Experimental Software Framework for Hybrid Simulation

Andreas Schellenberg & Stephen Mahin

With contributions from:

Gregory Fenves, Yoshikazu Takahashi, Frank McKenna

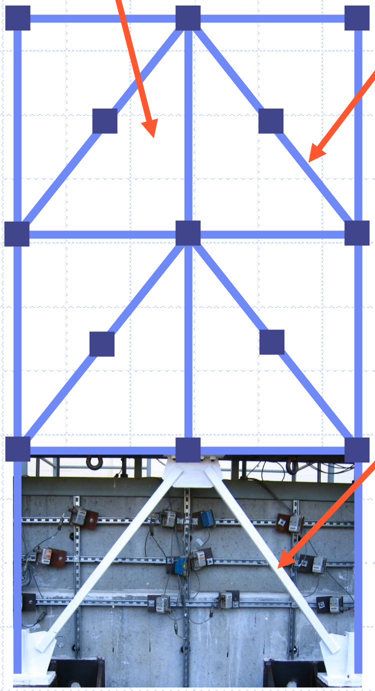
Department of Civil and Environmental Engineering
University of California, Berkeley

Hybrid Simulation

$$\mathbf{M} \cdot \ddot{\mathbf{u}} + \mathbf{C} \cdot \dot{\mathbf{u}} + \mathbf{P_r}(\mathbf{u}) = \mathbf{P}(t)$$

Dynamic Loading

- Seismic
- Wind
- Blast/Impact
- Wave
- Traffic



analytical model of structural energy dissipation and inertia

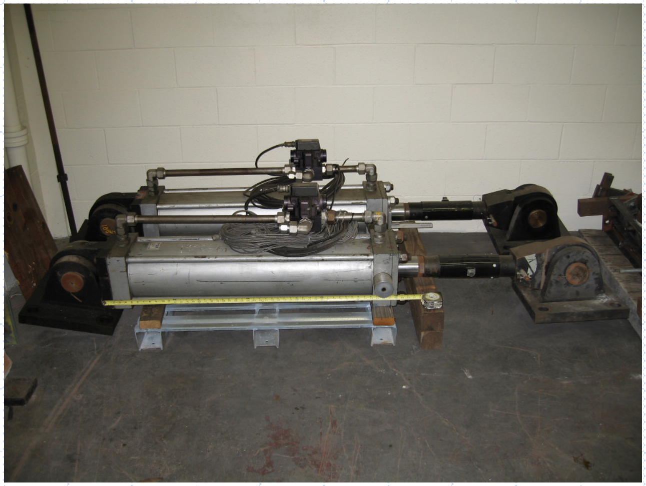
physical model of structural resistance

Hybrid Simulation

- Model the well understood parts of a structure in a finite element program on one or more computers
- Leave the construction and testing of the highly nonlinear and/or numerically hard to model parts of the structure in one or more laboratories
- Can also be seen as an advanced form of component testing, where boundary conditions are correctly imposed

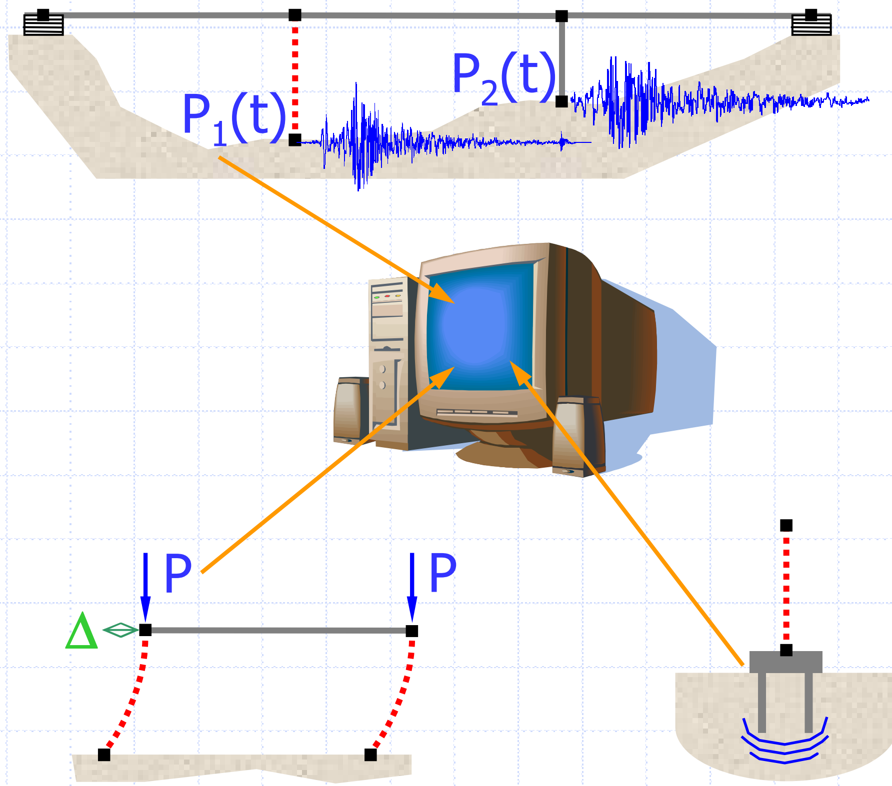
Advantages

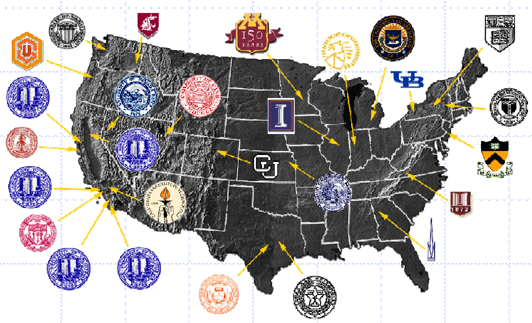
- Enables dynamic testing of full-scale specimens
- Quasi-static testing equipment sufficient
- Fewer restrictions on size, weight and strength of a specimen



Advantages

- Geometric nonlinearities, three-dimensional effects, multi-support excitations and soil-structure interactions can be incorporated into the analytical model





- Geographically distributed testing is made possible

Main Challenge

- Lack of a common framework for development and deployment
- Problem specific implementations which are site and control system dependant
- Such highly customized software implementations are difficult to adapt to different structural problems

→ Need a robust, transparent, adaptable, and easily extensible framework for research and deployment

OpenFresco

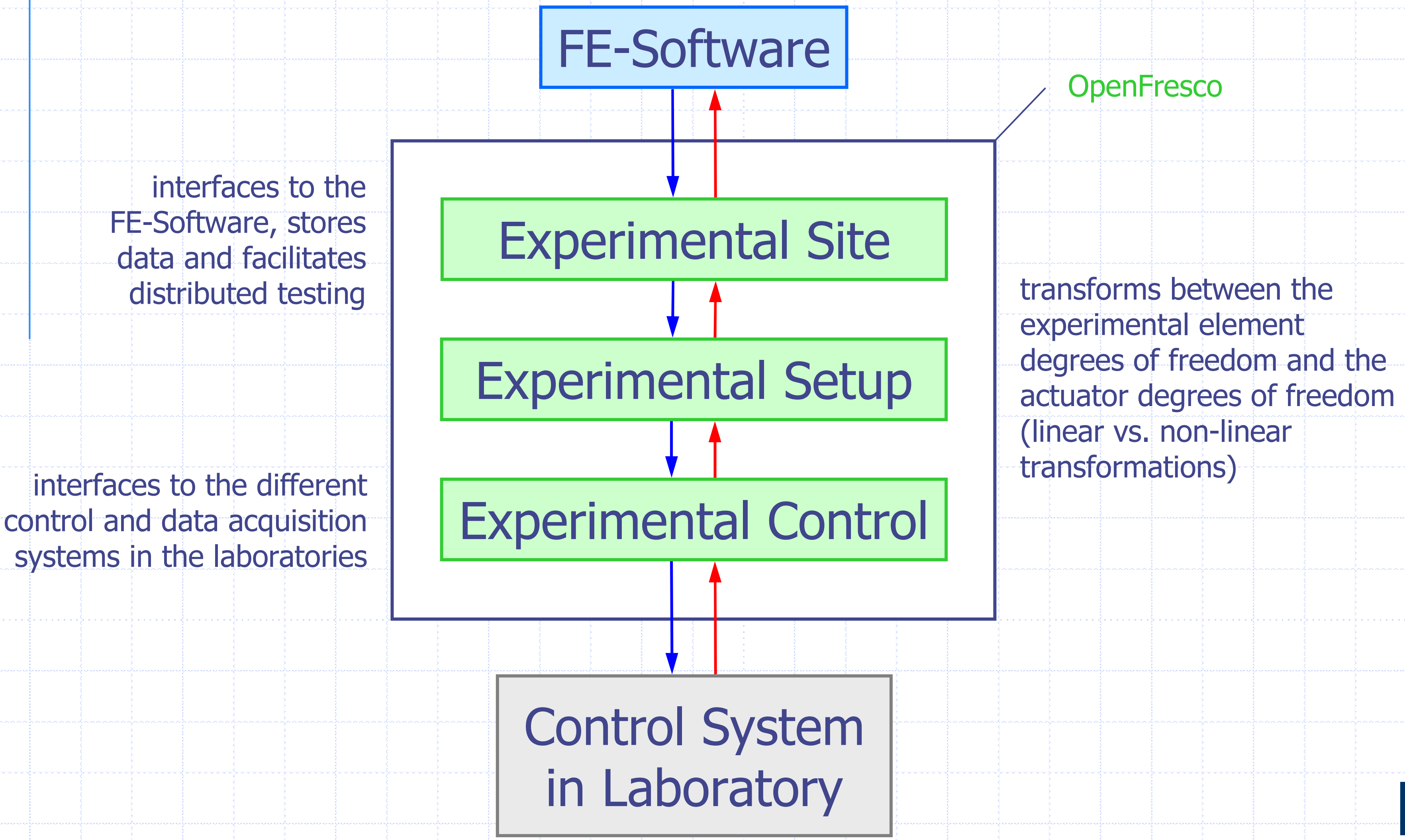
- Open source Framework for Experimental Setup and Control
- Enable domain researchers to carry out Hybrid Simulations without specialized knowledge
- Allow IT and hybrid simulation specialists to extend frontiers of methodology, focusing only on their portions of interest
  - Facilitate additions and extensions for new equipment and procedures
- ➡ Object-oriented programming approach

