Brandenburg Technical University Cottbus

Chair of Computer Networks and Communication Systems

Brandenburgische Technische Universität Cottbus

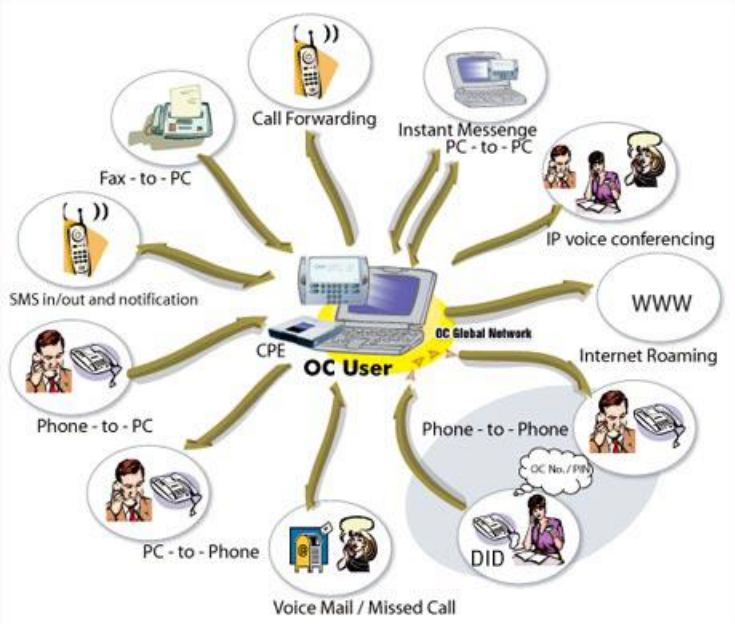
On the Formalization of UML Activities for Component-Based Protocol Design Specifications

Prabhu Shankar Kaliappan, Hartmut König

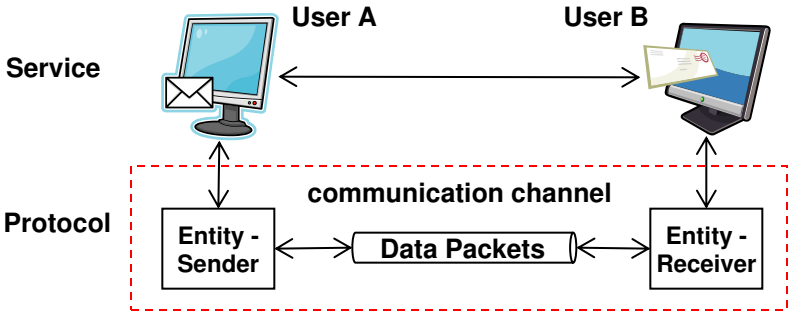
SOFSEM 2012

23 Jan 2012

Introduction - Work Focus (1)



- Communication protocol

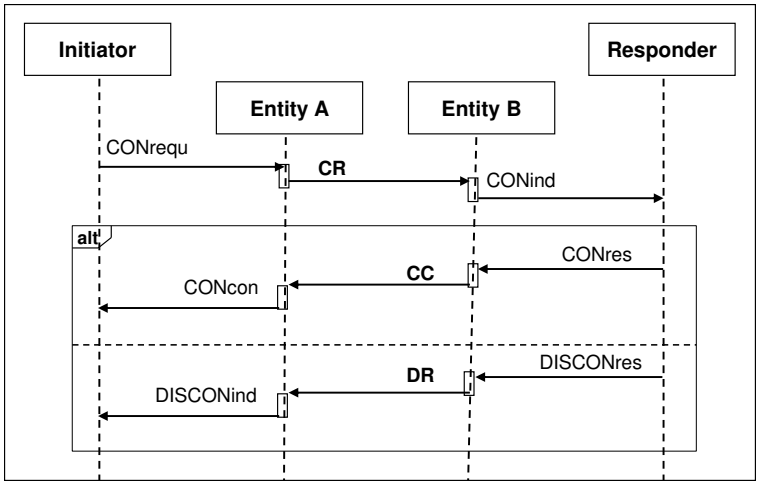


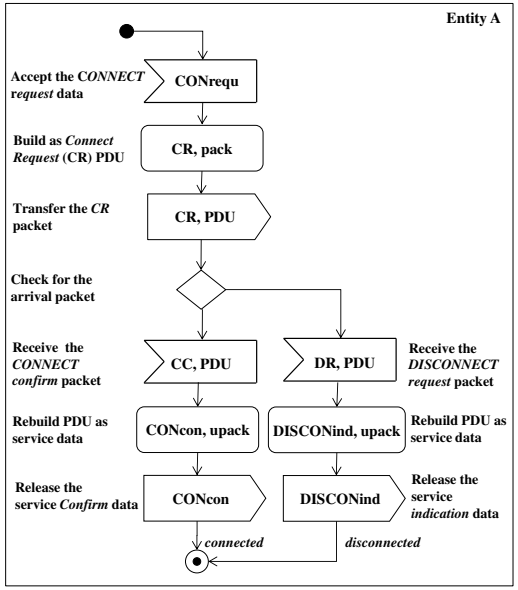
Protocol functions	Mechanisms
Connection establishment	Explicit connection establishment
	Implicit connection establishment
Data Transfer	Stream-based data transfer
	Datagram-based data transfer
Error Control	Go back N
	Selected repeat
…..	…..

