Towards Lightweight Formal Development of MPI Applications

Nelson Rosa[1], Alan Wagner[2], Humaira Kamal[2]
[1]Centro de Informática
Universidade Federal de Pernambuco, PE, Brazil

[2]Department of Computer Science
University of British Columbia, BC, Canada

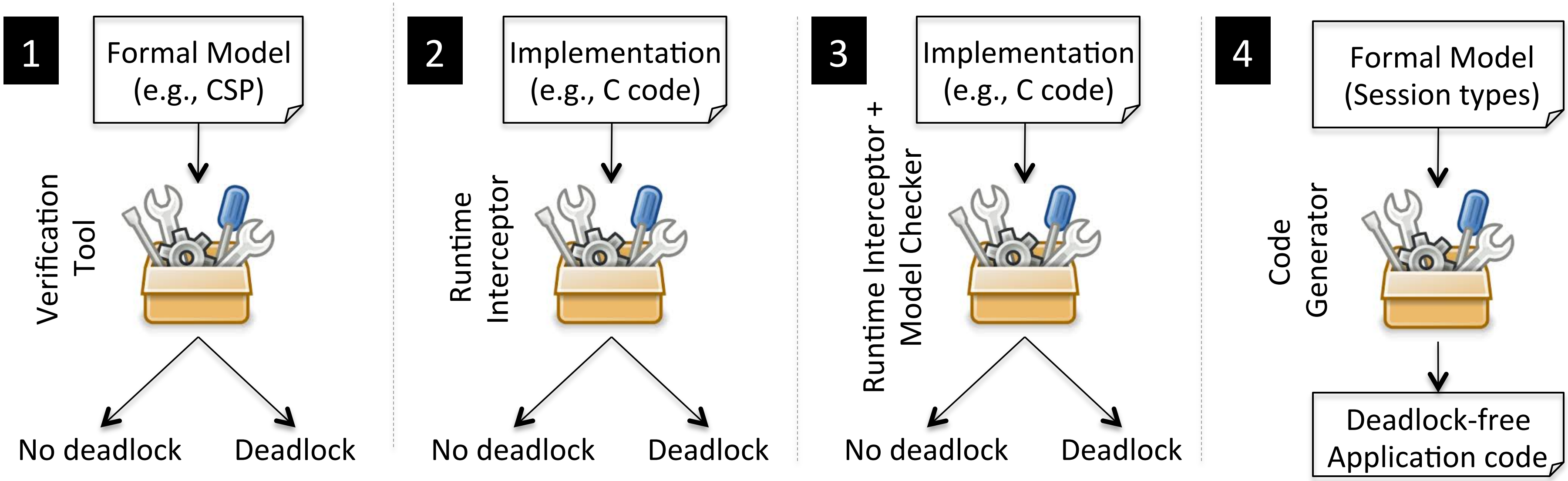
NSERC and ICICS provided support for this work.

Motivation

- Development of safe application in MPI is a complex task
  - MPI imposes few restrictions to enforce correct messaging
  - Incorrect use of arguments results in unmatched messages
  - Potential problems are usually not identified immediately
- Externalizing MPI structure and control
  - Work with FG-MPI using millions of processes
  - Containers: sizing and packing
  - Performance portability, fragility
- Large Scale System Design
  - Nested Collections of Communicating Processes
  - Exposing the structure
  - Distributed locus of control (not a single locus of control)
  - Composable components
  - Robustness and resilience

Motivation

- Several existing approaches focus on the verification of MPI applications
  1. Purely formal
  2. Runtime based
  3. Hybrid
  4. Code Generation