NEES-Compliant Deployment of OpenFresco

- No modification of numerical simulation framework is needed, other than the addition of new finite elements representing physical elements tested
- Calls to obtain element stiffness, restoring force and other parameters made just like they would be in a numerical analysis, except they are executed physically somewhere on a local or wide area network
- OpenFresco mediates in a modular and highly structured manner instructions between numerical simulation computer(s) and laboratory equipment

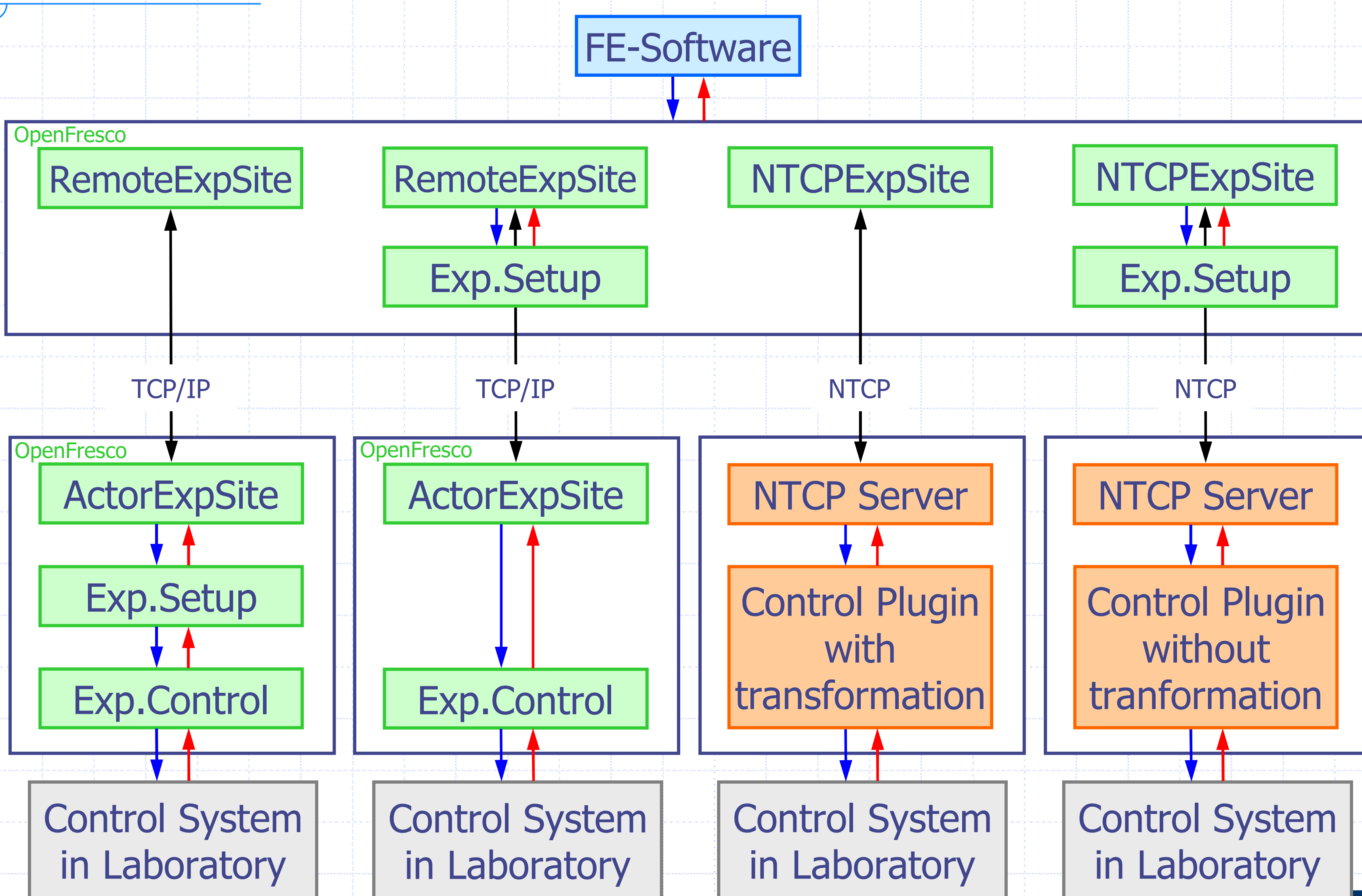
OpenFresco Components

local deployment

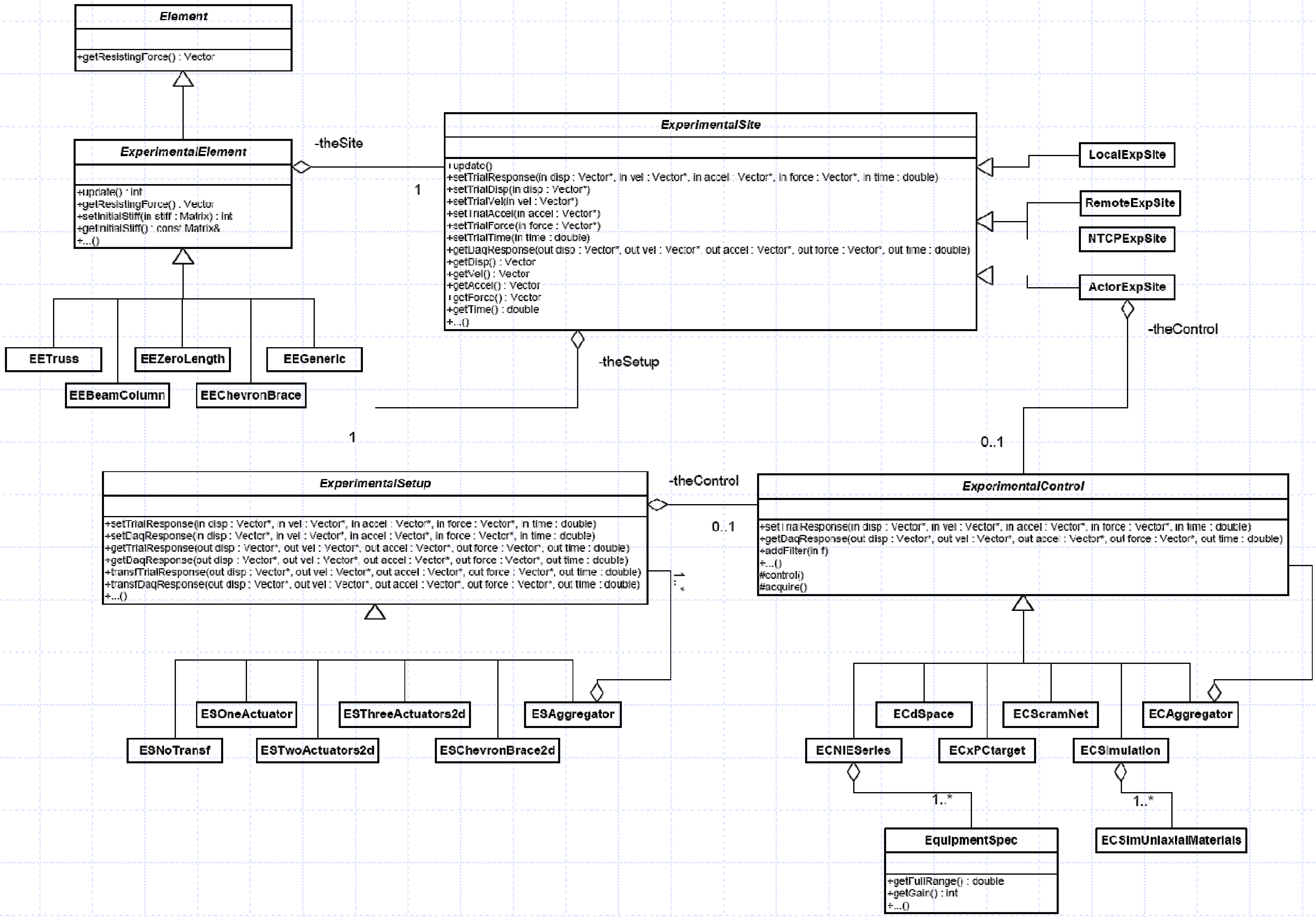


OpenFresco Components

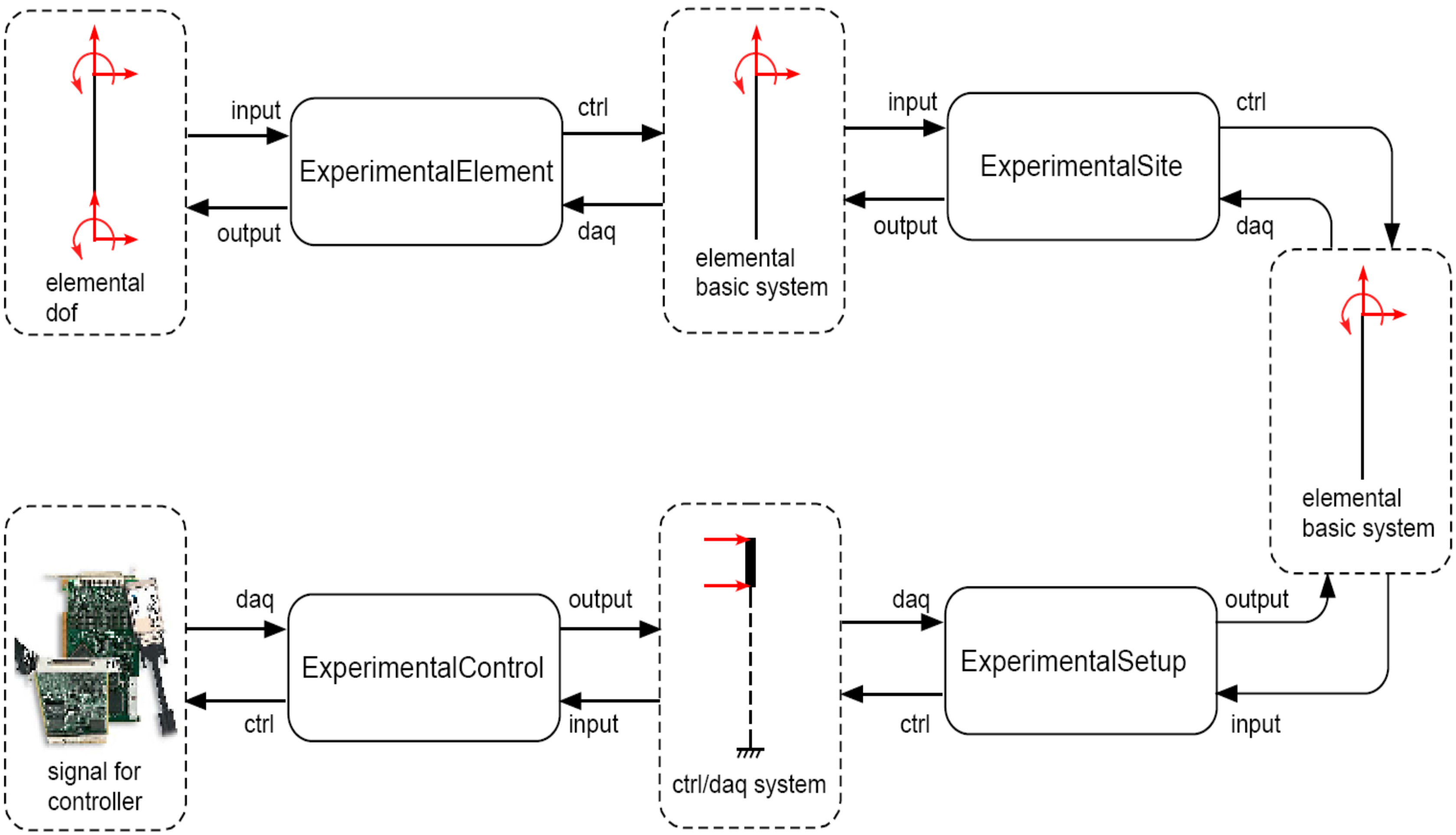
network deployment



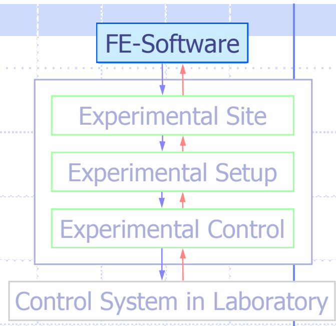
OpenFresco Class Diagram



OpenFresco Data Transformation

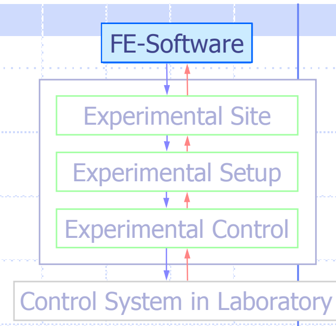


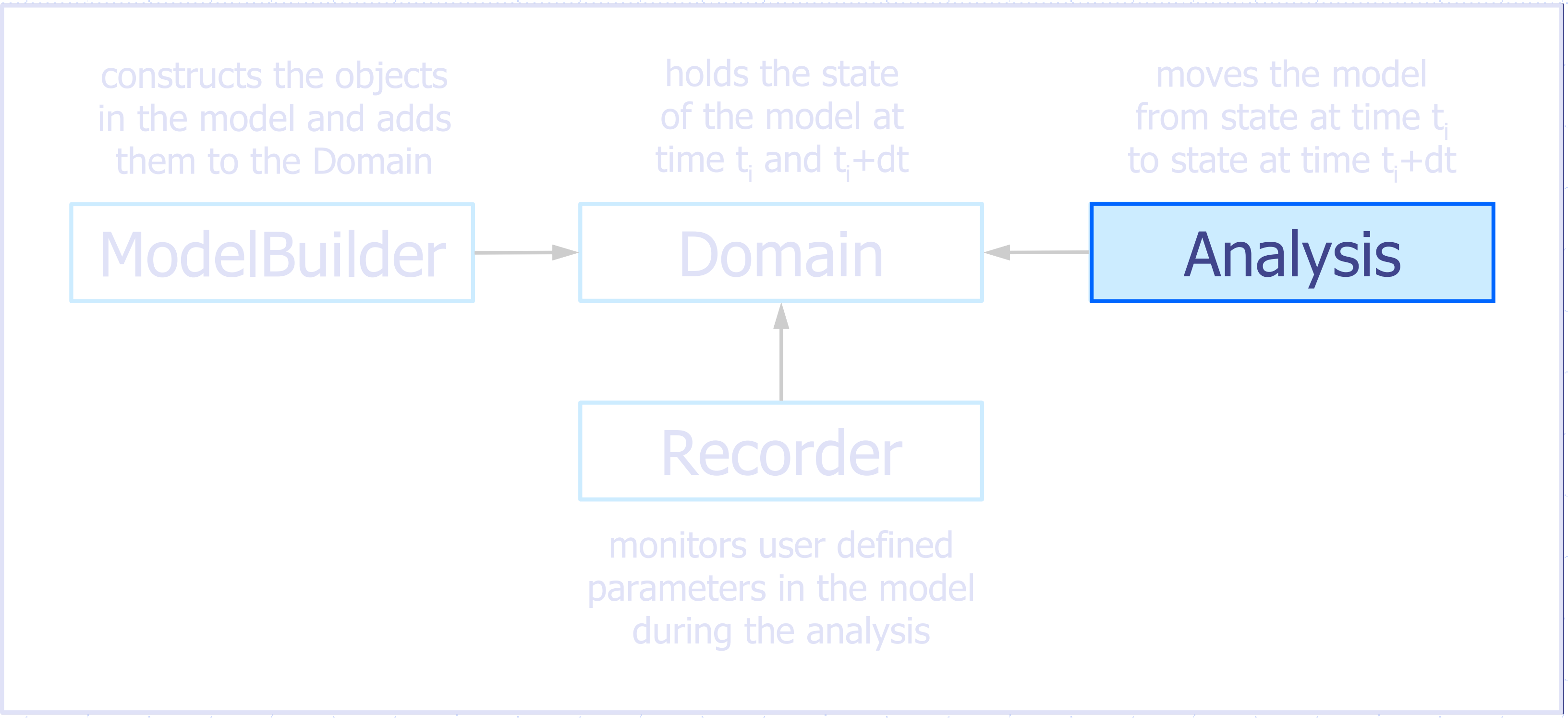
Finite-Element Software



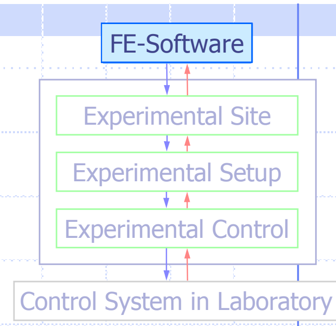
- Currently using OpenSees; however, nearly any software allowing the addition of elements and having the appropriate communication channels can be used