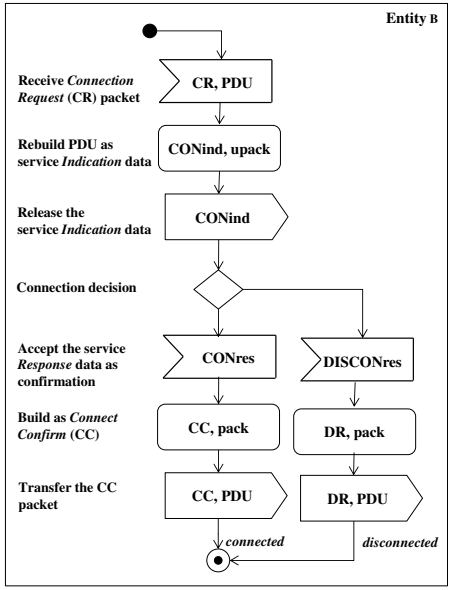
Introduction - Work Focus (2)

- Model-driven approach*
  - UML sequence diagram (message communication)
  - UML activity diagram (entity behavior)

- Component-based protocol design
  - Reusability





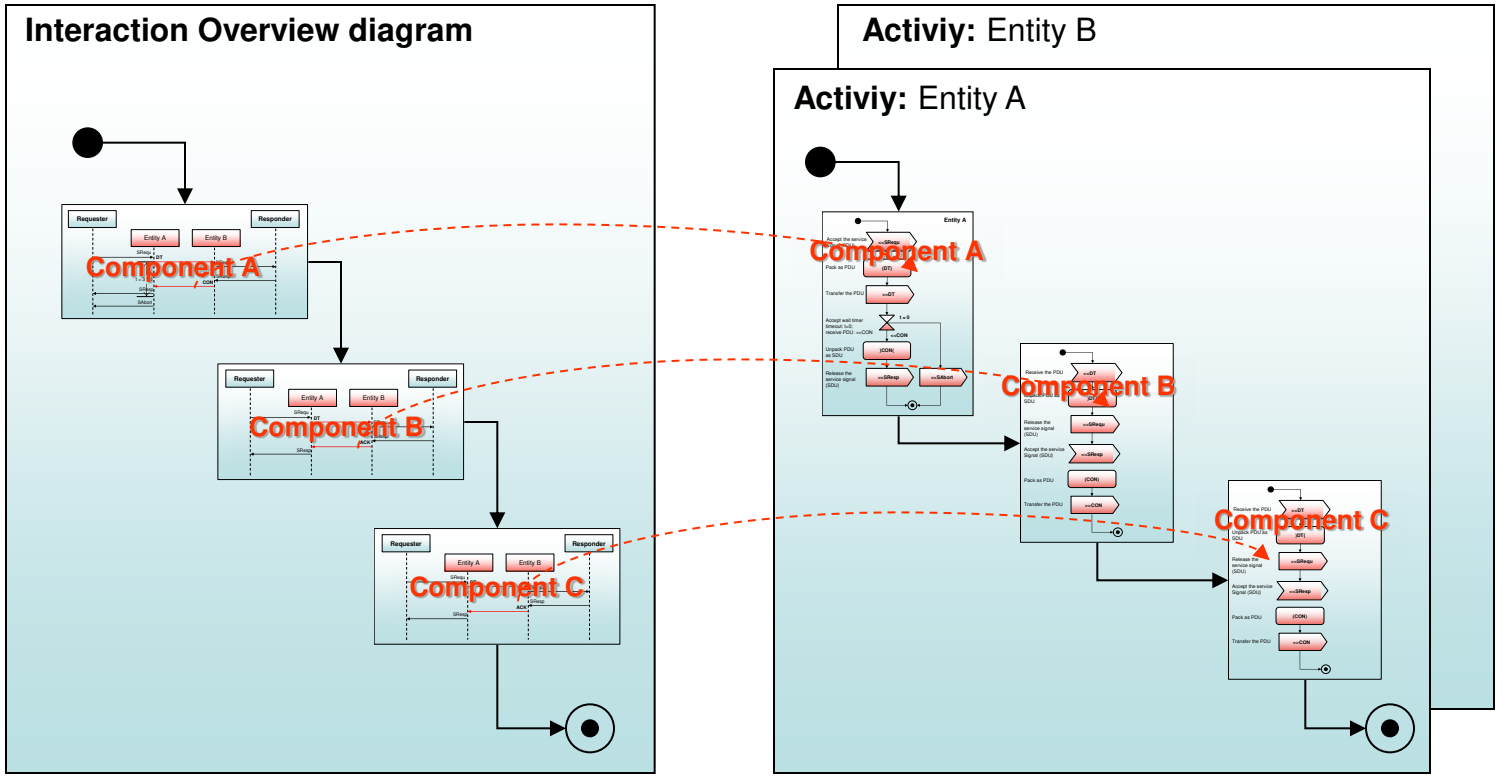


An example - Connection establishment

* Kaliappan, P.S.; König, H.; Schmerl, S.: Model-Driven Protocol Design Based on Component Oriented Modeling. In: *Proc. of the 12th International Conference on Formal Engineering Methods*, Shanghai. LNCS 6447, Springer, pp. 613-629, 2010

Introduction - Work Focus (3)

