NTCP: A Grid Service for Remote Control Systems

Laura Pearlman[1], Mike D'Arcy[1], Carl Kesselman[1], Pawel Plaszczak[2]
[1]USC Information Sciences Institute, Marina del Rey, CA
[2]Argonne National Laboratory, Argonne, IL

1 Introduction

The Network for Earthquake Engineering Simulation (NEES) project aims to advance collaborative earthquake engineering research in the United States by improving facilities for physical and computational earthquake simulations and encouraging the sharing of data, facilities, and computational models.

Traditionally, earthquake engineers have simulated the effects of ground motion on structures using one of two basic approaches: using computational simulations or using physical simulations. More recently, earthquake engineers have begun doing *hybrid experiments*: coupled computational and physical simulations in which one part of a structure is modeled computationally and another part is modeled as a physical experiment. The computational and physical simulations are run simultaneously; the results of both the computational and physical component for one time-step are used to determine the inputs to each for the next time-step. Hybrid experiments have generally been tightly coupled, with the computational system and the physical control system communicating via a shared-memory backbone.

We have designed and implemented the NEESgrid Teleoperations Control Protocol (NTCP)[1] protocol and service to support distributed hybrid experiments that may include several computational simulations and several physical experiments. The initial version of NTCP supports only "slow" hybrid experiments (experiments without strict performance requirements).

Several of the design goals of NTCP were driven by properties of physical experiments: the specimens involved are often very large (weighing tens of tons) and take months to construct and install, and it is often difficult or impossible to "undo" a physical action (applying a force to an experiment specimen may cause a permanent, significant change to the properties of that specimen; the only way to reverse that change may be to construct and install a new specimen).

One design goal for NTCP was to support a common protocol for both physical experiments and computational simulations; this allows for development and testing of computational components while the physical components are being constructed, for simulation-only dry-runs of experiments, and for the possibility of replacing a physical experiment with a computational simulation in the event of a fault (such as a network partition) during a distributed experiment run.

Another design goal was to support fault recovery to the greatest extent possible: a distributed simulation should not fail because of a momentary network interruption or because a system hosting a client or computational simulation crashed; for this reason, we chose to implement at-most-once execution semantics in NTCP.

A final design goal was to allow for separate negotiation and execution phases, to allow a client application to verify that the actions proposed for a time-step are acceptable to all sites involved, before actually sending a request to take any physical action.

2 The NTCP Service

The NTCP service performs actions on *control points*. In general, a control point is a simply a point at which an action may be taken (or simulated). In a physical experiment, a control point is a physical point on the experiment specimen at which forces may be applied by actuators. In a physical experiment, there is often a many-to-one relationship between actuators and control points (for example, there may be actuators positioned to move the same control point along different axes).

2.1 Protocol and State Model

The NTCP protocol is transaction-based, and the three primary requests in the NTCP protocol (*propose*, *execute*, and *cancel*) manage the state of a transaction:

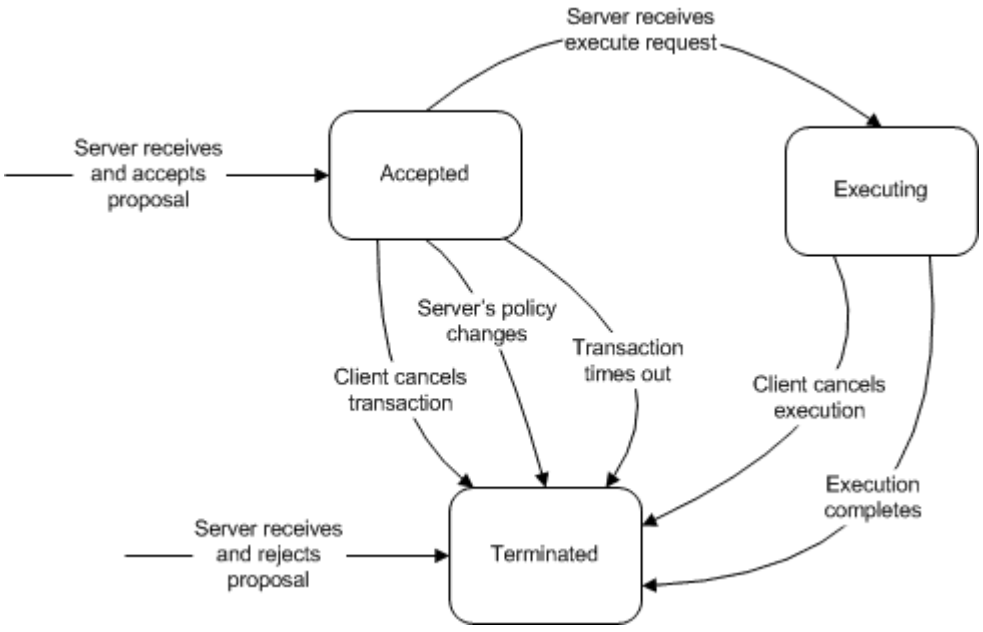


Figure 1: Transaction life-cycle

A transaction may be in one of three possible states: *accepted*, *executing*, or *terminating*. Transactions are created with the NTCP *propose* request, in which the client sends a proposal, consisting of a new transaction name, a set of control points and associated parameters that specify the proposed actions, and three timeout values. In a typical time-step, the client sends a *propose* request, which the server accepts, creating a new

transaction in the accepted state and replying to the client. The client then sends an *execute* request to the server, which will change the transaction's state to *executing* , begin the action (or computation) specified in the transaction, and reply to the client. Finally, when the action (or computation) is complete, the server will change the transaction's state to *terminated*. The execute request does not return the transaction results to the user (because it returns when execution is initiated, not when it completes); instead, transaction results are communicated to the user via service data.

Other circumstances may cause the transaction to become *terminated*: the server may reject the proposal (and create the transaction in the *terminated* state); the client may send a *cancel* request; the site's policies may change between the time that the transaction was created and the time the server receives the *execute* request.

The timeout values included in the proposal are the *proposal timeout* (if the proposal is received after this time, the server will ignore the proposal), the *transaction timeout* (if this time passes and no execute request has been received for this transaction, the transaction will time out and become *terminated*), and *transaction state timeout* (the time until which the server is expected to remember the transaction's state).

The server will reject duplicate requests: *propose* requests in which the transaction name is the same as the name of an existing transaction, and *execute* requests in which the named transaction is not in the *accepted* state.

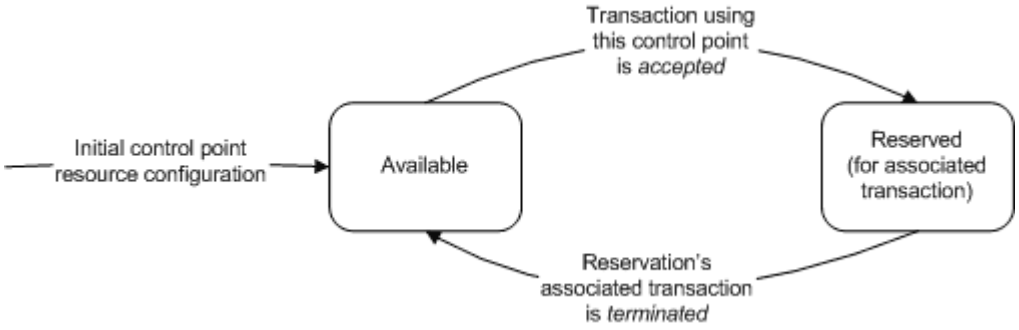


Figure 2: Control point state transitions

Control points have their own associated state; a control point may be either *reserved* (associated with a transaction that is not *terminated*) or *available*. A control point may not be associated with two active transactions at once; a proposal for a transaction involving a *reserved* control point will be rejected.

This state model guarantees that, for each control point, at most one transaction involving that control point is executing at any time, and that any two transactions that are executed on one control point are executed in the same order in which they were accepted. We do not make any guarantees about the order in which the actions involved in a single transaction are executed (e.g., if a transaction involves control points A and B, the action involving A may be executed before, at the same time as, or after the action involving B) or about what happens in the time between two transactions involving the

same client; if the latter kind of guarantee is desired, it could be provided via a community scheduler.