- Furthermore, a Matlab client which is able to interface with OpenFresco is under development as well

OpenSees Components





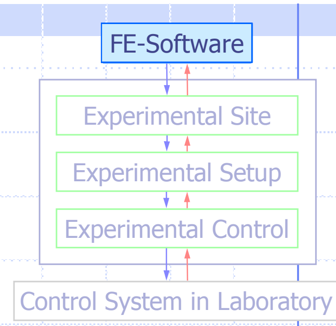
Direct Integration Methods



$$\mathbf{M}\cdot\ddot{\mathbf{u}}_{\mathbf{n}} + \mathbf{C}\cdot\dot{\mathbf{u}}_{\mathbf{n}} + \mathbf{P}_{\mathbf{r}}\left(\mathbf{u}_{\mathbf{n}}\right) = \mathbf{P}\left(t_n\right)$$

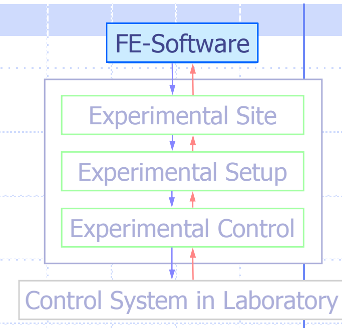
- Mass matrix M is often singular
  -> second order differential equation infinitely stiff -> fully implicit numerical methods
- Make as few function calls as possible
- Use constant Jacobian in the numerical methods since tangent stiffness is not available

Direct Integration Methods

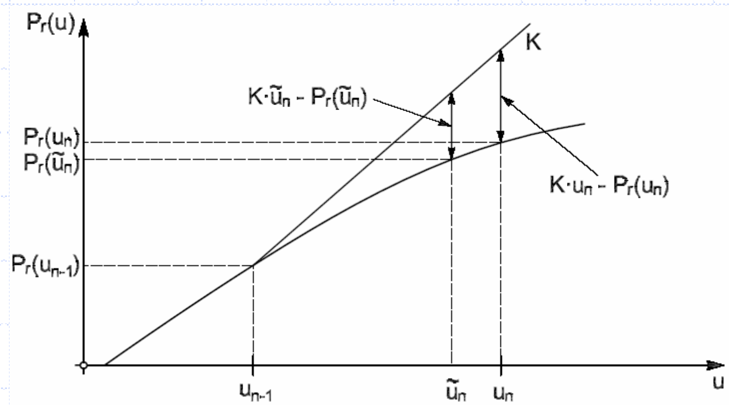


- Explicit Integrators
  - explicit Newmark Method
  - Central-Difference Method
  - explicit Alpha-Method
  - generalized explicit Alpha-Method
- Implicit Integrators
  - Newmark Method
  - Alpha-Method
  - generalized Alpha-Method
  - Collocation Method