- Presentation interface

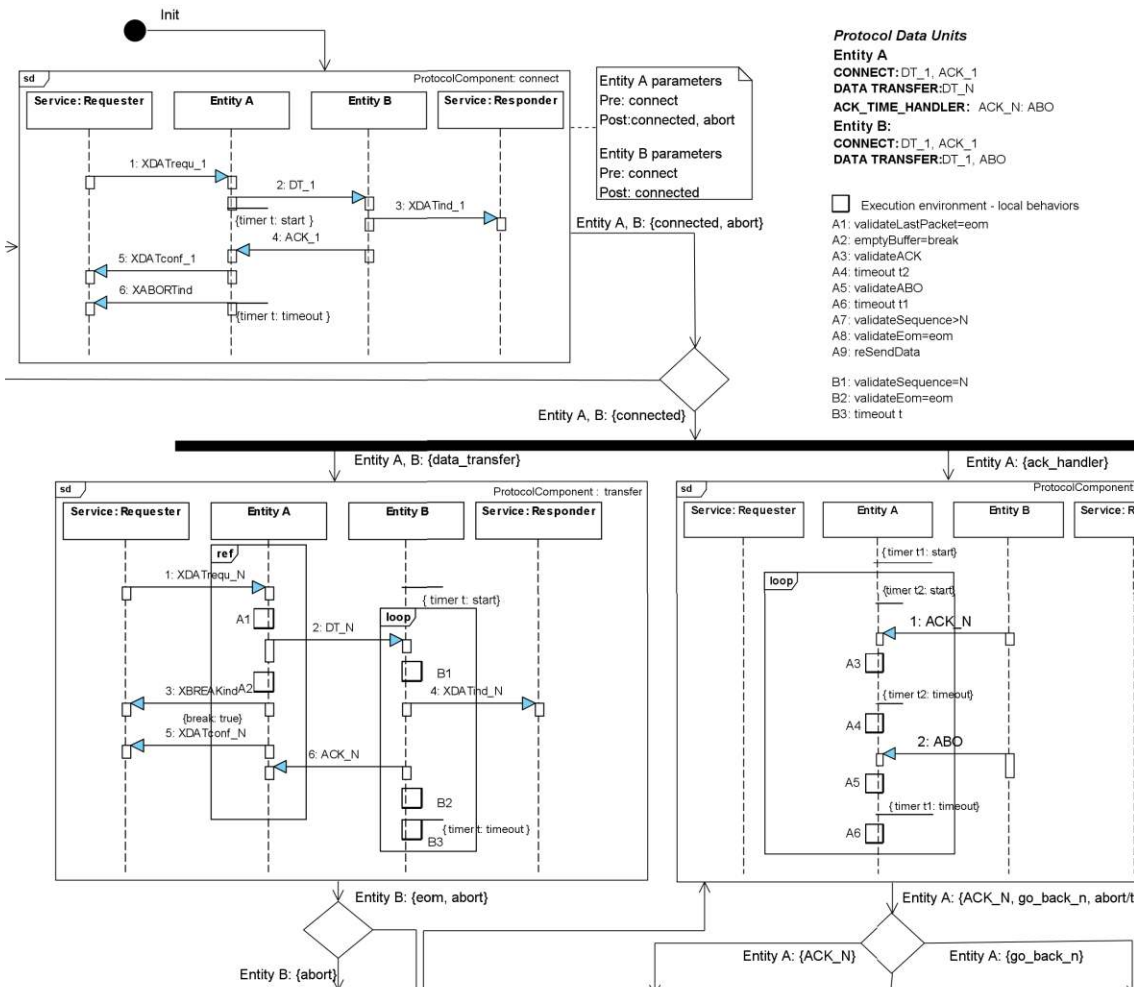


Communication perspective

Behavior perspective

* Kaliappan, P.S.; König, H.: An Approach to Synchronize UML-Based Design Components for Model-Driven Protocol Development. In: *34th Annual IEEE Software Engineering Workshop*, Limerick, Ireland, IEEE, 2011

eXample Data Transfer (XDT) Protocol Specification



Protocol Data Units

Entity A
CONNECT: DT_1, ACK_1
DATA TRANSFER:DT_N
ACK_TIME_HANDLER: ACK_N: ABO

Entity B:
CONNECT: DT_1, ACK_1
DATA TRANSFER:DT_1, ABO

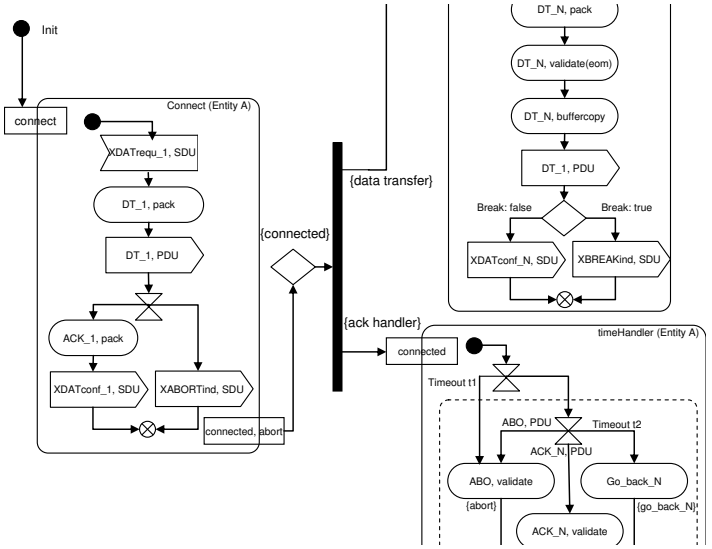
Execution environment - local behaviors

A1: validateLastPacket=eom
A2: emptyBuffer=break
A3: validateACK
A4: timeout t2
A5: validateABO
A6: timeout t1
A7: validateSequence>N
A8: validateEom=eom
A9: reSendData