A typical time-step in a distributed experiment using NTCP includes a negotiation phase, in which the client sends proposals to all the NTCP servers involved in that time-step and an execution phase in which the client sends execute requests to those NTCP servers, and a verification phase in which the client receives the final results from the transactions (and possibly additional sensor readings from external sources) and calculates the desired behavior for the next time-step. If a proposal is rejected during the negotiation phase, the client may choose to cancel some of the outstanding proposals and send new proposals with different request parameters, repeating that process until a satisfactory set of proposals has been accepted.

There are several additional NTCP requests: *setParameters* and *getParameters* to set and query experiment parameters, *openSession* and *closeSession* to bring the control system associated with the server to a well-known state, and *getControlPoint* (to query the status of a control point, which may involve sending a request to a control system or simulation).

2.2 Service Data Elements

The state of each transaction is kept as a service data element that is created when the transaction is created and updated whenever the transaction changes state:

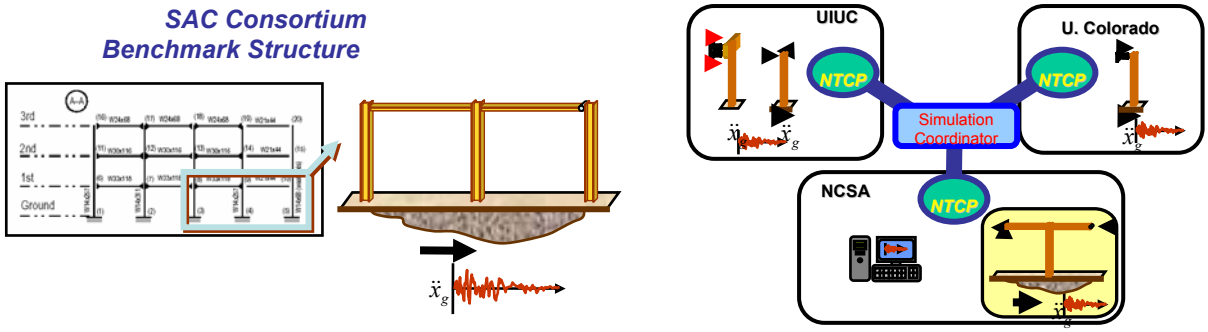
This service data element includes the transaction name, control point parameters, and timeouts that were specified in the proposal that created this transaction, the transaction state (*accepted*, *executing*, or *terminated*), and the name of the authenticated user who created the transaction. If the transaction entered the *executing* state, the time of that state transition is included; similarly, if the transaction is *terminated*, the transaction's termination time is also recorded. Finally, if the transaction executed successfully, the measured (or calculated) results.

NTCP maintains a SDE for each transaction (until that transaction's *transaction state timeout* has passed, and an additional SDE that contains this information about the transaction that was most recently created or changed.

3 Application Experience: the MOST Experiment

The first real test of NTCP was the Multi-site Online Simulation Test (MOST) experiment, designed by earthquake engineers at the University of Illinois at Urbana-Champaign and the University of Colorado[2]. The MOST experiment modeled part of a the frame from the interior of a (hypothetical) building, and involved a physical experiment at UIUC (with a full-scale model of a frame column), a physical experiment at the University of Colorado (with a full-scale model of another frame column) and a computational simulation at NCSA (modeling the rest of the partial frame). The full MOST experiment consisted of 1500 time-steps, which ran for approximately 5.5 hours. The computational simulations were written by earthquake engineers at UIUC and modified to use NTCP by an earthquake engineer at USC. The original simulations were

provided to us early, while we were still in the process of designing NTCP, and resulted in the addition of some NTCP requests (including the requests to set and query experiment parameters) that had not been part of our original design.



Our implementation of NTCP is based on the Globus Toolkit, version 3.0, and consists of a server that provides the core functionality (state management, etc.) and calls out to a *control plugin* (a Java class that implements the NTCP Control Plugin Interface[3]) and handles communication to the (site-specific) control system or backend simulation.

The MOST experiment involved a single client, called the *simulation coordinator*, which communicated with three NTCP servers: an NTCP server at UIUC configured to use a custom control plugin to control a Shore Western servo-hydraulic system, an NTCP server at NCSA configured to use a control plugin (called the *Mplugin*) to communicate with a Matlab simulation, and an NTCP server at the University of Colorado configured to use the Mplugin to communicate with a Matlab simulation, which in turn used Matlab's xPC product to control a servo-hydraulic system. The simulation coordinator was written in Matlab; it made calls to a Matlab toolbox (a library of Matlab functions) which in turn made calls to the NTCP Java client API.