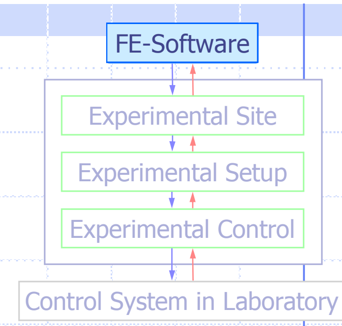
Direct Integration Methods

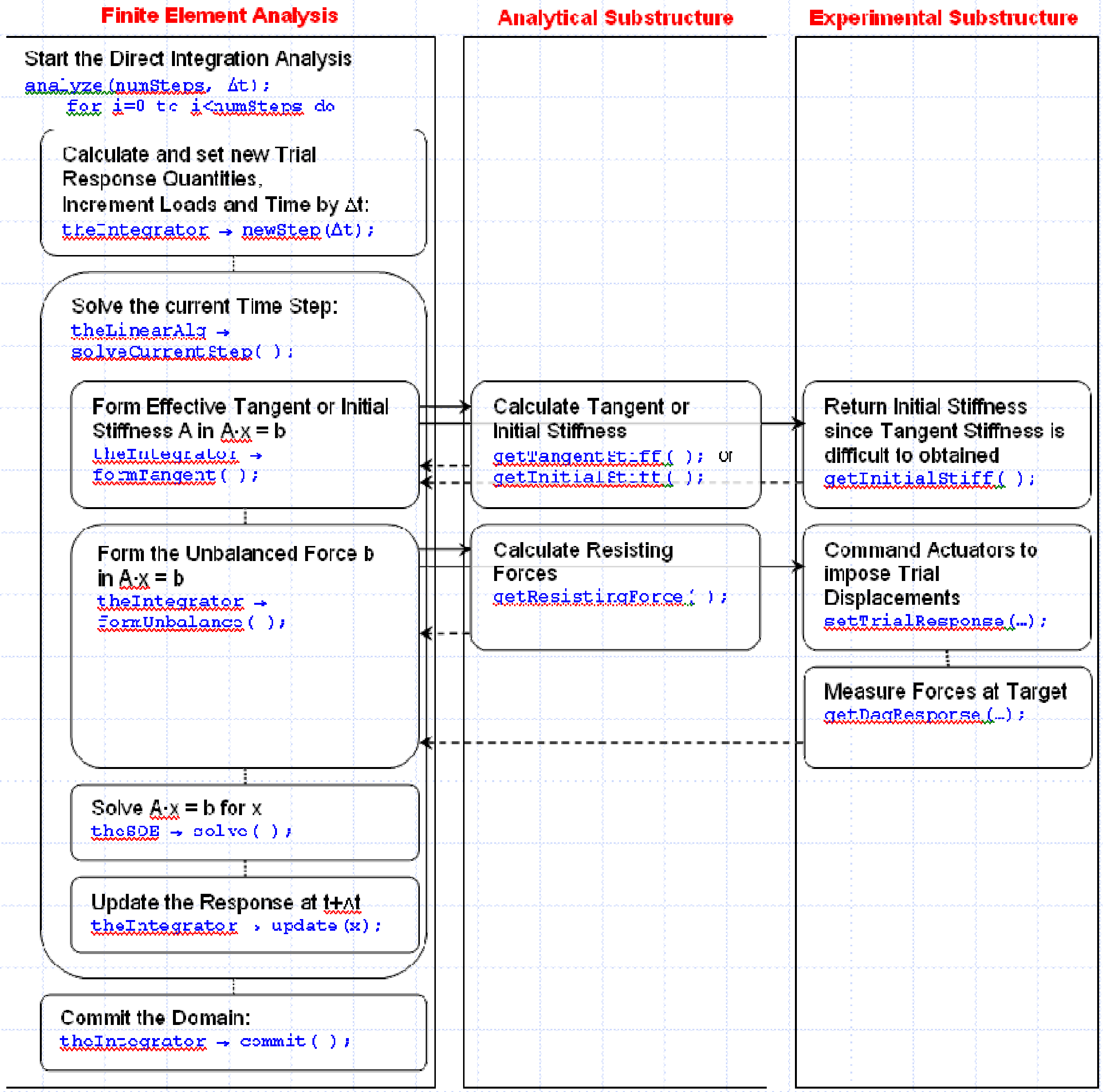


- Implicit Integrators with sub-stepping (constant number)
  - Newmark-HybridSimulation Method
  - generalized Alpha-HybridSimulation Method
  - Collocation-HybridSimulation Method
- Predictor-Corrector Integrators
  - Alpha-OS Method
  - generalized Alpha-OS Method