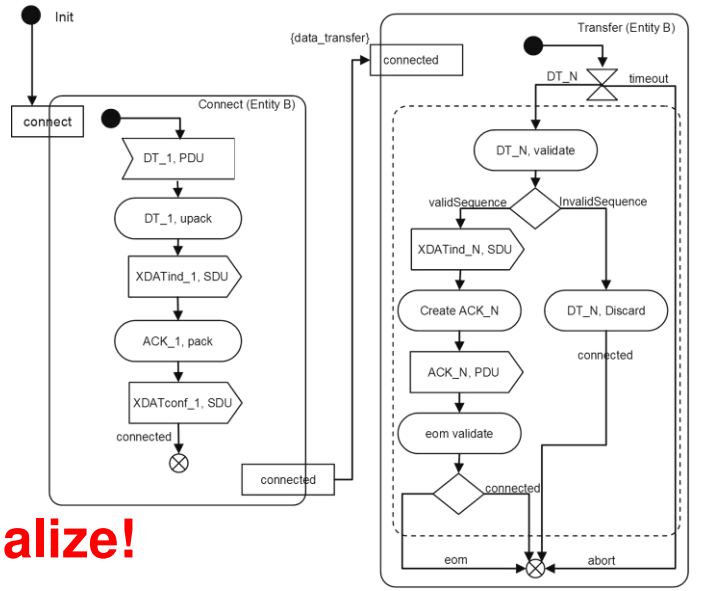
B1: validateSequence=N
B2: validateEom=eom
B3: timeout t

(Communication perspective)

Entity A

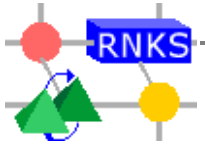


Entity B



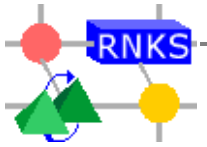
How to Formalize!

(Behavior perspective)



Outline of the talk

1. Motivation
2. Related Work
3. cTLA – compositional Temporal Logic of Actions
4. Formalizing the Semantics of UML Activity Diagrams
5. Final Remarks



1. Motivation

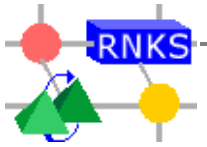
- UML superstructure document
  - 14 diagrams
  - Classified according to the structure and behavior viewpoints
    - → UML Use case diagram, collabration diagram, etc. – Structure(s)
    - → UML Activity diagram, State charts, etc. – Behavior(s)

- Syntax
  - To visualize the design models

Send signal action

- Semantics
  - A way to interpret the design models – Formally!

```
Send(Data: Value)
 {
   ........
 }
```



1. Motivation (2)

- UML sequence diagrams
  - Front end design – Design aid
  - Execution specification is unidentified
- UML activity diagrams
  - Preferred for further development
    - → Enable Verification
    - → Enable Code generation
- Formalizing activity diagrams
- Possible approaches
  - Object Constraint Language, Z, B, Communicating Sequential Processes, Petri Nets, Prototype Verification System, Logics, C++, JAVA, etc.

2. Related Work

- Object Constraint Language
  - Textual constraint
- Communicating Sequential Processes
  - Process Algebra
- Prototype Verification System (PVS)
  - High order logic
- Temporal Logic
  - Process Algebra + High order Logic
  - For e.g., Temporal Logic of Actions (TLA), compositional Temporal Logic of Actions (cTLA)
- Programming Language – C++, JAVA
  - Language definitions

ActivityA.Send: DT,
ActivityB.Receive:DT

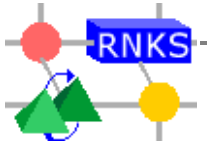
Send signal action

ActivityA.Send=DT → ActivityB.Receive→final

```
Send[DT: Data]: THEORY
BEGIN
  send: [ DT →ActivityB?]
  .....
END
```

[ ]ActivityA.Send_DT→ .....

```
Send(DT: Data)
{
  ........
}
```



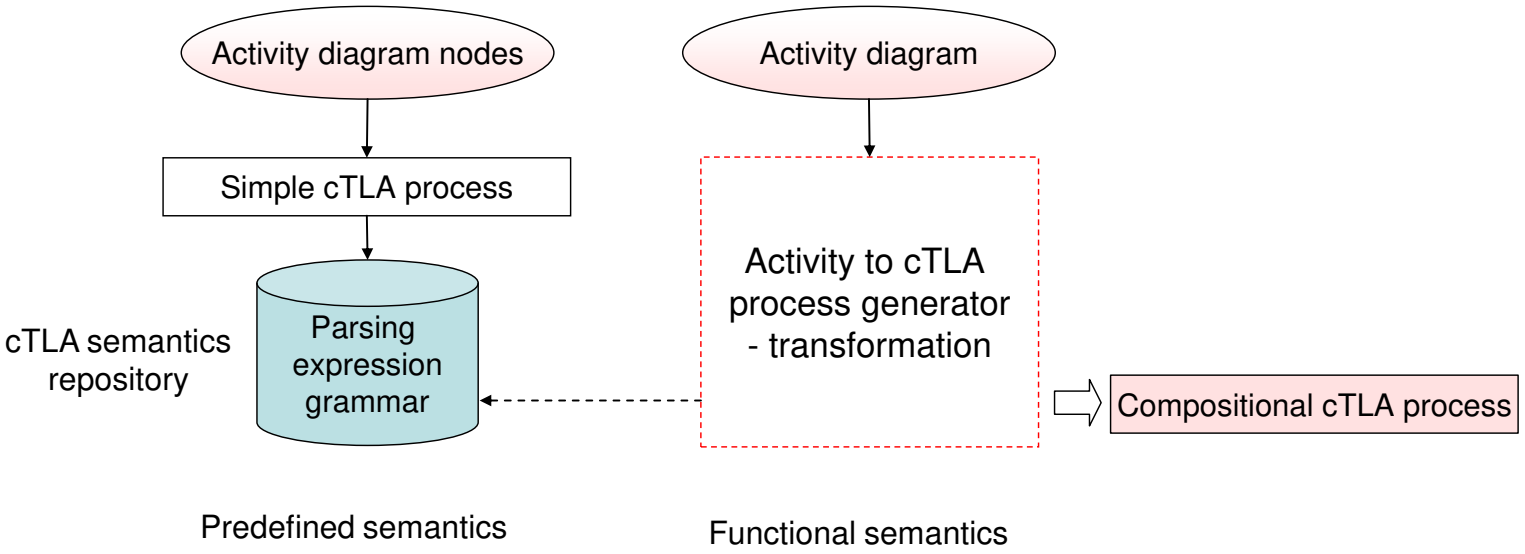
3. cTLA – compositional Temporal Logic of Actions