1 Formal Model (e.g., CSP)

Verification Tool

No deadlock Deadlock

2 Implementation (e.g., C code)

Runtime Interceptor

No deadlock Deadlock

3 Implementation (e.g., C code)

Runtime Interceptor + Model Checker

No deadlock Deadlock

4 Formal Model (Session types)

Code Generator

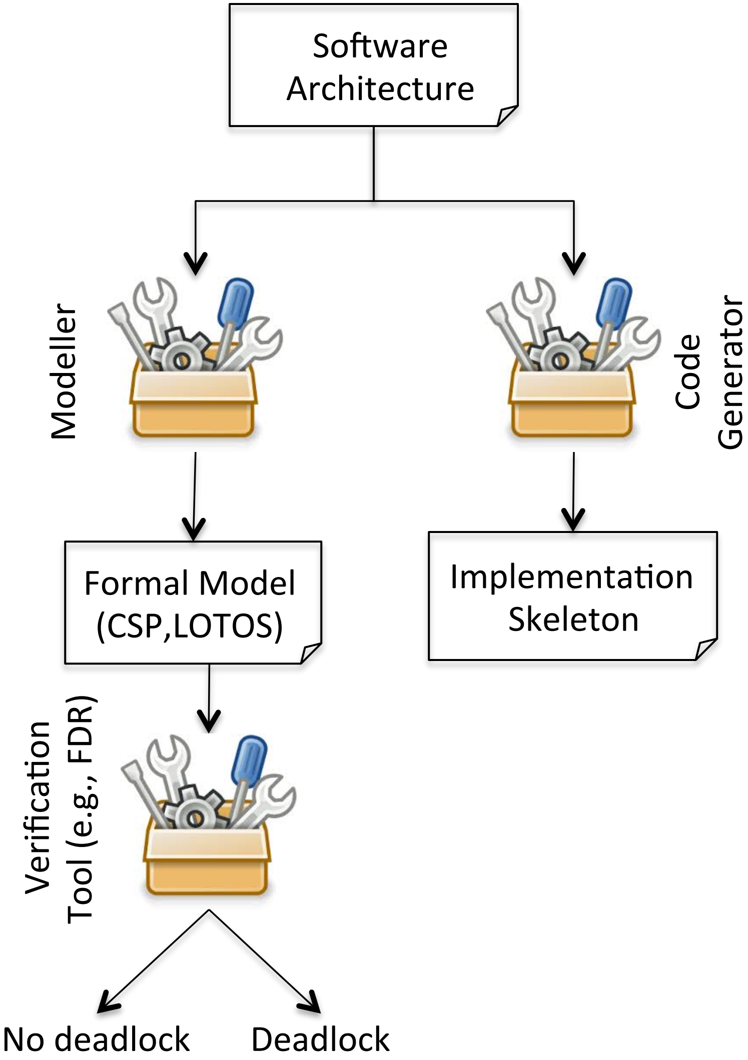
Deadlock-free Application code

Proposed Approach

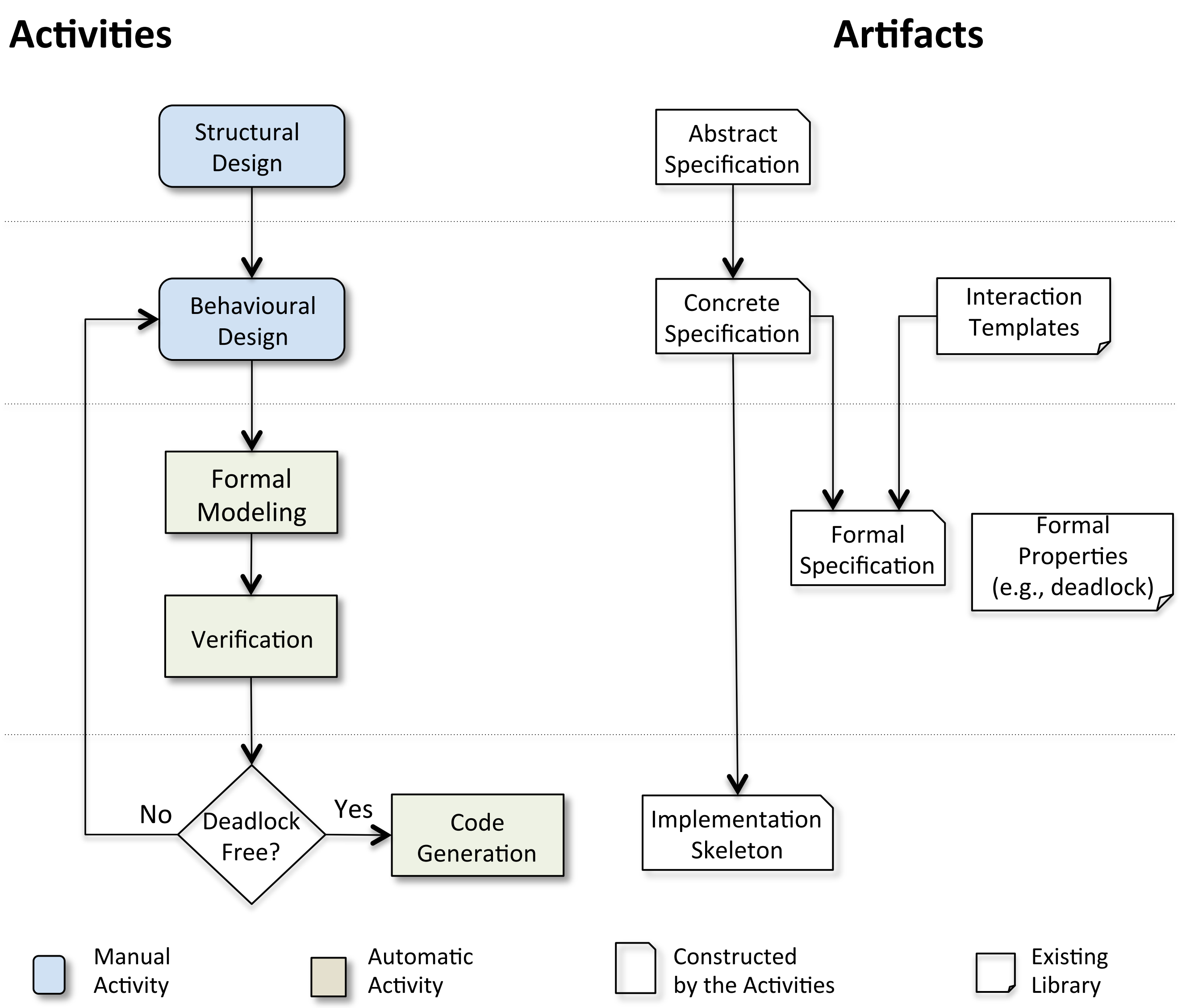
- Lightweight approach that incorporates verification into the early stages of the MPI code development process

- Key characteristics
  - Focus on development time
  - Focus on communication primitives
  - Adoption of software architecture principles
  - Code generation

- Key elements
  - Architecture Description Language
  - Development methodology
  - Formal models
  - Tool support to the proposed methodology



Development Process



Activities

Artifacts

Structural Design

Behavioural Design

Formal Modeling

Verification

Deadlock Free?

No

Yes

Code Generation

Abstract Specification

Concrete Specification

Interaction Templates

Formal Specification

Formal Properties (e.g., deadlock)

Implementation Skeleton

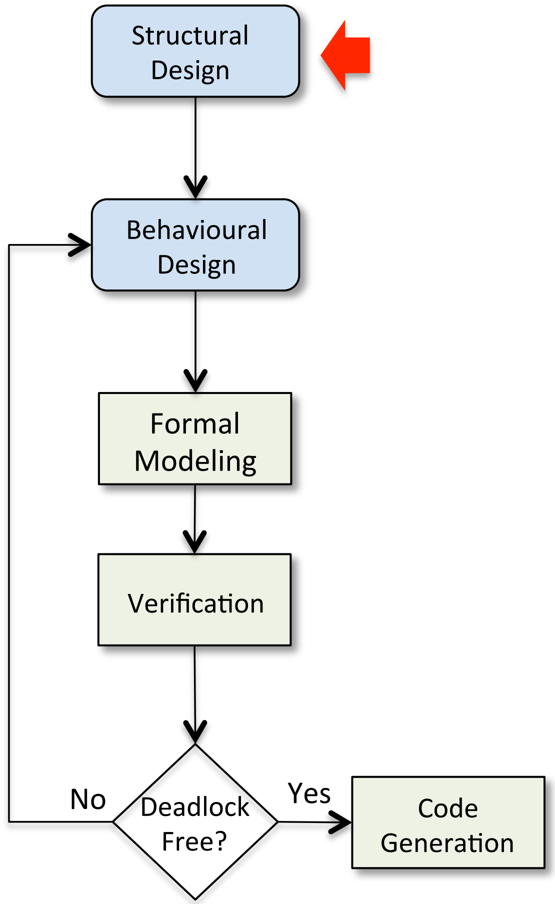
Manual Activity

Automatic Activity

Constructed by the Activities

Existing Library

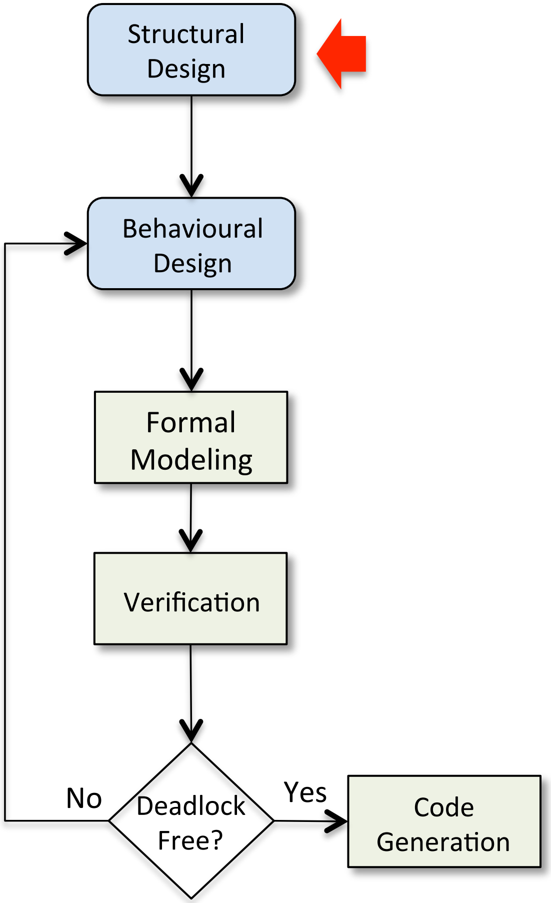
Development Process:: Structural Design

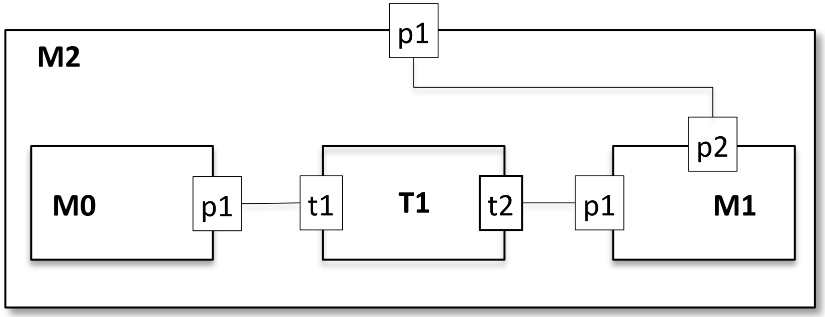


- **Structural Design** decomposes the application into components and connectors
  - Components: computation elements
  - Connectors: communication elements

- ArcMPI (Architecture Description Language)
  - Components can be composed to make up new components
  - There is always a connector between any two interacting components
  - Components and connectors have interaction points (ports) through which they communicate with the external world
  - The description of components and connectors includes their behaviour in addition to their structure.