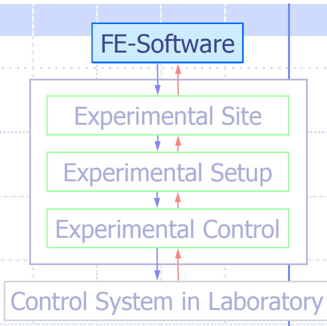


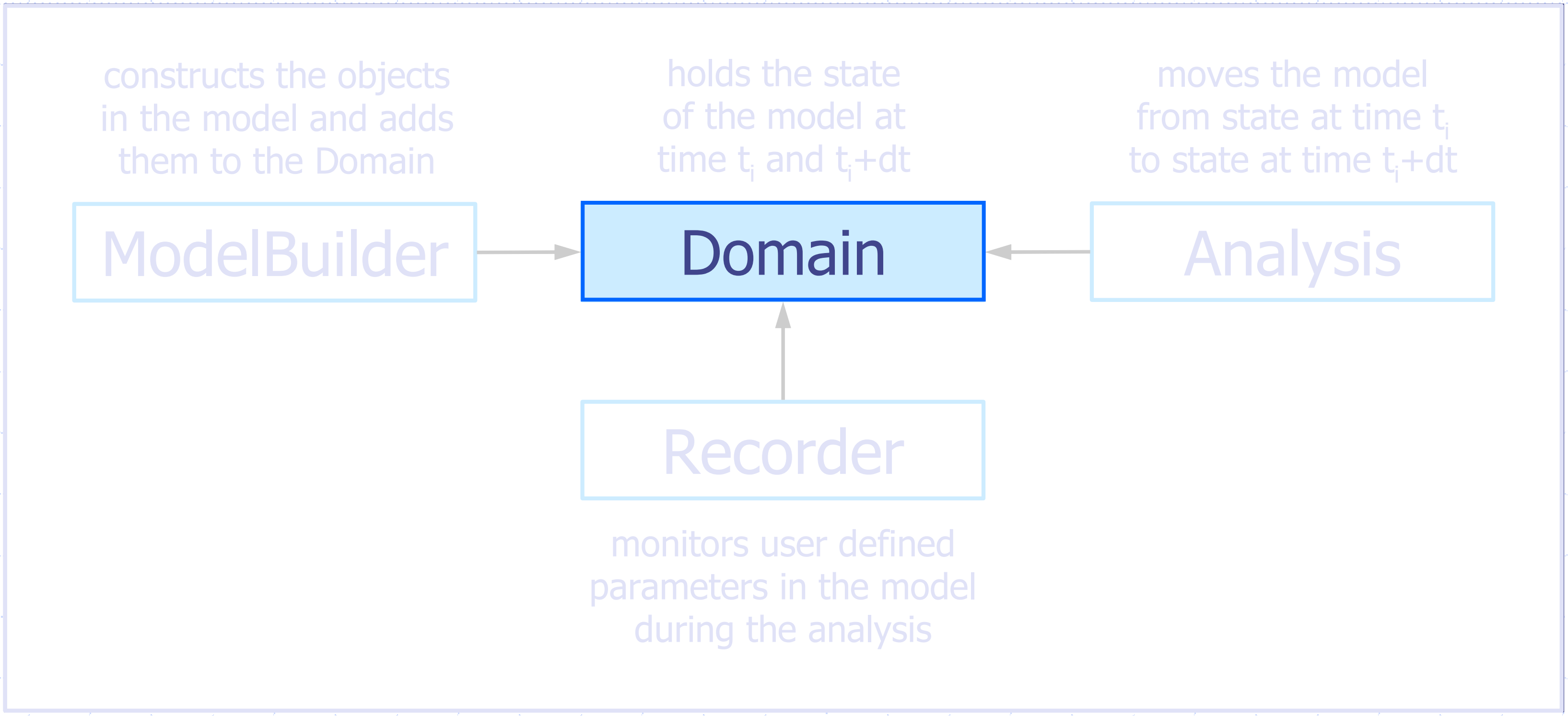
Hybrid Simulation Procedure



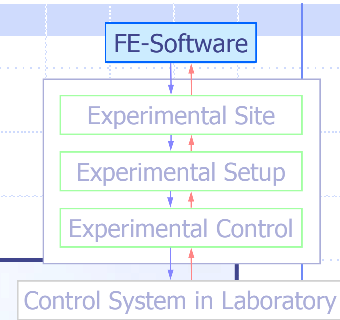


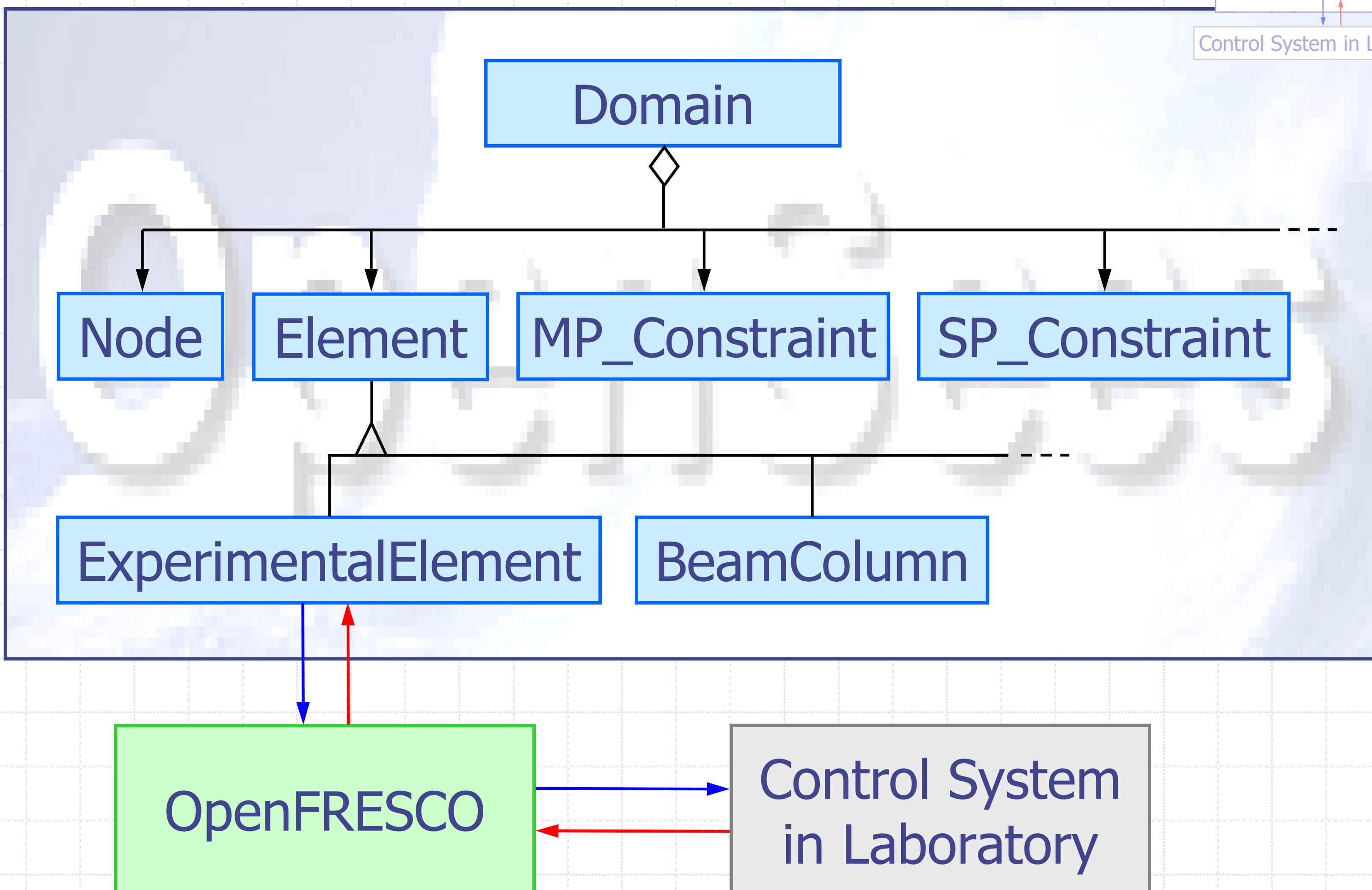
OpenSees Components



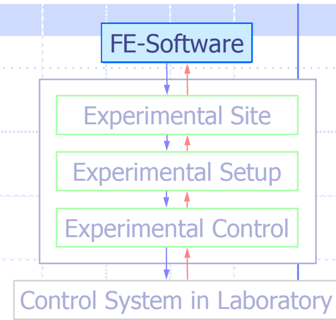


OpenSees Components





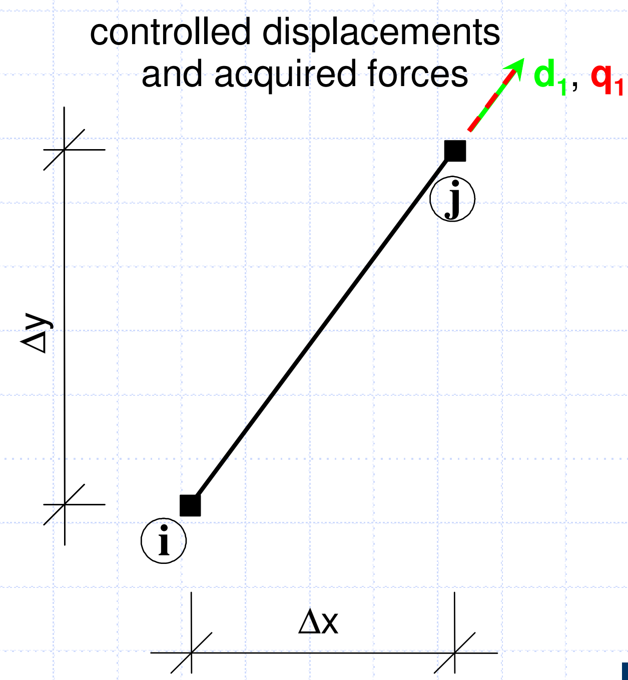
Experimental Elements

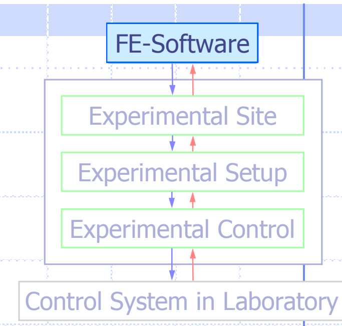


1) EETruss (2D,3D)

```
element expTruss $eleTag $iNode $jNode $siteTag
        -initStif $Kij <-iMod> <-rho $rho>
```

`$eleTag`	unique element tag
`$iNode,$jNode`	end nodes
`$siteTag`	tag of previously defined site object
`$Kij`	initial stiffness matrix element (1 x 1)
`-iMod`	flag for I-Modification (optional, default=false)
`$rho`	mass per unit length (optional, default=0.0)



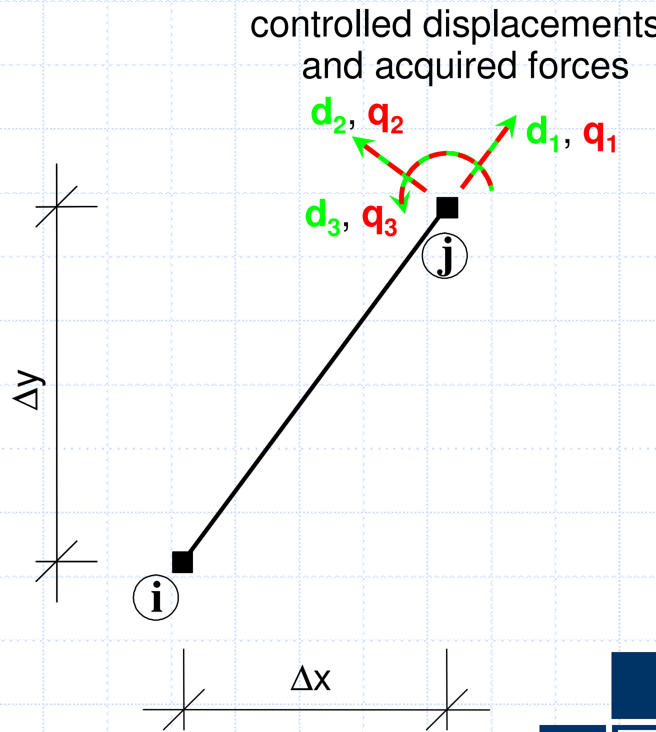


Experimental Elements

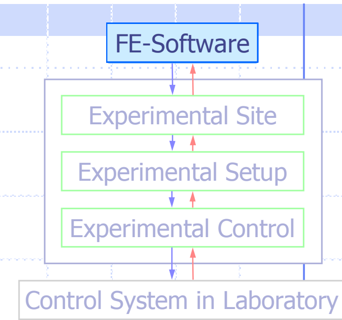
2) EEBeamColumn (2D,3D)

```
element expBeamColumn $eleTag $iNode $jNode
        $siteTag $tranTag -initStif $Kij <-iMod>
        <-rho $rho>
```

$eleTag	unique element tag
$iNode,$jNode	end nodes
$siteTag	tag of previously defined site object
$tranTag	tag of previously defined crd-transf object
$Kij	initial stiffness matrix elements (ndf x ndf)
-iMod	flag for I-Modification (optional, default=false)
$rho	mass per unit length (optional, default=0.0)



Experimental Elements

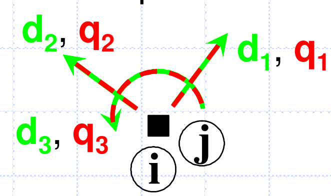


3) EEZeroLength (2D,3D)

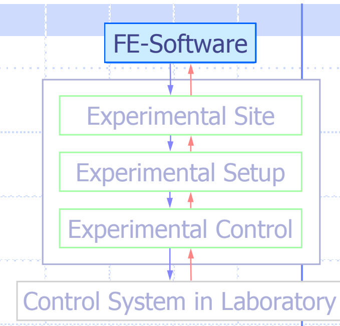
```
element expZeroLength $eleTag $iNode $jNode $siteTag
        -dir $dirs -initStif $Kij <-iMod>
        <-orient $x1 $x2 $x3 $y1 $y2 $y3>
```

`$eleTag`	unique element tag
`$iNode,$jNode`	end nodes
`$siteTag`	tag of previously defined site object
`$dirs`	force directions (1-3,1-6)
`$Kij`	initial stiffness matrix elements (nDir x nDir)
`-iMod`	flag for I-Modification (optional, default=false)
`$xi, $yi`	local x- and y-axis (optional, default=X,Y)

controlled displacements and acquired forces



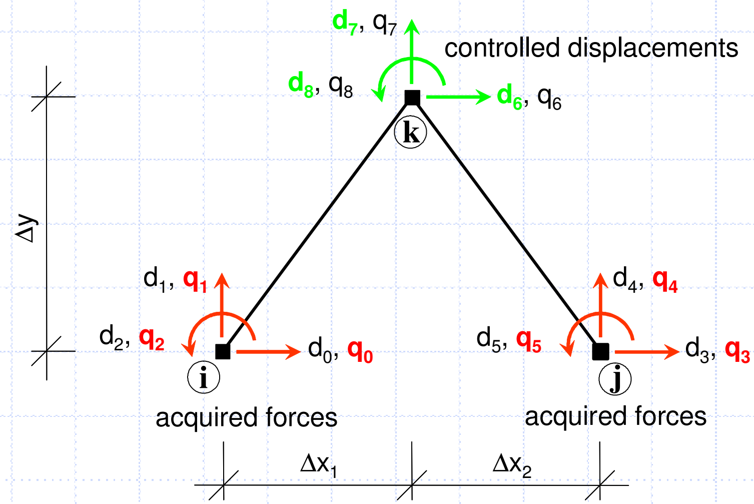
Experimental Elements



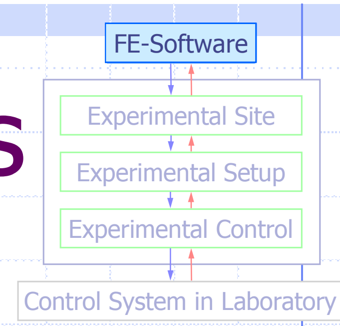
4) EEChevronBrace (2D,3D)

```
element expChevronBrace $eleTag $iNode $jNode
        $kNode $siteTag -initStif $Kij <-iMod>
        <-rho1 $rho1> <-rho2 $rho2>
```

`$eleTag`	unique element tag
`$iNode,$jNode` `$kNode`	end nodes
`$siteTag`	tag of previously defined site object
`$Kij`	initial stiffness matrix elements (ndf x ndf)
`-iMod`	flag for I-Modification (optional, default=false)
`$rho1,$rho2`	masses per unit length (optional, default=0.0)



Adding Experimental Elements



Public Member Functions

Constructor and Destructor

```
ExperimentalElement(int tag, int classTag, ExperimentalSite &theSite);
virtual ~ExperimentalElement();
```

Methods dealing with nodes and number of external dof

```
virtual int getNumExternalNodes(void) const = 0;
virtual const ID &getExternalNodes(void) = 0;
virtual Node **getNodePtrs(void) = 0;
virtual int getNumDOF(void) = 0;
```