For pre-experiment testing, we also ran a fourth NTCP server: a server at UIUC configured to use the Mplugin to communicate with a Matlab simulation that emulated the physical experiment at UIUC. We were thus able (by configuring the simulation coordinator to use one or the other of these NTCP servers) to switch between using a physical experiment or computational simulation at UIUC. We were also able to switch between physical and computational modes at Colorado by changing a parameter on the Colorado simulation; thus, we could test with any combination of physical and computational simulations at UIUC and Colorado. We did the vast majority of pre-experiment testing using only computational simulations.

A dry-run of the MOST experiment (using servo-hydraulics at UIUC and Colorado and a computational simulation at NCSA) ran successfully to completion in about 5.5 hours. The MOST experiment itself ran to step 1494 (of 1500), at which point an NTCP request timed out, and the simulation coordinator terminated the simulation. In fact, we had outlined a strategy for client-side retries in the event of such a timeout but have not yet implemented that strategy.

4 Security Considerations

The possible risks associated with a physical experiment are higher than the risks associated with most computing applications – accepting a "bad" request from a malicious (or simply broken) remote application could damage equipment or experiment systems, or in some cases even lead to serious injury. Our implementations of NTCP and related software were built using commonly-availably tools and run on commodity operating systems, and thus cannot be guaranteed to be completely secure. We have recommended that the equipment sites should have appropriate non-software-based controls and procedures in place to safeguard their equipment and personnel.

5 Future Work

The NEES project is in the planning stages of a program of "experiment-based deployment" of NTCP, in which the experiment sites will install NTCP, integrate it with their local systems, and test it in MOST-like multi-site experiments.

We plan to implement more fault-tolerance features, including persistent state in the server, retries in the client API; we are also plan to provide tools and guidance for developers of plugins and backend systems to improve the fault-tolerance throughout the system.

NTCP has some obvious parallels to OGSI-Agreement[4]: the propose/accept exchange is essentially the creation of an agreement; however, NTCP agreements would be much shorter-lived than agreements for most other services. We plan to investigate the possibility of migrating NTCP to the OGSI-Agreement framework.

The current version of NTCP supports "slow" hybrid experiments but does not support "fast" experiments with near-real-time performance requirements. We are working now with earthquake engineers to try to support these experiments; in the NTCP model, this will involve putting some fine-grained experiment control logic into the backend system (that is, the local control system).

6 Acknowledgements

Ben Clifford, Sridhar Gullapalli, Erik Johnson, Peter Lane, Ravi Madduri, Narutoshi Nakata, Andrei Reinhorn, Bill Spencer, Bozidar Stojadinovic, and many other members of the NEESgrid community have provided valuable input on the design of NTCP. The MOST experiment pictures that appear here are derived from drawings by Bill Spencer. Narutoshi Nakata wrote the Matlab simulations used in the MOST experiment; Erik Johnson wrote the Matlab toolboxes used to communicate with NTCP and modified the MOST simulations to use them. Ravi Madduri wrote the NTCP plugin that was used to control the Shore Western servo-hydraulics used in the MOST experiment.
This work was supported primarily by the George E. Brown, Jr. Network for Earthquake Engineering Simulation (NEES) Program of the National Science Foundation under Award Number CMS-0117853

[1] NEESgrid Teleoperation Control Protocol (NTCP). L. Pearlman, M. D'Arcy, E. Johnson, C. Kesselman, P. Plaszczak. NEESgrid Technical Report NEESgrid-2003-07. September, 2003

[2] MOST experiment report (not yet published)

[3] A Plugin Interface for an NTCP Server. L. Pearlman, M. D'Arcy, C. Kesselman, P. Plaszczak. NEESgrid Technical Report NEESgrid-2003-07. September, 2003

[4] K. Czajkowski, A. Dan, J. Rofrano, S. Tuecke, M. Xu,. Agreement-based Grid Service Management (OGSI-Agreement) Version 0. Global Grid Forum document. 2003.