Development Process:: Structural Design

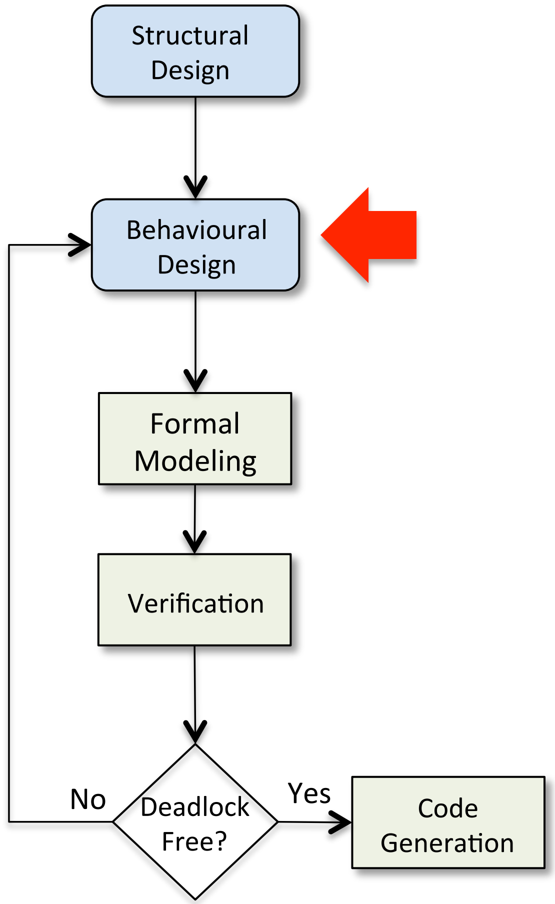




```
component M0{
   port p1;
}

component M1{
    port p1,p2;
}

connector T1{
   port t1,t2;
}
```

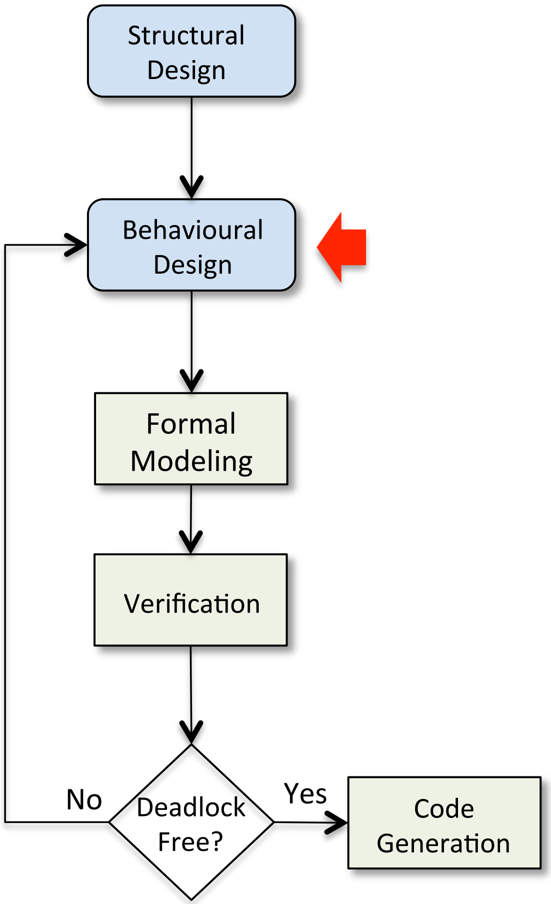
```
component M2{
   port p1;
   components M0, M1;
   connectors T1;
}
implementation{
   M2.p1 = M1.p2;
   M0.p1 to T1.t1;
   M1.p1 to T1.t2;
}
```

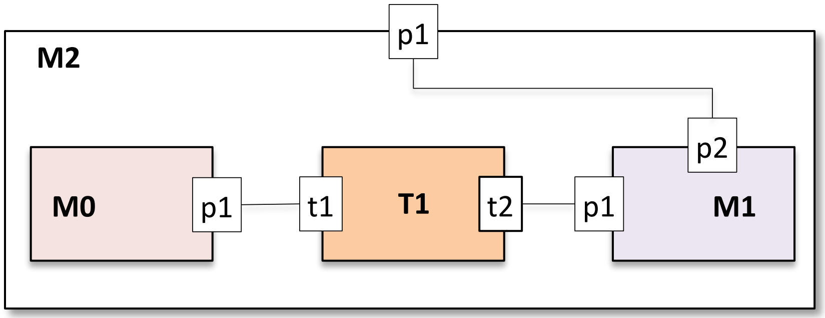
Development Process:: Behavioural Design



- **Behavioural design** enriches component and connector with behavioural descriptions
- Component's behaviour includes "business" actions and invocations to middleware interface
- Connector's behaviour defines how the interaction between components occurs
- Interaction is the pattern of message-flows between components
  - e.g., pipe-filter, send-receive (MPI), request-reply, multicast, publish/subscribe

Development Process:: Behavioural Design



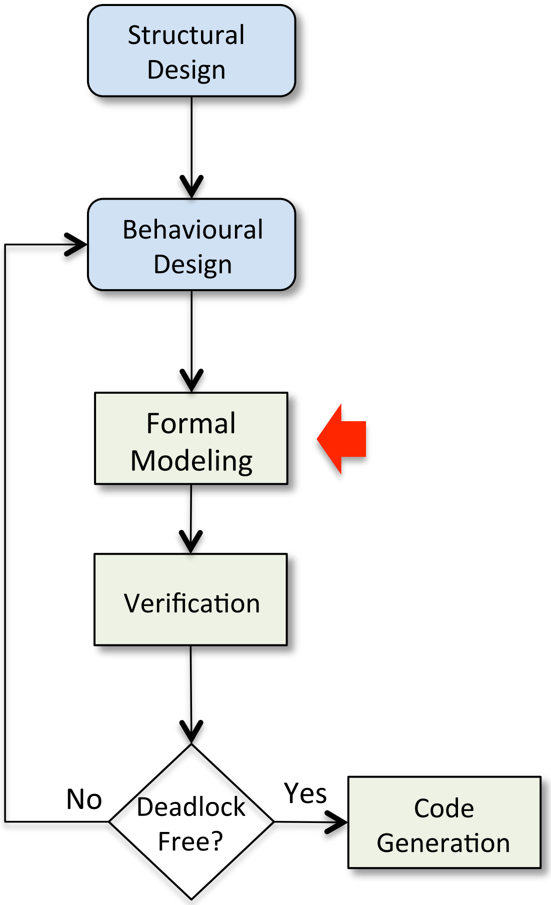


```
uses "mpi.template";
component M0{
   port p1;
   implementation {
         p1.MPI_Send(msg);
         p1.MPI_Recv(msg);
         ...
   }
 }
```

```
uses "mpi.template";
  component M1 {
  port p1,p2;
  implementation {
    p1.MPI_Recv(msg);
    p1.MPI_Send(msg);
    ...
  }
}
```

```
uses "mpi.template";
Connector T1:SendRecvSync {
  port t1,t2;
  implementation {
    t1 = SendRecvSync.t1;
    t2 = SendRecvSync.t2;
  }
}
```