Method to obtain basic dof size; equal to the max num dof that can be controlled

```
virtual int getNumBasicDOF(void) = 0;
```

Methods dealing with committed state and update

```
virtual int commitState(void);
virtual int update(void);
virtual bool isSubdomain(void);
```

Methods to set and to obtain the initial stiffness matrix

```
virtual int setInitialStiff(const Matrix& stiff) = 0;
const Matrix &getInitialStiff(void);
```

Methods to return the damping and mass matrices

```
virtual const Matrix &getDamp(void);
virtual const Matrix &getMass(void);
```

Methods for applying loads

```
virtual void zeroLoad(void) = 0;
virtual int addLoad(ElementalLoad *theLoad, double loadFactor) = 0;
virtual int addInertiaLoadToUnbalance(const Vector &accel) = 0;
virtual int setRayleighDampingFactors(double alphaM, double betaK, double betaK0, double
betaKc);
```

Methods for obtaining resisting force (force includes elemental loads)

```
virtual const Vector &getResistingForce(void) = 0;
virtual const Vector &getResistingForceIncInertia(void);
```

Methods for obtaining information specific to an element

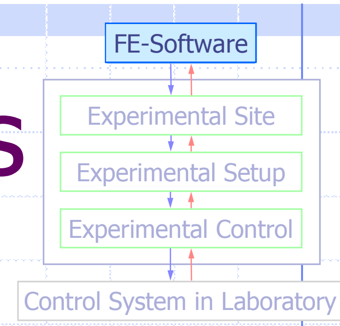
```
virtual Response *setResponse(const char **argv, int argc, Information &eleInformation);
virtual int getResponse(int responseID, Information &eleInformation);
```

```
int update( )
```

```
int setInitialStiff( )
```

```
const Vector
&getResistingForce( )
```

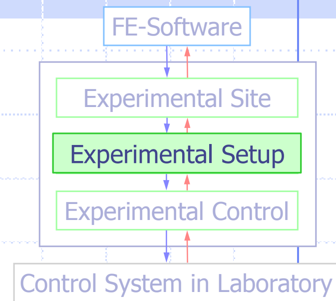
Adding Experimental Elements




```
int EEBeamColumn2d::update(void)
{
    // get current time
    Domain *theDomain = this->getDomain();
    double time = theDomain->getCurrentTime();

    // update the coordinate transformation
    theCoordTransf->update();

    // determine dsp, vel and acc in basic system A
    const Vector &db = theCoordTransf->getBasicTrialDisp();
    const Vector &vb = theCoordTransf->getBasicTrialVel();
    const Vector &ab = theCoordTransf->getBasicTrialAccel();

    // transform displacements from basic sys A to basic sys B
    static Vector dbB(3), vbB(3), abB(3);
    dbB = T*db;
    vbB = T*vb;
    abB = T*ab;

    if (dbB != targDsp)  {
        // save the target displacement
        targDsp = dbB;
        // set the trial response at the site
        if (isCopy == false)
            eSite->setTrialResponse(dbB, vbB, abB);
    }

    return 0;
}
```

```
int EEBeamColumn2d::setInitialStiff(const Matrix &kbinit)
{
    kbInit = kbinit;

    if (kbInit.noRows() != 3 || kbInit.noCols() != 3)  {
        opserr << "EEBeamColumn2d::setInitialStiff() - "
            << "matrix size is incorrect for element: "
            << this->getTag() << endln;
        return -1;
    }

    // transform the stiffness from basic sys B to basic sys A
    static Matrix kbInitA(3,3);
    kbInitA.addMatrixTripleProduct(0.0, T, kbInit, 1.0);

    // transform the stiffness from the basic to the global system
    theInitStif.Zero();
    theInitStif = theCoordTransf->getInitialGlobalStiffMatrix(kbInitA);

    return 0;
}
```

```
const Vector& EEBeamColumn2d::getResistingForce()
{
    theCoordTransf->update();

    // zero the residual
    theVector.Zero();

    // get measured resisting forces
    static Vector qB(3);
    qB = eSite->getForce();

    // add initial forces
    for (int i=0; i<3; i++)  {
        qB(i) += q0[i];
    }

    if (iMod == true)  {
        // get measured displacements
        static Vector measDsp(3);
        measDsp = eSite->getDisp();

        // correct for displacement control errors using I-Modification
        qB -= kbInit*(measDsp - targDsp);
    } else  {
        // use elastic axial force if axial force from test is zero
        if (qB(0) == 0)  {
            qB(0) = kbInit(0,0) * targDsp(0);
        }
    }

    // transform from basic sys B to basic sys A
    q = T^qB;

    // Vector for reactions in basic system A
    Vector p0Vec(p0, 3);

    // determine resisting forces in global system
    theVector = theCoordTransf->getGlobalResistingForce(q, p0Vec);

    // subtract external load
    theVector.addVector(1.0, theLoad, -1.0);

    return theVector;
}
```

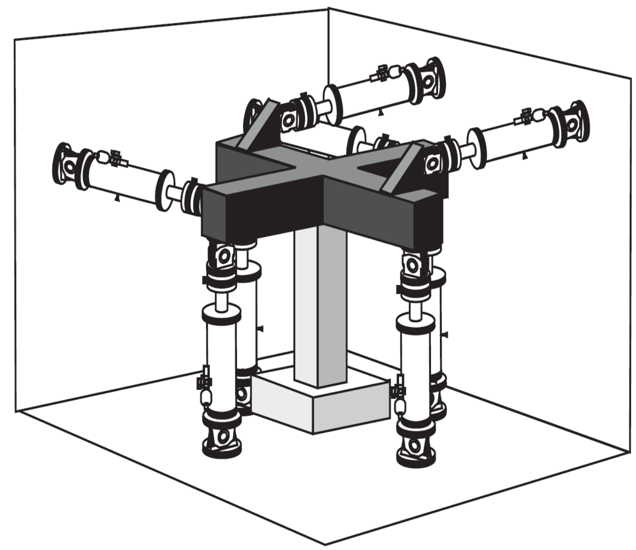


Experimental Setups

1) ESNoTransformation

```
expSetup NoTransformation $tag $ctrlTag -dir $dirs
         <-dspCtrlFact $dspCF> ...
```

$tag	unique setup tag
$ctrlTag	tag of previously defined control object
$dirs	directions (1-6)



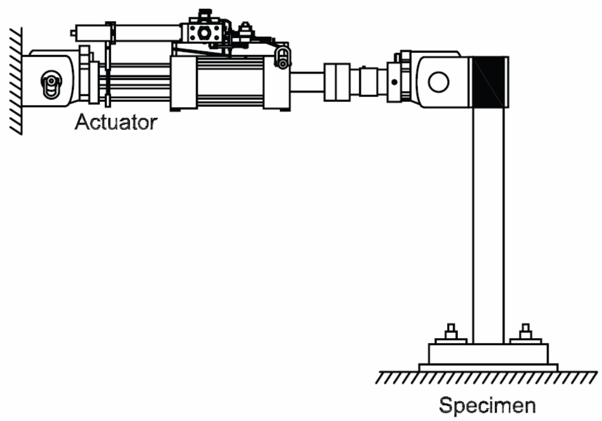
FE-Software
Experimental Site
Experimental Setup
Experimental Control
Control System in Laboratory

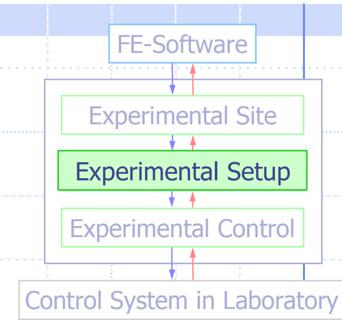
Experimental Setups

2) ESOneActuator

```
expSetup OneActuator $tag $ctrlTag $dir
         <-dspCtrlFact $dspCF> ...
```

`$tag`	unique setup tag
`$ctrlTag`	tag of previously defined control object
`$dir`	direction (1-6)



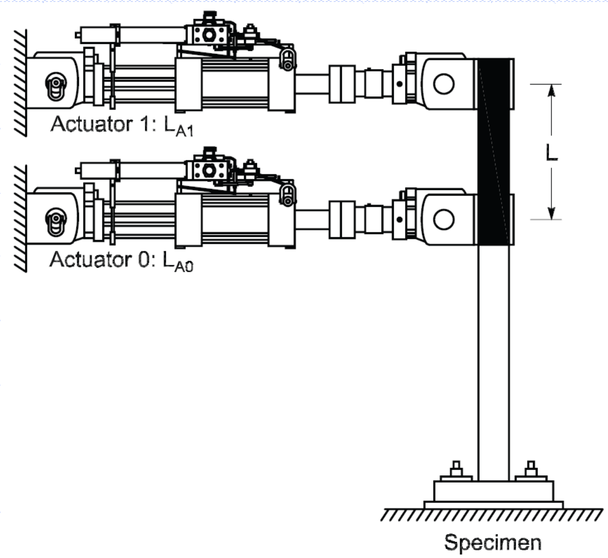


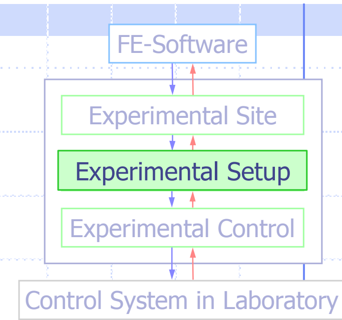
Experimental Setups

3) ESTwoActuators

```
expSetup TwoActuators $tag $ctrlTag $nlGeomFlag
         $La0 $La1 $L <-dspCtrlFact $dspCF> ...
```

`$tag`	unique setup tag
`$ctrlTag`	tag of previously defined control object
`$nlGeomFlag`	nonlinear geometry flag
`$La0`	length of actuator 0
`$La1`	length of actuator 1
`$L`	length of rigid link