- Specification language for modeling transfer protocols *
  - Simplified version of L. Lamport'sTemporal Logic of Actions (TLA)
- Simple cTLA processes
  - To model single resource in a system
- Compositional cTLA process
  - To model multiple resources in a system

Simple cTLA Process	Compositional cTLA Process
PROCESS communicate(pdu_type: ANY) **CONSTANTS** FREE ∈ pdu_type **BODY** **VARIABLES** channel: pdu_type **INIT** ≙ channel = FREE **ACTIONS** send(sd: pdu_type) ≙ channel = FREE ∧ channel′ = sd; receive(rd: pdu_type) ≙ channel ≠ FREE ∧ channel = rd ∧ channel′ = FREE; **END**	**PROCESS** connect_s **IMPORT** DT-PDU **BODY** **VARIABLES** state: {"idle","wait connection","connected"}; **INIT** ≙ state = "idle"; **PROCESSES** C: communicate(pdu: pdu_type); t: Timer(to: natural); … **ACTIONS** Con-Init(pdu: pdu_type) ≙ pdu ∧ pdu.type = "DT" ∧ pdu.sequ = 1 ∧ state = "idle" ∧ state′= "wait connection" ∧ C.send(pdu) ∧ t.start(5); … **END**

* Herrmann, P.; Krumm, H.: A Framework for Modeling Transfer Protocols. In: Computer Networks 34, 2000, pp. 317-337

4. Formalizing the Semantics of UML-based Specifications



- Predefined semantics
  - Simple cTLA process to define frequently used UML activity nodes
- Functional semantics
  - Compositional cTLA process to define an entire UML activity diagram

Simple cTLA process

Initial node

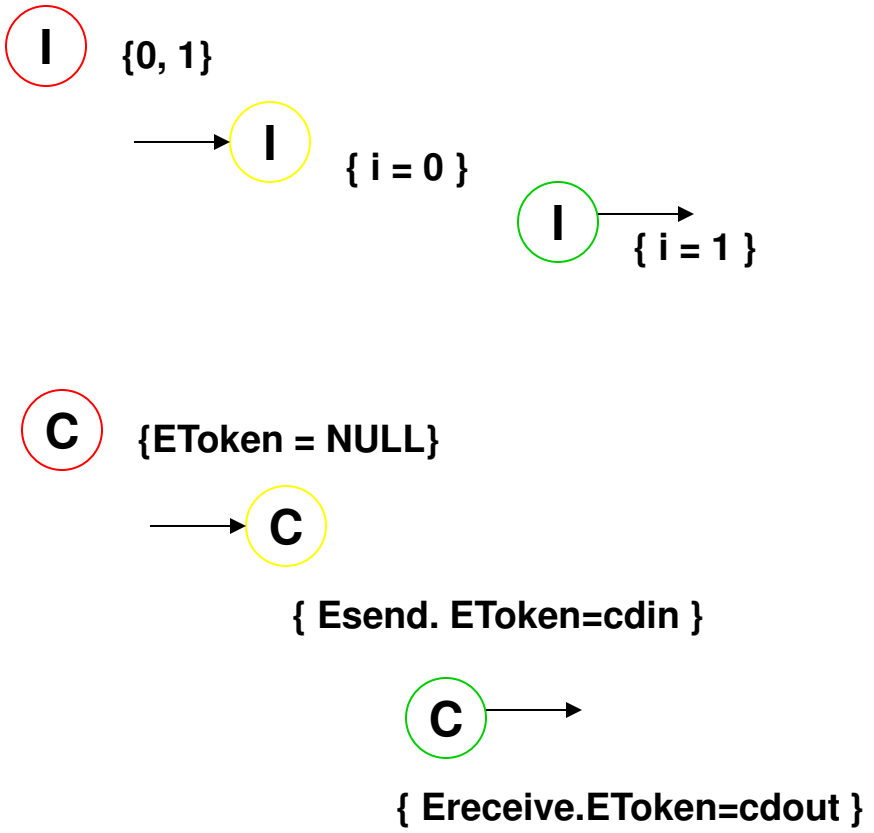
```
PROCESS Initial(AT: Any)
VARIABLES i:{"0","1"};
INIT ≜ i = "0";
ACTIONS
 start(st: AT) ≜ i="0" /\ st \in AT /\ i'="1";
END
```