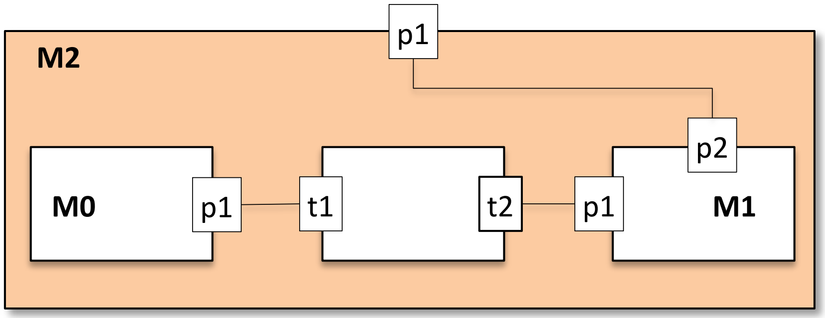
Development Process:: Formal Modelling



- **Formal modelling** defines a mapping from the Concrete Specification (ArcMPI) into a formal technique (e.g., CSP)
- Two steps

  Step 1: To model the structure

  Step 2: To model the behaviour (component and connector)

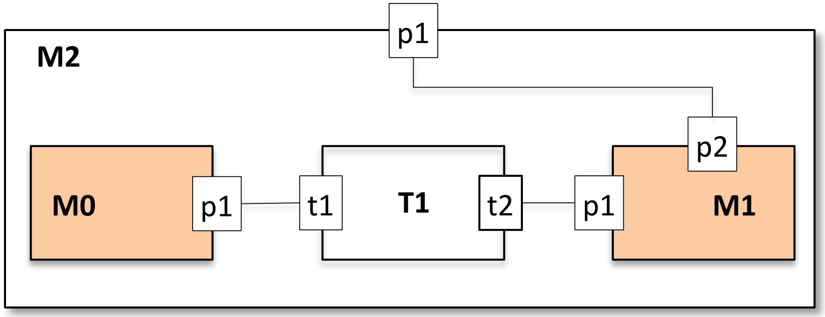


Step 1

T1 = SendRecvSync({0,1})

M2 = (**M0**(0)[[p1 <-t1]] ||| **M1**(1) [[p1<-t2]])
[|{|t1,t2|}|]
T1

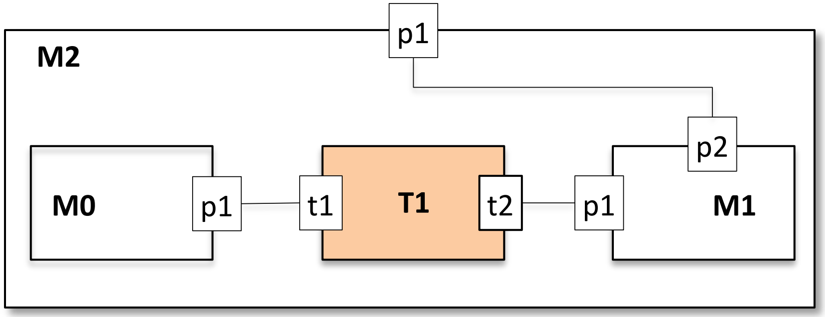
Development Process:: Formal Modelling



Step 2 (Component)

```
          ArcMPI                                             CSP
…
component Mx{
   port pi;
   implementation {
      pi.ai;                             =>          Mx = pi.ai -> pi.aj… -> Mx
      pi.aj;
      …
   }
}
```

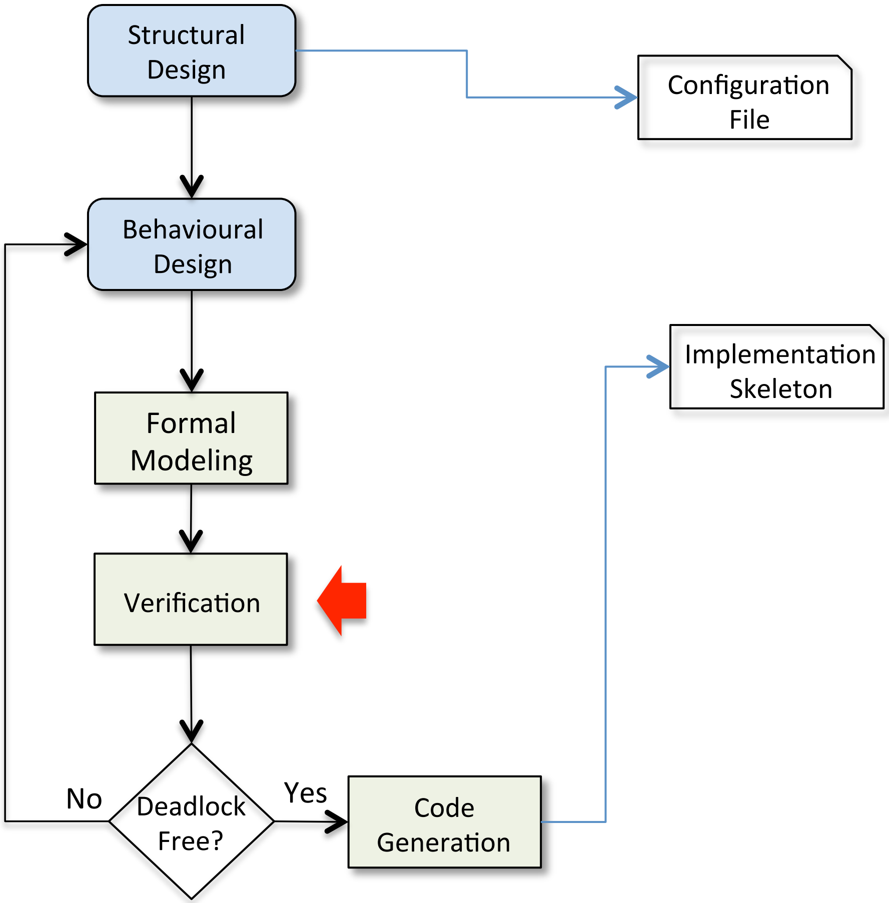
Development Process:: Formal Modelling

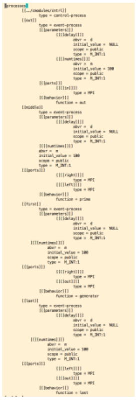


Step 2 (Connector)