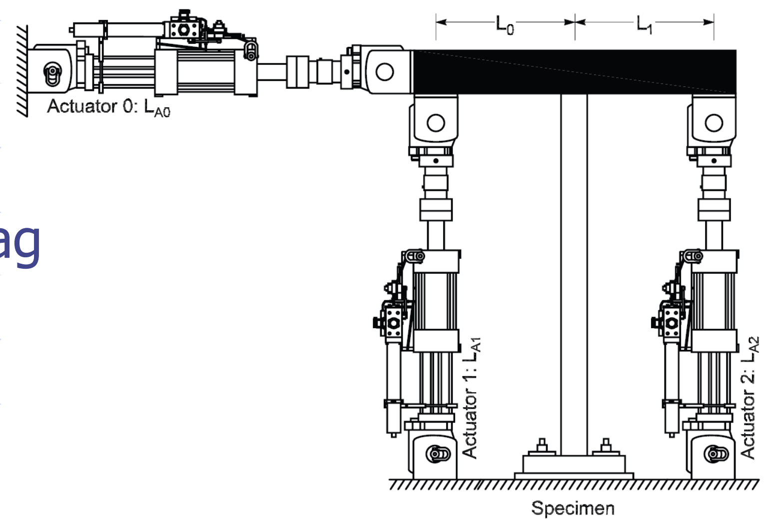


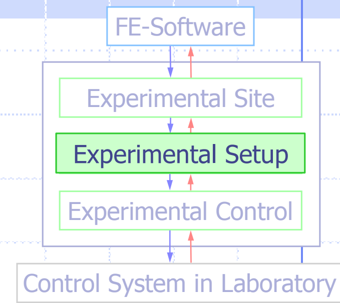
Experimental Setups

4) ESThreeActuators

```
expSetup ThreeActuators $tag $ctrlTag $nlGeomFlag
         $La0 $La1 $La2 $L0 $L1
         <-dspCtrlFact $dspCF> ...
```

`$tag`	unique setup tag
`$ctrlTag`	tag of previously defined control object
`$nlGeomFlag`	nonlinear geometry flag
`$La0`	length of actuator 0
`$La1`	length of actuator 1
`$La2`	length of actuator 2
`$L0`	length of rigid link 0
`$L1`	length of rigid link 1



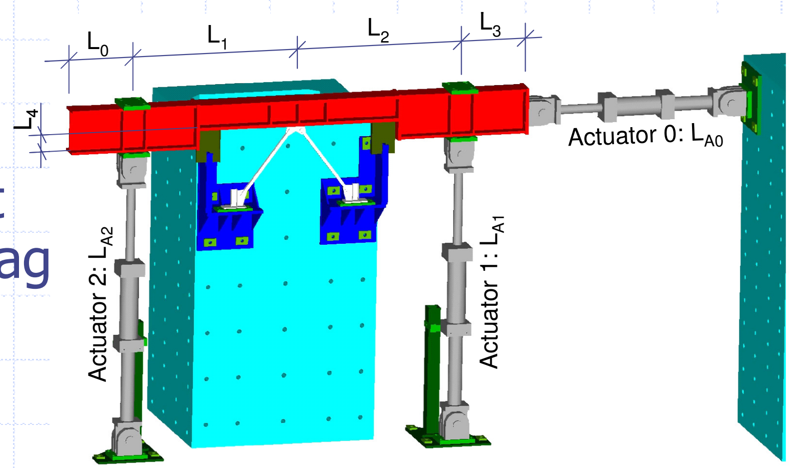


Experimental Setups

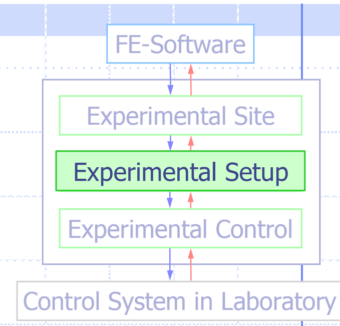
5) ESChevronBraceJntOff

```
expSetup ChevronBraceJntOff $tag $ctrlTag
         $nlGeomFlag $La0 $La1 $La2 $L0 $L1 $L2 $L3
         $L4 <-dspCtrlFact $dspCF> ...
```

$tag	unique setup tag
$ctrlTag	tag of previously defined control object
$nlGeomFlag	nonlinear geometry flag
$La0	length of actuator 0
$La1	length of actuator 1
$La2	length of actuator 2
$L0	length of rigid link 0
$L1	length of rigid link 1
$L2	length of rigid link 2
$L3	length of rigid link 3
$L4	length of rigid link 4



Adding Experimental Setups



Public Member Functions

Constructor and Destructor

```
ExperimentalSetup(int tag, ExperimentalControl& theControl);
ExperimentalSetup(const ExperimentalSetup& es);
virtual ~ExperimentalSetup();
```

Methods dealing with data sizes

```
void setElmtDataSize(int s);
int getElmtDataSize();
void setCtrlDataSize(int s);
int getCtrlDataSize();
void setDaqDataSize(int s);
int getDaqDataSize();
```

int propose( )

Methods dealing with execution and acquisition

```
virtual int setup() = 0;
virtual int propose(const Vector& dsp, const Vector& vel, const Vector& acc) = 0;
virtual int execute() = 0;
virtual int commitState() = 0;
virtual int acquire() = 0;
```

Methods to obtain the response

```
const Vector& getDisp();
const Vector& getVel();
const Vector& getAccel();
const Vector& getForce();
```

int acquire( )

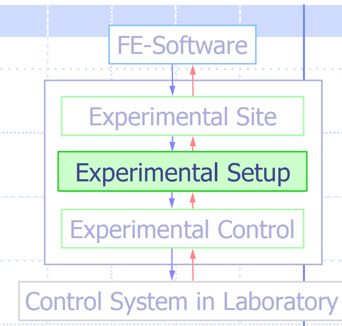
Methods to set the control and data acquisition factors

```
void setDspCtrlFactor(const Vector& f);
void setVelCtrlFactor(const Vector& f);
void setAccCtrlFactor(const Vector& f);
void setDspDaqFactor(const Vector& f);
void setFrcDaqFactor(const Vector& f);
```

Method to get a copy

```
virtual ExperimentalSetup *getCopy (void) = 0;
```

Adding Experimental Setups




```
int ESTwoActuators2d::propose(const Vector& dsp, const Vector& vel,
    const Vector& acc)
{
    // linear geometry
    if (nlFlag == 0)  {
        // actuator 0
        (*dspCtrl)(0) = (*dspCtrlFact)(0) * (-dsp(1));
        (*velCtrl)(0) = (*velCtrlFact)(0) * (-vel(1));
        (*accCtrl)(0) = (*accCtrlFact)(0) * (-acc(1));

        // actuator 1
        (*dspCtrl)(1) = (*dspCtrlFact)(1) * (-dsp(1) - L * dsp(2));
        (*velCtrl)(1) = (*velCtrlFact)(1) * (-vel(1) - L * vel(2));
        (*accCtrl)(1) = (*accCtrlFact)(1) * (-acc(1) - L * acc(2));
    }
    // nonlinear geometry
    else if (nlFlag == 1)  {
        opserr << "ESTwoActuators2d::propose() - nonlinear geometry not implemented yet";
        return ExperimentalReturnType_failed;
    }

    return ExperimentalReturnType_ready;
}
```

```
int ESTwoActuators2d::acquire()
{
    theControl->acquire(dspDaq, frcDaq);

    // linear geometry
    if (nlFlag == 0)  {
        // measured displacements
        (*dspElmt)(0) = 0;
        (*dspElmt)(1) = - (*dspDaqFact)(0) * (*dspDaq)(0);
        (*dspElmt)(2) = 1.0/L * ((*dspDaqFact)(0) * (*dspDaq)(0) - (*dspDaqFact)(1) * (*dspDaq)(1));

        // measured forces
        (*frcElmt)(0) = 0;
        (*frcElmt)(1) = - (*frcDaqFact)(0) * (*frcDaq)(0) - (*frcDaqFact)(1) * (*frcDaq)(1);
        (*frcElmt)(2) = - L * (*frcDaqFact)(1) * (*frcDaq)(1);
    }
    // nonlinear geometry
    else if (nlFlag == 1)  {
        opserr << "ESTwoActuators2d::acquire() - nonlinear geometry not implemented yet";
        return ExperimentalReturnType_failed;
    }

    return ExperimentalReturnType_completed;
}
```

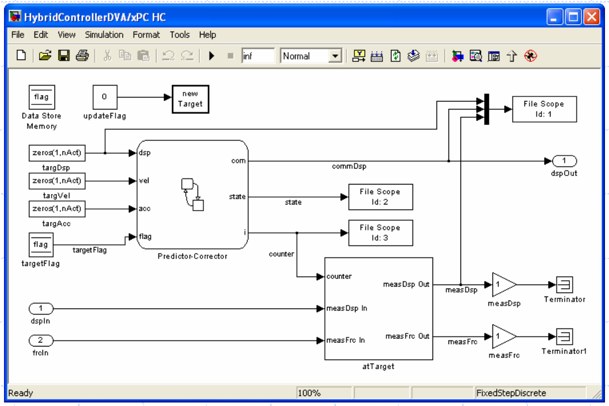
FE-Software
Experimental Site
Experimental Setup
Experimental Control
Control System in Laboratory

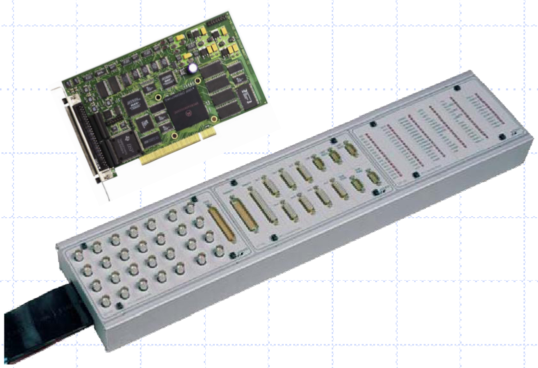
Experimental Controls

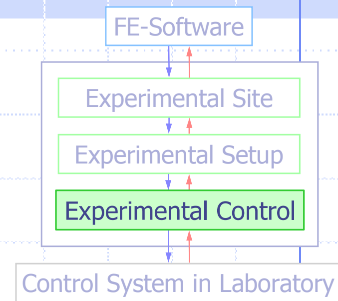
1) ECdSpace

```
expControl dSpace $tag $numSetups "type"
              "boardName"
```

`$tag`	unique control tag
`$numSetups`	number of setups
`"type"`	predictor-corrector type
`"boardName"`	name of dSpace board