Control flow edges

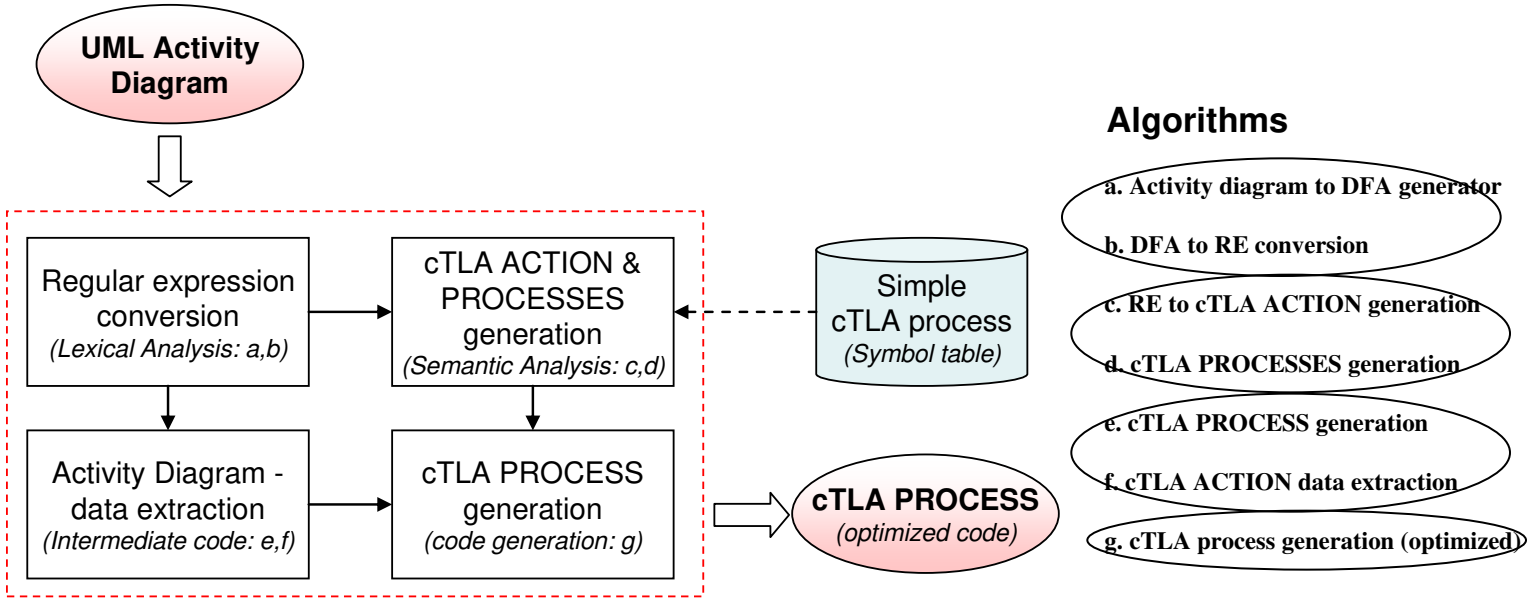
```
PROCESS ControlEdge(AT: Any)
VARIABLES EToken;
INIT ≜  EToken = NULL;
ACTIONS
Esend(cdin: AT) ≜  EToken= cdin;
Ereceive(cdout: AT) ≜ EToken # NULL /\ cdout = EToken /\ EToken'=NULL;
END
```

Action node

```
PROCESS AcceptEvent(pdu_type: Any)
CONSTANTS FREE ∈ pdu_type
BODY
    VARIABLES channel: pdu_type;
    INIT ≜ channel = FREE;
    ACTIONS  receive(rd: pdu_type) ≜ channel ≠ FREE ∧ rd=channel  ∧ channel′ = FREE
;
END
```



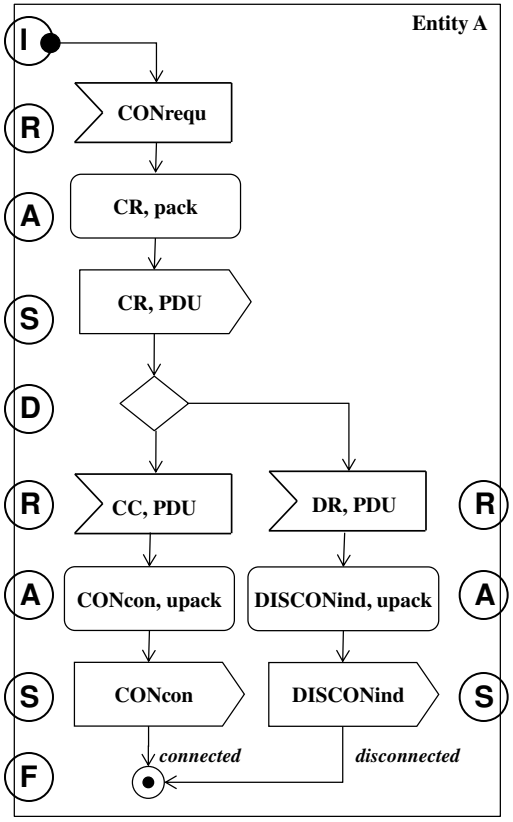
Functional semantics (1)

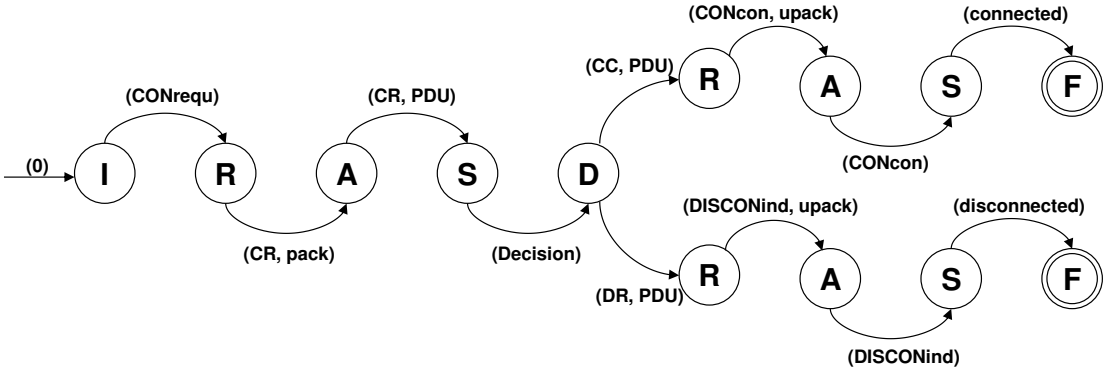


- Activity to cTLA Process Generator (A2cTLA)
  - Given an UML activity diagram as an XML format, A2cTLA process generator obtains an equivalent compositional cTLA process.
  - Similarly to compiling process