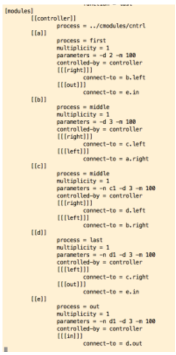
```
SendRecvSync(s) =
    t1.MPI_Send?msg?ct?dt?src:s?dst:s?tag?comm
        -> SendRecvSyncT1’(s,msg,ct,dt,dst,src,tag,comm) []
    t1.MPI_Recv?msg?ct?dt?src:s?dst?tag?comm?sta
        -> SendRecvSyncT1’’(s,msg,ct,dt,dst,src,tag,comm,sta) []
    t2.MPI_Send?msg?ct?dt?src:s?dst?tag?comm
        -> SendRecvSyncT2’(s,msg,ct,dt,dst,src,tag,comm) []
    t2.MPI_Recv?msg?ct?dt?src:s?dst?tag?comm?sta
        -> SendRecvSyncT2’’(s,msg,ct,dt,dst,src,tag,comm,sta)

SendRecvSyncT1'(s,msg,ct,dt,src,dst,tag,comm) =
    t2.MPI_Recv?msg?ct?dt!src?ip2:diff(s,{src})!tag!comm!0
        -> SendRecvSync’’’’(s)

SendRecvSync’’’’(s) =
    (t1!MPI_SUCCESS -> SKIP) ||| (t2!MPI_SUCCESS -> SKIP);
          SendRecvSync(s)

```

Development Process:: Code Generation





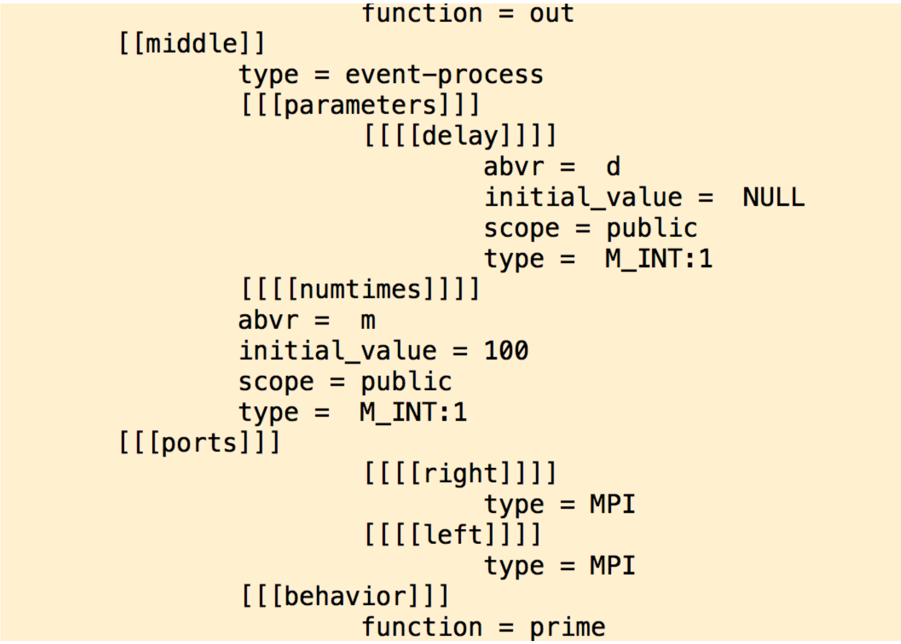
```
[processes]
        [[../cmodules/cntrl]]
                type = control-process
        [[out]]
                type = event-process
                [[[parameters]]]
                        [[[[delay]]]]
                                abvr =  d
                                initial_value =  NULL
                                scope = public
                                type =  M_INT:1
                        [[[[numtimes]]]]
                                abvr =  m
                                initial_value = 100
                                scope = public
                                type =  M_INT:1
                [[[ports]]]
                        [[[[in]]]]
                                type = MPI
                [[[behavior]]]
                        function = out
        [[middle]]
                type = event-process
                [[[parameters]]]
                        [[[[delay]]]]
                                abvr =  d
                                initial_value =  NULL
                                scope = public
                                type =  M_INT:1
                [[[[numtimes]]]]
                abvr =  m
                initial_value = 100
                scope = public
                type =  M_INT:1
        [[[ports]]]
                        [[[[right]]]]
                                type = MPI
                        [[[[left]]]]
                                type = MPI
                [[[behavior]]]
                        function = prime
        [[first]]
                type = event-process
                [[[parameters]]]
                        [[[[delay]]]]
                                abvr =  d
                                initial_value =  NULL
                                scope = public
                                type =  M_INT:1
            [[[[numtimes]]]]
                    abvr =  m
                    initial_value = 100
                    scope = public
                    type =  M_INT:1
        [[[ports]]]
                        [[[[right]]]]
                                type = MPI
                        [[[[out]]]]
                                type = MPI
                [[[behavior]]]
                        function = generator
        [[last]]
                type = event-process
                [[[parameters]]]
                        [[[[delay]]]]
                                abvr =  d
                                initial_value =  NULL
                                scope = public
                                type =  M_INT:1
            [[[[numtimes]]]]
                    abvr =  m
                    initial_value = 100
                    scope = public
                    type =  M_INT:1
        [[[ports]]]
                        [[[[left]]]]
                                type = MPI
                        [[[[out]]]]
                                type = MPI
                [[[behavior]]]
                        function = last
```



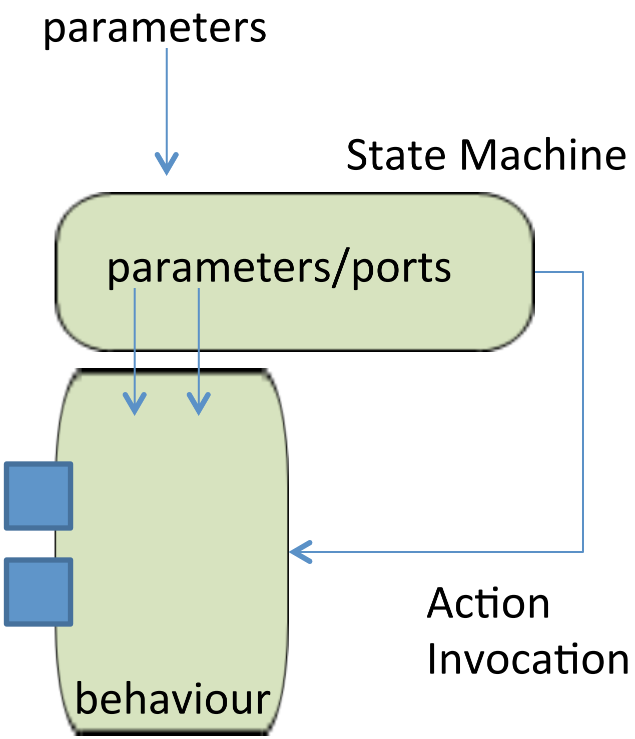
```
[modules]
        [[controller]]
                process = ../cmodules/cntrl
        [[a]]
                process = first
                multiplicity = 1
                parameters = -d 2 -m 100
                controlled-by = controller
                [[[right]]]
                        connect-to = b.left
                [[[out]]]
                        connect-to = e.in
        [[b]]
                process = middle
                multiplicity = 1
                parameters = -d 3 -m 100
                controlled-by = controller
                [[[right]]]
                        connect-to = c.left
                [[[left]]]
                        connect-to = a.right
        [[c]]
                process = middle
                multiplicity = 1
                parameters = -n c1 -d 3 -m 100
                controlled-by = controller
                [[[right]]]
                        connect-to = d.left
                [[[left]]]
                        connect-to = b.right
        [[d]]
                process = last
                multiplicity = 1
                parameters = -n d1 -d 3 -m 100
                controlled-by = controller
                [[[left]]]
                        connect-to = c.right
                [[[out]]]
                        connect-to = e.in
        [[e]]
                process = out
                multiplicity = 1
                parameters = -n d1 -d 3 -m 100
                controlled-by = controller
                [[[in]]]
                        connect-to = d.out
```

```
#include “firstP.h”
PROCESS generator(MPI_Comm comm, PROCESSVARS *pargs, PROCESSPORTS *ports)
{
        int n=0, ack=1;
        MPI_Send(&n,1,MPI_INT,ports->right.dst,0,ports->right.out);
        MPI_Recv(&ack,1,MPI_INT,ports->right.src,0,ports->right.in,&status);
        return 0;
}

#include “middleP.h”
PROCESS prime(MPI_Comm comm, PROCESSVARS *pargs, PROCESSPORTS *ports)
{
        int n=0, ack=1;
        MPI_Recv(&n,1,MPI_INT,ports->left.src,0,ports->left.in,&status);
        MPI_Send(&ack,1,MPI_INT,ports->left.dst,0,ports->left.out);
        MPI_Send(&n,1,MPI_INT,ports->right.dst,0,ports->right.out);
        MPI_Recv(&ack,1,MPI_INT,ports->right.src,0,ports->right.in,&status);
        return 0;
}

PROCESS last(MPI_Comm comm, PROCESSVARS *pargs, PROCESSPORTS *ports)
{
        int n=0, ack=1;
        MPI_Recv(&n,1,MPI_INT,ports->left.src,0,ports->left.in,&status);
        MPI_Send(&ack,1,MPI_INT,ports->left.dst,0,ports->left.out);
}
```