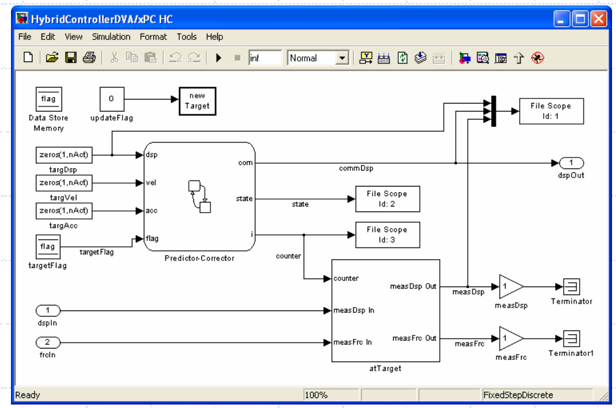


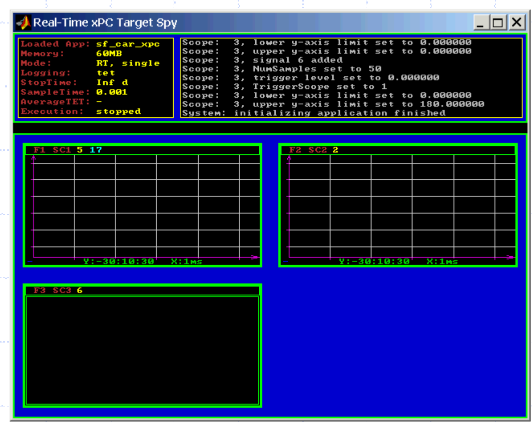
Experimental Controls

2) ECxPCtarget

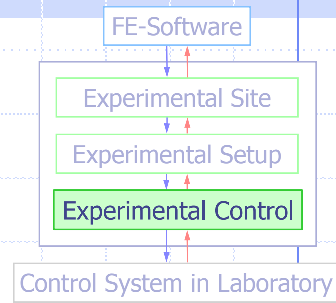
```
expControl xPCtarget $tag $numSetups $type "ipAddr"
                     "ipPort" "appName" "appPath"
```

$tag	unique control tag
$numSetups	number of setups
$type	predictor-corrector type
"ipAddr"	IP address of xPC Target
"ipPort"	IP port of xPC Target
"appName"	name of Simulink application to be loaded
"appPath"	path to Simulink application





Experimental Controls

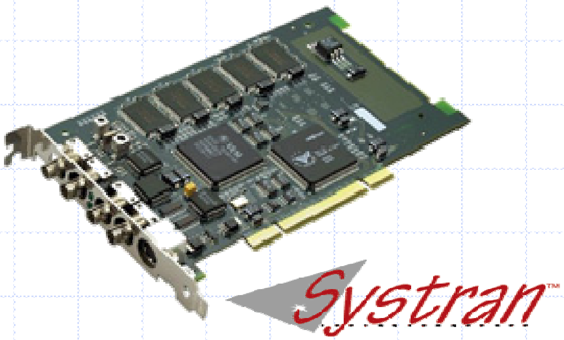


3) ECScramNet

```
expControl ScramNet $tag $numSetups ...
```

`$tag`	unique control tag
`$numSetups`	number of setups

`...`



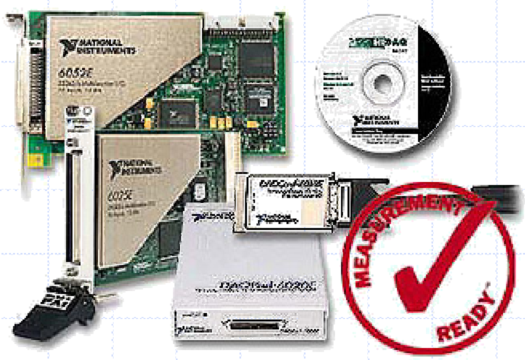
under development

Experimental Controls

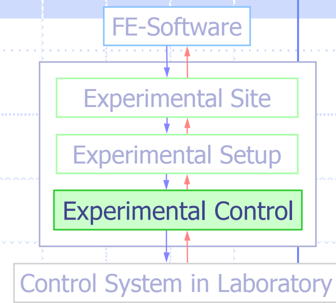
4) ECNIEseries

```
expControl NIEseries $tag $numCtrl $device
```

`$tag`	unique control tag
`$numSetups`	number of setups
`$device`	id of device



Adding Experimental Controls



Public Member Functions

Constructor and Destructor

```
ExperimentalControl(int tag, int nCtrl, int nDaq);
ExperimentalControl(const ExperimentalControl& ec);
virtual ~ExperimentalControl();
```

Methods dealing with data sizes

```
int getCtrlDataSize();
int getDaqDataSize();
```

Methods to set and obtain the responses

```
virtual int setup() = 0;
virtual int execute(const Vector& dsp, const Vector& vel, const Vector& acc) = 0;
virtual int commitState();
virtual int acquire(Vector *dspDaq, Vector *frcDaq) = 0;
```

Method to add a data filter

```
void addFilter(SignalFilter& f);
```

Method to get a copy

```
virtual ExperimentalControl *getCopy (void) = 0;
```

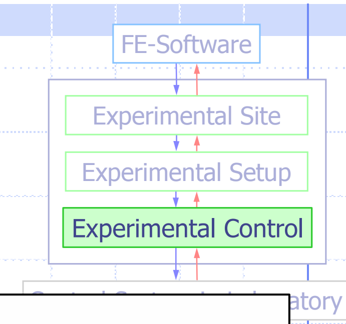
ExperimentalControl( )

int setup( )

int execute( )

int acquire( )

Adding Experimental Controls




```
int ECxPCtarget::execute(const Vector &dsp, const Vector &vel,
    const Vector &acc)
{
    errCode = 0;

    for (int i=0; i<ndimCtrl(sNum); i++)  {
        targDsp[offset+i] = dsp(i);
        targVel[offset+i] = vel(i);
        targAcc[offset+i] = acc(i);

        if (theFilter != 0)  {
            targDsp[offset+i] = theFilter->filtering(targDsp[offset+i]);
        }
    }

    if (sNum<(numSetups-1))  {
        offset += ndimCtrl(sNum);
        ++sNum;
    } else  {
        // send targDsp, targVel and targAcc and set updateFlag
        xPCSetParam(port, targDspId, targDsp);
        if (xPCGetLastError())  {
            xPCErrorMsg(xPCGetLastError(), errMsg);
            opserr << "ECxPCtarget::execute() - xPCSetParam: error = " << errMsg << endl
            xPCClosePort(port);
            xPCFreeAPI();
            return xPCGetLastError();
        }
        if (pcType==2 || pcType==3)  {
            xPCSetParam(port, targVelId, targVel);
            if (xPCGetLastError())  {
                xPCErrorMsg(xPCGetLastError(), errMsg);
                opserr << "ECxPCtarget::execute() - xPCSetParam: error = " << errMsg << e
                xPCClosePort(port);
                xPCFreeAPI();
                return xPCGetLastError();
            }
        }
        if (pcType==3)  {
            xPCSetParam(port, targAccId, targAcc);
            if (xPCGetLastError())  {
                xPCErrorMsg(xPCGetLastError(), errMsg);
                opserr << "ECxPCtarget::execute() - xPCSetParam: error = " << errMsg << e
                xPCClosePort(port);
                xPCFreeAPI();
                return xPCGetLastError();
            }
        }
        updateFlag = 1;
        xPCSetParam(port, updateFlagId, &updateFlag);
        if (xPCGetLastError())  {
            xPCErrorMsg(xPCGetLastError(), errMsg);
            opserr << "ECxPCtarget::execute() - xPCSetParam: error = " << errMsg << endl
            xPCClosePort(port);
            xPCFreeAPI();
            return xPCGetLastError();
        }

        offset = 0;
        sNum = 0;
    }

    return errCode;
}
```

```
int ECxPCtarget::acquire(Vector *dspDaq, Vector *frcDaq)
{
    errCode = 0;

    if (sNum==0)  {
        targetFlag = 1;
        while (targetFlag)  {
            // pause for 1msec
            this->sleep(1);

            // get target flag
            targetFlag = xPCGetSignal(port, targetFlagId);
            if (xPCGetLastError())  {
                xPCErrorMsg(xPCGetLastError(), errMsg);
                opserr << "ECxPCtarget::acquire() - xPCGetSignal: error = " << errMsg << endln;
                xPCClosePort(port);
                xPCFreeAPI();
                return xPCGetLastError();
            }
        }

        // read displacements and resisting forces at target
        xPCGetSignals(port, ndimDaqAll, measDspId, measDsp);
        if (xPCGetLastError())  {
            xPCErrorMsg(xPCGetLastError(), errMsg);
            opserr << "ECxPCtarget::acquire() - xPCGetSignal: error = " << errMsg << endln;
            xPCClosePort(port);
            xPCFreeAPI();
            return xPCGetLastError();
        }
        xPCGetSignals(port, ndimDaqAll, measFrcId, measFrc);
        if (xPCGetLastError())  {
            xPCErrorMsg(xPCGetLastError(), errMsg);
            opserr << "ECxPCtarget::acquire() - xPCGetSignal: error = " << errMsg << endln;
            xPCClosePort(port);
            xPCFreeAPI();
            return xPCGetLastError();
        }

        // reset updateFlag
        updateFlag = 0;
        xPCSetParam(port, updateFlagId, &updateFlag);
        if (xPCGetLastError())  {
            xPCErrorMsg(xPCGetLastError(), errMsg);
            opserr << "ECxPCtarget::acquire() - xPCSetParam: error = " << errMsg << endln;
            xPCClosePort(port);
            xPCFreeAPI();
            return xPCGetLastError();
        }
        // pause for 1msec
        this->sleep(1);
    }

    for (int i=0; i<ndimDaq(sNum); i++)  {
        (*dspDaq)(i) = measDsp[offset+i];
        (*frcDaq)(i) = measFrc[offset+i];
    }

    if (sNum<(numSetups-1))  {
        offset += ndimDaq(sNum);
        ++sNum;
    } else  {
        offset = 0;
        sNum = 0;
    }

    return errCode;
}
```

Conclusions

- Environment-independent framework for development and deployment will boost the use of hybrid simulation (on-site and tele-operation)
- Modularity and transparency of the framework permits existing components to be modified and new components to be added without much dependence on other objects.
  - Speed development of refined hybrid simulation procedures

Conclusions

- Large library of hybrid simulation direct integration methods, experimental elements, controller models, and event-driven solution strategies will be available to the user to choose from or adapt.
- Need:
  - User-community input of parameter passage and features
  - User feedback
  - NEESit assistance in streamlining network communications

Thank you!

Development and operation of the *nees@berkeley* Equipment Site is sponsored by NSF George E. Brown Jr. NEES grants.

http://nees.berkeley.edu