Functional semantics (2) – Step 1

- Activity diagram to regular expression conversion





Q: {I,R,A,S,D,A,S,F,S,F}
Σ: {(0),(CONrequ),(CR,pack), (CR, PDU),
(Dec:CC/DR), (CC, PDU), (CONconf, upack), ...}
δ: {δ(I,(CONrequ)=R), δ(R,(CR, pack)=A), ..}
q_o: {I}
F: {F}

RE: IRASD(RAS|RAS)F

States (Q), Transition inputs (Σ), State relations (δ), Start (q0), and End state (F) are identified, i.e. M = {Q, Σ, δ, q0, F}

Functional semantics (3) – Step 2

- cTLA PROCESSES and ACTIONS generation
  - Regular expression + simple cTLA = cTLA PROCESSES
  - Regular expression + simple cTLA (functional declaration) = cTLA ACTIONS

RE: IRASD(RAS|RAS)F

```
PROCESS Action(pdu_type: Any, flag: Any)
IMPORT pack_PDU, upack_PDU, analyze_data;
BODY
    VARIABLES pdata;
    INIT ≙ pdata = NULL;
    ACTIONS  execute(ad: pdu_type, at: flag)
                    ≙ at ∈ {“pack”, “unpack”, “analyze”}:

          at = “pack” ∧ pack_PDU(ad) ∧ pdata´= at;
          at = “unpack ∧ unpack_PDU(ad) ∧ pdata´= at;
          at = “analyze” ∧ analyze_data(ad) ∧ pdata´ = at;
END
```

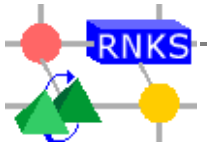
cTLA PROCESSES	cTLA ACTIONS
In: Initial(AT: Any)	In.start(st: AT) ∧
Acc: AcceptEvent(pdu_type: Any)	Acc.receive(rd: pdu_type) ∧
Act: Action(pdu_type: Any, flag: Any)	Act.execute(ad: pdu_type, at: flag) ∧
……	……..
Fin: Final(AT: Any)	Fi.stop(fin:AT) ∨

Functional semantics (4) – Step 3

- cTLA PROCESS generation
  - Meta-labels inclusion
    - → PROCESS
    - → BODY
    - → VARIABLES
    - → INIT
    - → PROCESSES
    - → ACTIONS
    - → END
- A predefined template
  - Automatically generated!

compositional cTLA process

A. **PROCESS** ConEst_Entity_A(Entity_B)
B. **BODY**
C. **VARIABLES**
PSS: {“connected”, “disconnected”};
D. **INIT** ≙ PSS = NULL;
E. **PROCESSES**
In: Initial(AT: Any);
Acc: AcceptEvent(pdu_type: Any);
…….
Dec: Decision(dec: Any, action: Any);
Fi: Final(AT: Any);
F. **ACTIONS**
con(du:SDU) ≙
(In.start(0) ∧
Acc.receive(…) ∧
Act.execute(…) ∧
Sen.send(…) ∧
Dec.decide(…) ∧
……
Fi.stop(…));
G. **END**