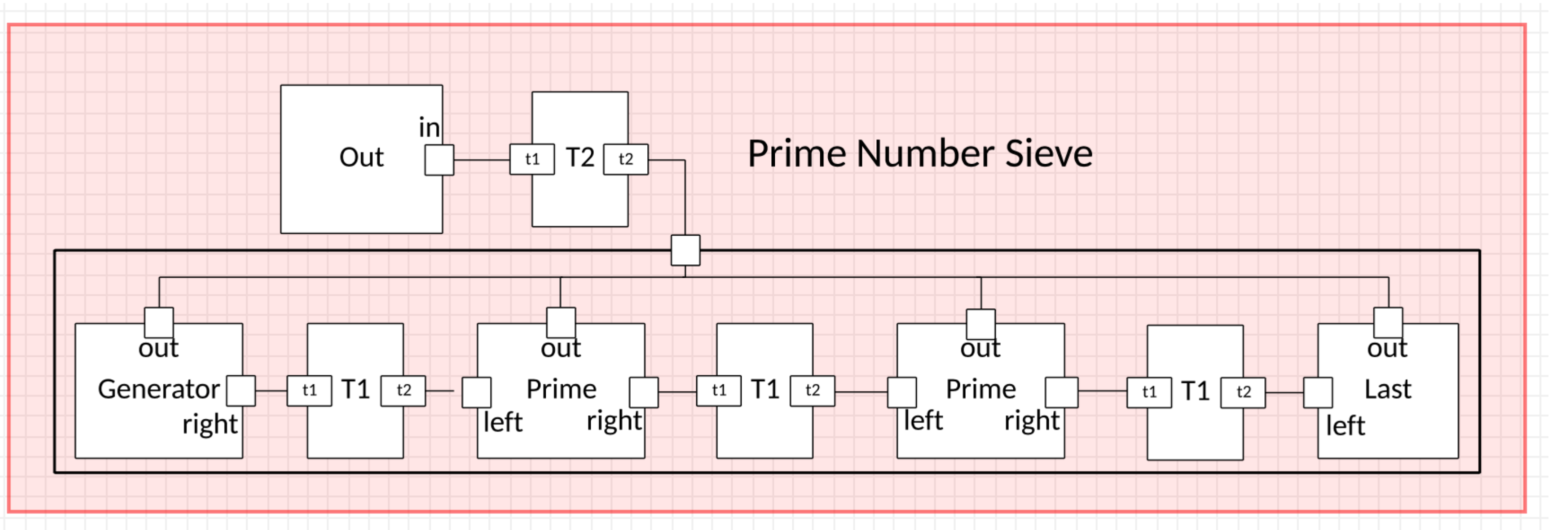
Externalized MPI Processes

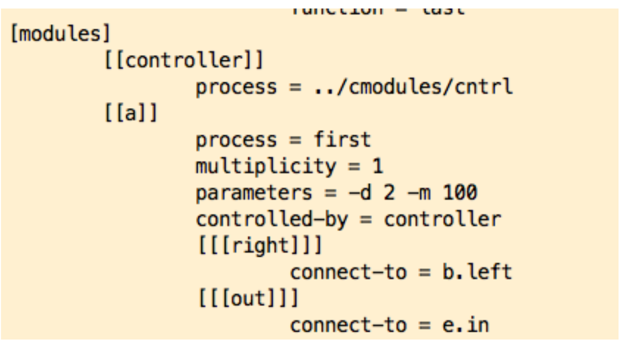


```
                        function = out
        [[middle]]
                type = event-process
                [[[parameters]]]
                        [[[[delay]]]]
                                abvr =  d
                                initial_value =  NULL
                                scope = public
                                type =  M_INT:1
                [[[[numtimes]]]]
                abvr =  m
                initial_value = 100
                scope = public
                type =  M_INT:1
        [[[ports]]]
                        [[[[right]]]]
                                type = MPI
                        [[[[left]]]]
                                type = MPI
                [[[behavior]]]
                        function = prime
```

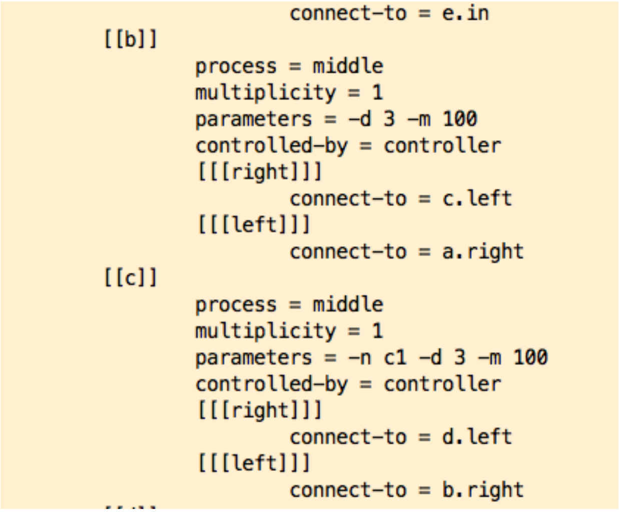


Nested Composition

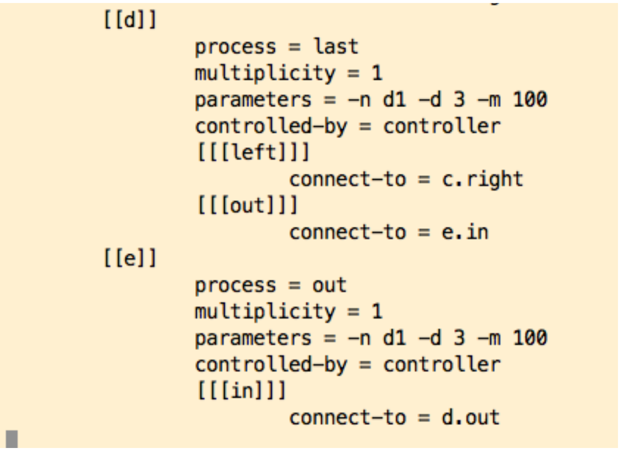




```
                function = last
[modules]
        [[controller]]
                process = ../cmodules/cntrl
        [[a]]
                process = first
                multiplicity = 1
                parameters = -d 2 -m 100
                controlled-by = controller
                [[[right]]]
                        connect-to = b.left
                [[[out]]]
                        connect-to = e.in
```



```
                        connect-to = e.in
        [[b]]
                process = middle
                multiplicity = 1
                parameters = -d 3 -m 100
                controlled-by = controller
                [[[right]]]
                        connect-to = c.left
                [[[left]]]
                        connect-to = a.right
        [[c]]
                process = middle
                multiplicity = 1
                parameters = -n c1 -d 3 -m 100
                controlled-by = controller
                [[[right]]]
                        connect-to = d.left
                [[[left]]]
                        connect-to = b.right
        [[d]]
```

