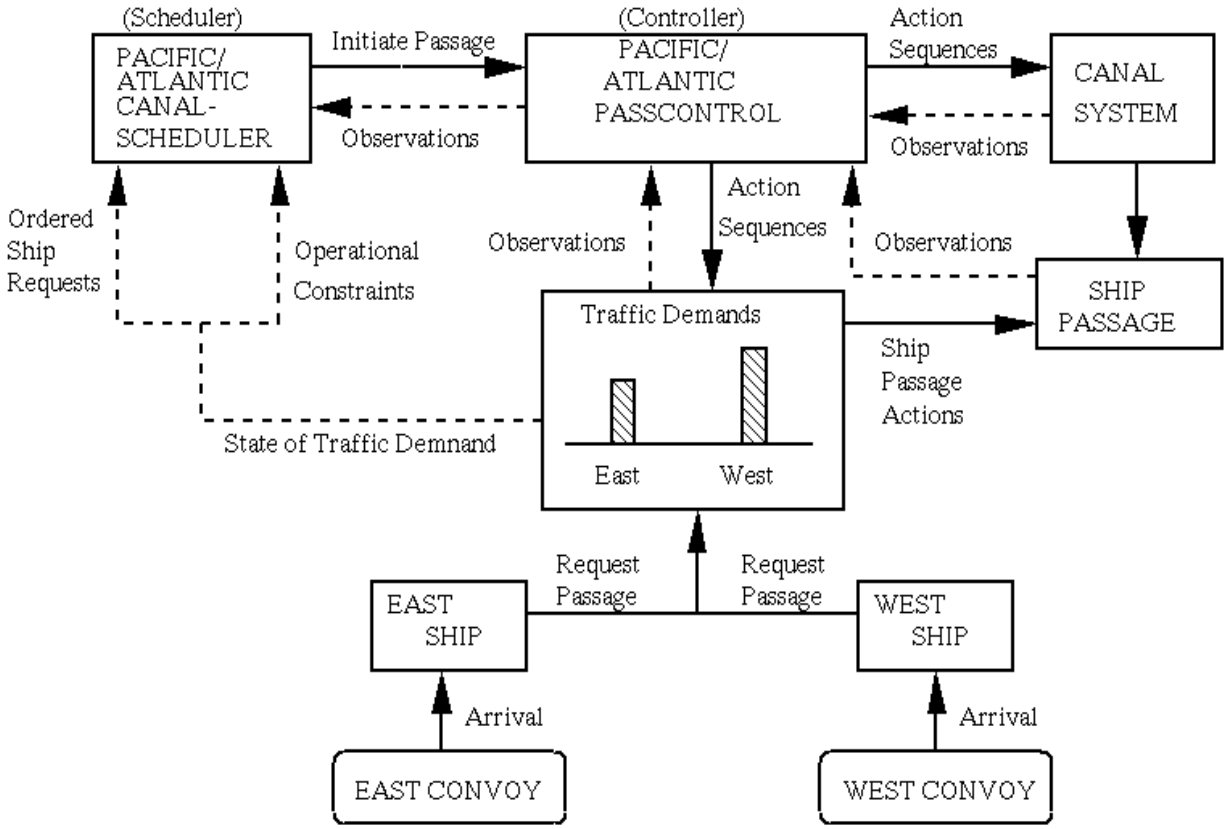
Once the model was completed, it was time to reflect and understand what had actually been accomplished, and how timing was not a part of it.

This model does not rely on timing to synchronize actions, and in that way it is elegantly straightforward from a controlling perspective. Interpreting what this model is actually doing can help clarify how such a simple idea can soundly trump methods with more complexity.

When describing the FSP (Finite State Processes language) programming in LTSA one refers to the different transitions from state to state as actions, therefore it is important to understand what is meant by the word "action." Actions can in fact be communications, or signals, or traditional performing of tasks. Although it was never used in this model, actions can be no action whatsoever, also known as a "wait" action.

In the context of this model, it is best to look at the SHIP process for understanding how one object in a system performs multiple different types of actions. SHIP is composed of the action set: "arrive->request->acquire->enterlock1->enterlock2->exitlock2->clear->(repeat)." The four actions "arrive," "enterlock1," "enterlock2," and "exitlock2," all perform the tasks of moving the ship. The three actions "request,"

“acquire,” and “clear,” are communications, or parts of communication with the monitor and/or controller. The story is that the ship arrives at the lockset, requests passage, acquires passage permission, enters the first lock in front of it, enters the second lock in front of it, exits the double locks, and then sends the signal that it has cleared the lockset. The ships that pass through a double lock perform seven actions, but only four of those actions actually affect the ship itself.