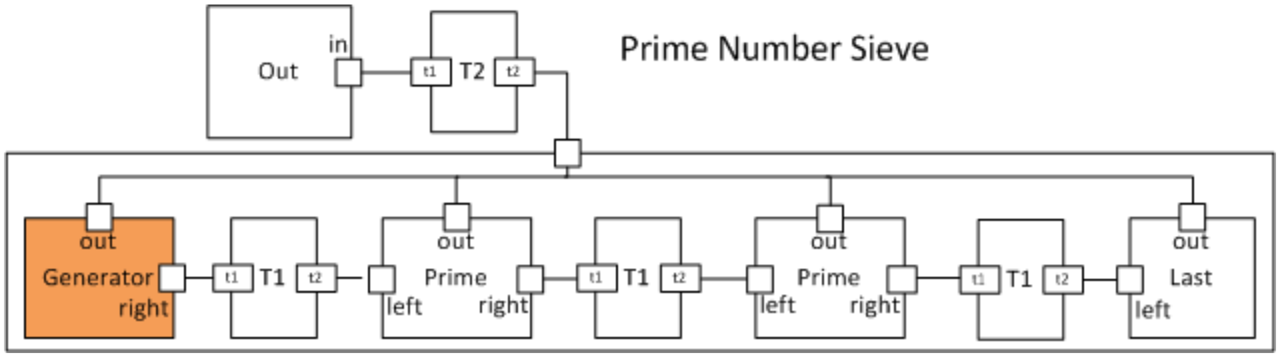


```
        [[d]]
                process = last
                multiplicity = 1
                parameters = -n d1 -d 3 -m 100
                controlled-by = controller
                [[[left]]]
                        connect-to = c.right
                [[[out]]]
                        connect-to = e.in
        [[e]]
                process = out
                multiplicity = 1
                parameters = -n d1 -d 3 -m 100
                controlled-by = controller
                [[[in]]]
                        connect-to = d.out
```

Auto-generated Code

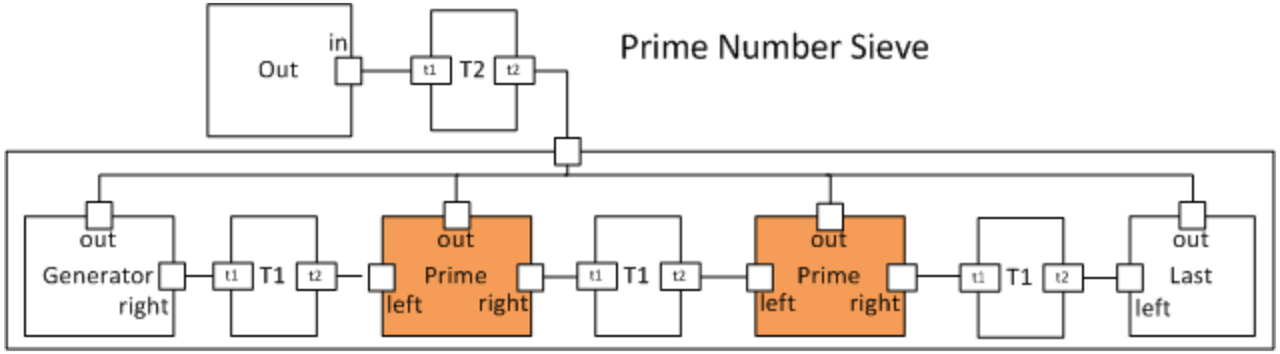
```
#include “firstP.h”
PROCESS generator(MPI_Comm comm, PROCESSVARS *pargs, PROCESSPORTS *ports)
{
     int n=0, ack=1;
     MPI_Send(&n,1,MPI_INT,ports->right.dst,0,ports->right.out);
     MPI_Recv(&ack,1,MPI_INT,ports->right.src,0,ports->right.in,&status);
     return 0;
}

#include “middleP.h”
PROCESS prime(MPI_Comm comm, PROCESSVARS *pargs, PROCESSPORTS *ports)
{
     int n=0, ack=1;
     MPI_Recv(&n,1,MPI_INT,ports->left.src,0,ports->left.in,&status);
     MPI_Send(&ack,1,MPI_INT,ports->left.dst,0,ports->left.out);
     MPI_Send(&n,1,MPI_INT,ports->right.dst,0,ports->right.out);
     MPI_Recv(&ack,1,MPI_INT,ports->right.src,0,ports->right.in,&status);
     return 0;
}

PROCESS last(MPI_Comm comm, PROCESSVARS *pargs, PROCESSPORTS *ports)
{
     int n=0, ack=1;
     MPI_Recv(&n,1,MPI_INT,ports->left.src,0,ports->left.in,&status);
     MPI_Send(&ack,1,MPI_INT,ports->left.dst,0,ports->left.out);
}
```

Development Process:: Code Generation:: Example



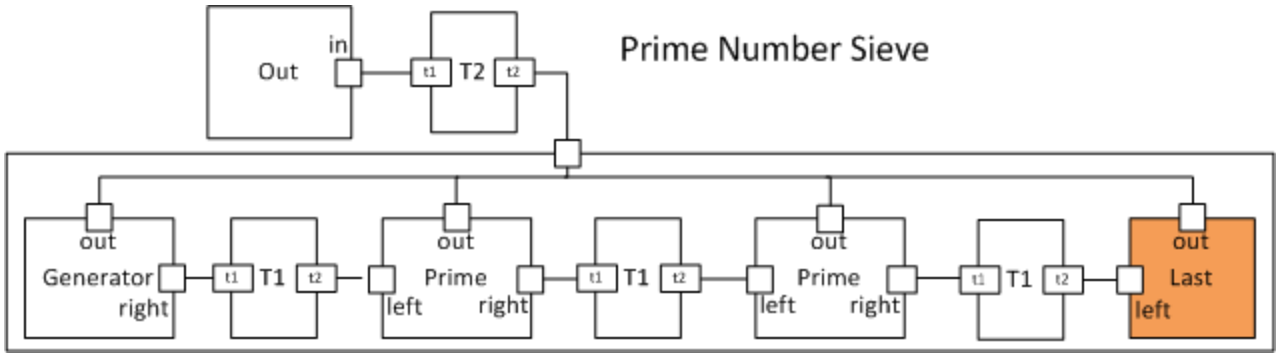
```
#include "firstP.h"
/* generates a sequence of odd numbers */
PROCESS generator(MPI_Comm groupcomm, PROCESSVARS *pargs, PROCESSPORTS *ports) {
  MPI_Status status;
  int myprime = 2, number = 3, stop = FALSE;
  MPI_Send(&myprime,1,MPI_INT,ports->out.src,0,ports->out.in);
  while ( !stop ) {
    MPI_Send(&number,1,MPI_INT,ports->right.dst,0,ports->right.out);
    MPI_Recv(&stop,1,MPI_INT,ports->right.src,0,ports->right.in,&status);
    number = number+2;
  }
  pargs->shutdown = TRUE;
  return 0;
}
```

Development Process:: Code Generation:: Example



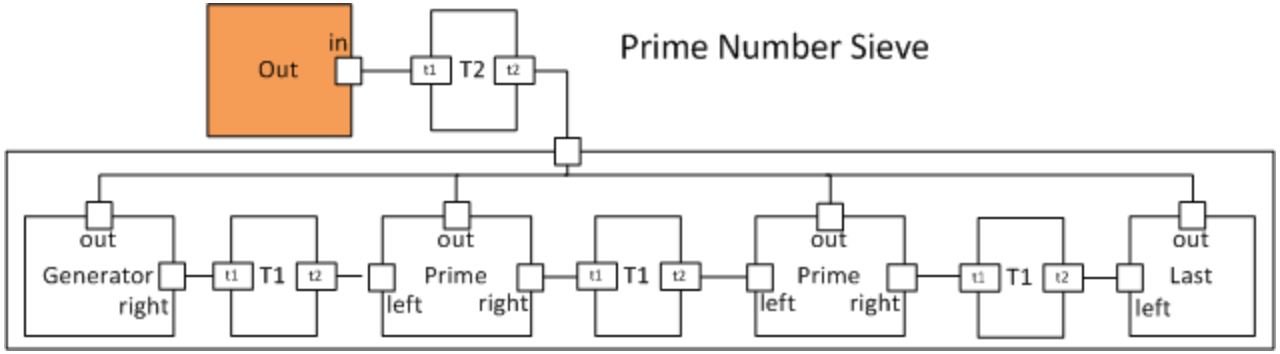
```
#include "middleP.h"
PROCESS prime(MPI_Comm groupcomm, PROCESSVARS *pargs, PROCESSPORTS *ports) {
  MPI_Status status;
  int number, myprime, stop = FALSE;
  MPI_Recv(&myprime,1,MPI_INT,ports->left.src,0,ports->left.in,&status);
  MPI_Send(&stop,1,MPI_INT,ports->left.dst,0,ports->left.out);
  MPI_Send(&myprime,1,MPI_INT,ports->out.src,0,ports->out.in);
  while ( !stop ) {
    MPI_Recv(&number,1,MPI_INT,ports->left.src,0,ports->left.in,&status);
    if ( number % myprime != 0) {
      MPI_Send(&number,1,MPI_INT,ports->right.dst,0,ports->right.out);
      MPI_Recv(&stop,1,MPI_INT,ports->right.src,0,ports->right.in,&status);
    }
    MPI_Send(&stop,1,MPI_INT,ports->left.dst,0,ports->left.out);
  }
  pargs->shutdown = TRUE;
  return 0;
}
```

Development Process:: Code Generation:: Example



```
#include "lastP.h"
PROCESS last(MPI_Comm groupcomm, PROCESSVARS *pargs, PROCESSPORTS *ports) {
  MPI_Status status;
  int myprime, stop = TRUE;
  MPI_Recv(&myprime,1,MPI_INT,ports->left.src,0,ports->left.in,&status);
  MPI_Send(&stop,1,MPI_INT,ports->left.dst,0,ports->left.out);
  MPI_Send(&myprime,1,MPI_INT,ports->out.src,0,ports->out.in);
  pargs->shutdown = TRUE;
  return 0;
}
```

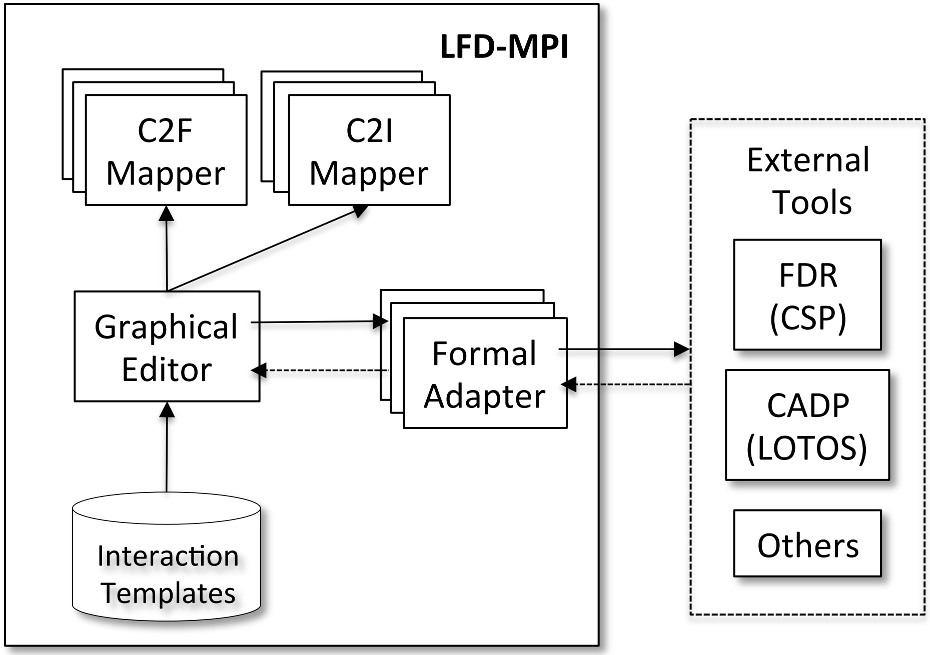
Development Process:: Code Generation:: Example



```
#include "outP.h"
/* output process to receive and print all of the primes */
PROCESS out(MPI_Comm groupcomm, PROCESSVARS *pargs, PROCESSPORTS *ports) {
  int size, myprime=0;
  MPI_Status status;
  MPI_Comm_size(ports->in.out,&size);   // ANY-PORT
  while (--size) {
    MPI_Recv(&myprime,1,MPI_INT,MPI_ANY_SOURCE,0,ports->in.out,&status);
    printf("%d\n",myprime);
  }
  pargs->shutdown = TRUE;
  return 0;
}
```

Implementation

- LFD-MPI
  - Implemented in Java
  - Supported formalisms: LOTOS and CSP
  - Interaction with CADP Toolbox (LOTOS) through files
  - Interaction with FDR3 (CSP) through FDR API
- Elements
  - Graphical Editor (Architectural Design and Behavioural Design)
  - C2F Mapper (Formal Modelling)
  - Formal Adapters (Verification)
  - C2I (Code Generation)
  - Repository of Interaction Templates



Conclusion and Future Work

- Lightweight formal approach to build safety into the MPI applications

- Combination of three key aspects:
  - Rapid prototyping of MPI applications
  - Use of formal model (in CSP and LOTOS) to specify MPI applications
  - Verification of MPI applications that works with partial implementation codes

- Modelling tool shields (as much as possible) the application developer from the formal techniques

- Limitations
  - Support to a subset of MPI point-to-point communication primitives

Conclusion and Future Work

- FG-MPI
  - Compatible with MPI processes
  - November release updated to MPICH 3.1
  - MPICH 3.1
    - InfiniBand
    - Xeon-Phi
    - Hierarchical communication support
- Dynamic Process API (separate from existing MPI one)
- http://www.cs.ubc.ca/~humaira/fgmpi.html