Understanding the definition of action used in LTSA is essential for comprehending how a system model works, but if one is going to create new system models and incorporate them into real life systems, one must understand how synchronization through shared actions can replace timing.

To display how shared actions replace timing, one can look at a ship as it passes through the lockset within the canal model. The ship, the monitor, and the controller share the actions performed to pass a ship through a lockset. In this explanation the SHIP process is a ship, and the controller and monitor are computer, sensor, and communication network systems in the canal.

When the SHIP process performs the action “request” it is shared by the monitor, and in fact is a communication to the monitor that then uses that information to create a queue for the ships on that side of the lock. The acquiring of the use of the lock, performed by the action “acquire,” must be able to be performed by the controller and the monitor; meaning the monitor must give the ship permission to pass through, and the controller must be ready to send the ship through as well. The SHIP process will want to then enter the first lock, but it will not be able to until the controller allows it to be sharing the action, which in reality is communication from the controller to the ship that the lockset is prepared for the ship to enter the first lock. The controller will make the

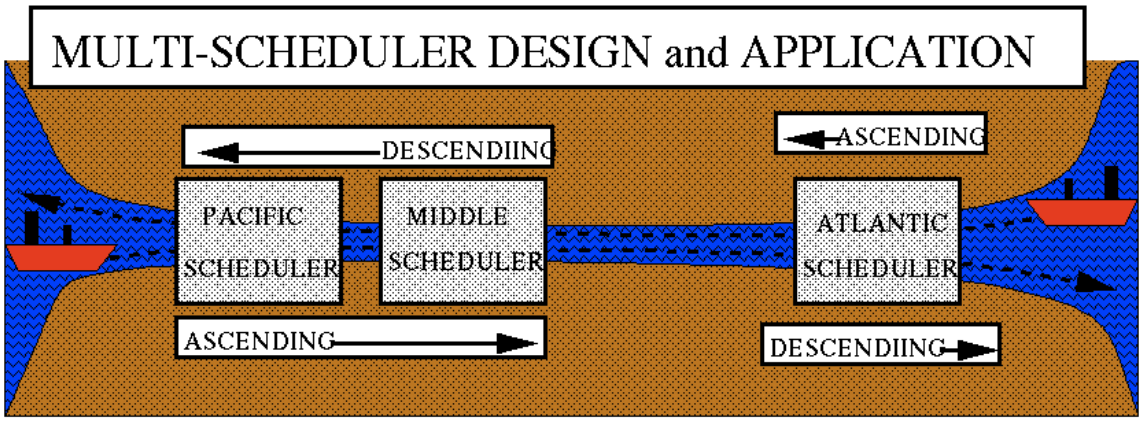
ship wait by not sharing the action until the controller has told the pumps to pump the water down or up to the correct levels and has told the gates to open the gate in front of the ship. The controller is able to do this because the actions of “pumpup/pumpdown” and “opengate” precede the action “(ship).enterlock1” in the CONTROL process, and all preceding actions to a shared action must be performed within a process before the shared action may be performed. In JAVA this is called synchronized methods, and it is built into the JAVA-based program LTSA.

This waiting and sharing of actions only after preceding actions have occurred is the timing within the system, and the timing is perfect. All actions that must be performed before another particular action should be performed are carried out, and immediately following the completion of the preceding actions, the shared action is performed. There is no fat to cut, no error bounds, just simple perform task set A before B, and perform each task when capable.

Conclusion:

The Panama Canal program has proven how capable the LTSA program is at creating a truly fail-safe and fair controlling system in a larger dynamic concurrent behavioral system. The program’s ability to check for all possible types of errors, and its state-focused nature allow for it to be efficient and precise in helping to cultivate programs like the Panama Canal program made in this project. The most important part of using LTSA is to understand how it models and how it controls systems, but once clarity is reached, it is simple and quick to accomplish, which makes the program’s future possible applications in systems engineering as extremely promising.

The next step in the process in moving the LTSA program towards real-world applications is to use it to model even more complicated systems. Currently the program needs a lot of computing power to handle a maximum of three locksets in a program; therefore, future users of the program should expect to need a lot of memory to run larger programs.



It will be interesting to see how a state-based, not time-based, software controlling system will be received and implemented in the future. As the world continues to be more

informed of the failures of bridges, and airports, the world wants to be certain that our technologies are fail-safe. LTSA should play a major part in ensuring that safety, especially since time is relative and states are certain.