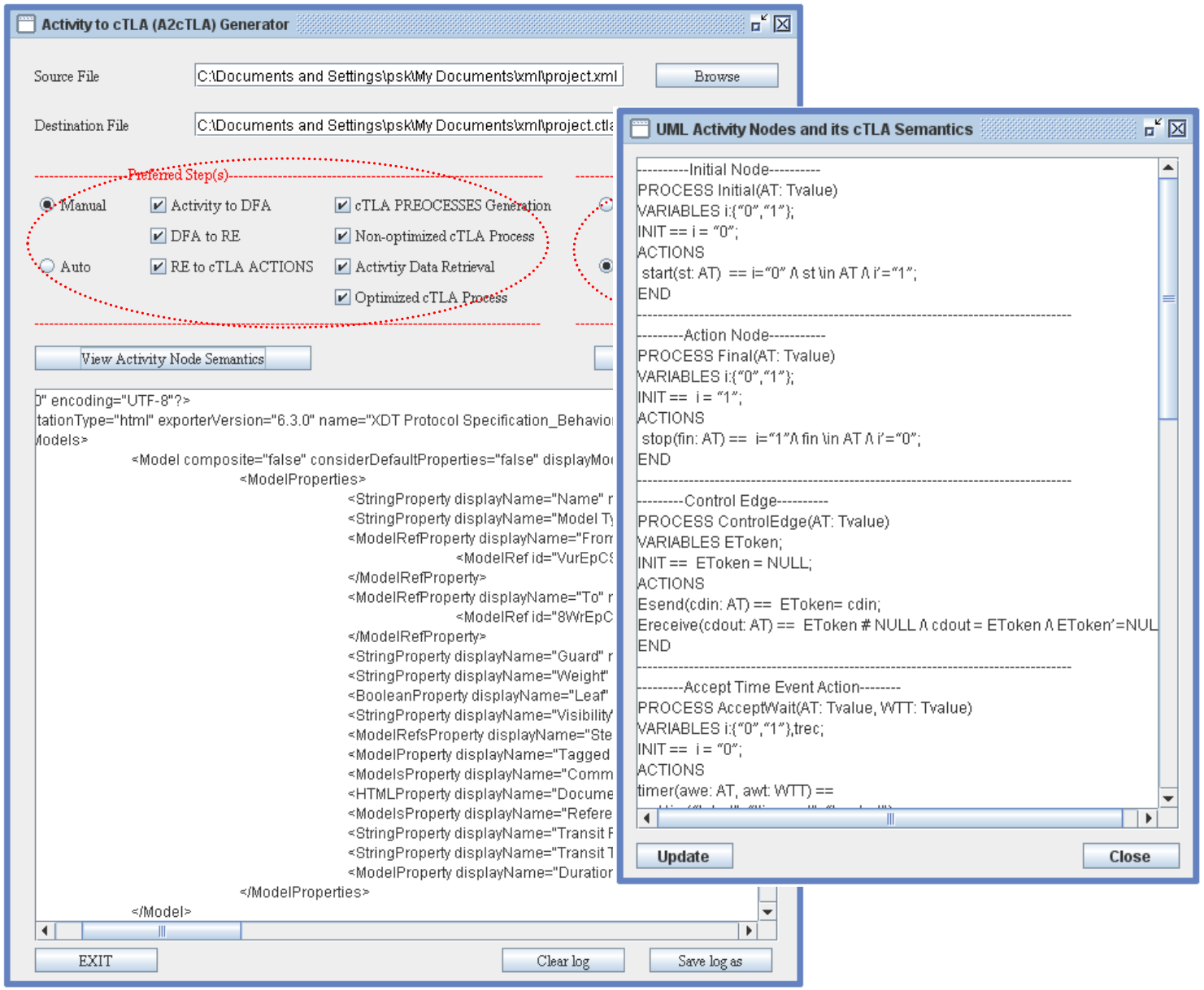


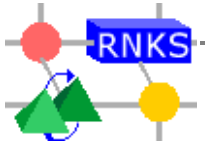
Functional semantics (5) – Final step

compositional cTLA process	
A. **PROCESS** ConEst_Entity_A(Entity_B)	/* Process name */
B. **BODY**	/* cTLA process structure begins */
C. **VARIABLES** PSS: {"connected", "disconnected"};	/* Process state status variables */
D. **INIT** ≜ PSS = NULL;	/* Variable(s) initialization */
E. **PROCESSES**	/* Processes declaration */
In: Initial(AT: Any);	/* Initial node */
Acc: AcceptEvent(pdu_type: Any);	/* Accept event node */
…….	
Dec: Decision(dec: Any, action: Any);	/* Decision node */
Fi: Final(AT: Any);	/* Final node */
F. **ACTIONS**	/* Process actions */
con(*du:SDU*) ≜	/* cTLA execution part begins */
(In.start(0) ∧	/* Initial state */
Acc.receive(*CONrequ*) ∧	/* Waiting for the SDU to arrive */
Act.execute(*CR, pack*) ∧	/* Encoding the SDU as data packet CR */
Sen.send(*CR, PDU*) ∧	/* Transferring the packet to receiver entity */
Dec.decide(*CC/DR*) ∧	/* Check for the arrival packet */
……	
Fi.stop(*disconnected*)));	/*End of unsuccessful connection set up */
G. **END**	/* cTLA process terminates */

A Successful connection establishment component for communications protocol – Entity A (fragment)

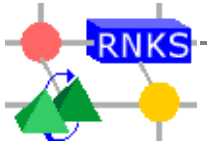
A2cTLA generator – Tool snapshot





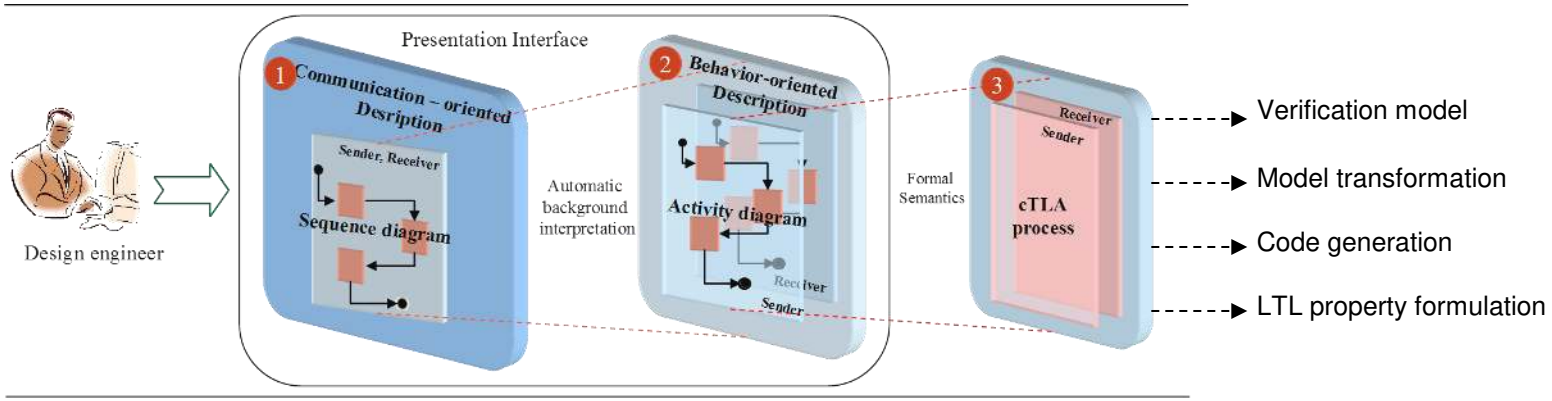
Benefits

- To formally interpret the system design specification
- Temporal logic of actions
  - Canonical form
    - → Can be used to verify system behaviors
    - → Possible to fomulate time ordering events as properties to prove whether it hold in the cTLA process or not
- Model transformations
  - Standard strucutre - appropriate for model transformations
    - → Specification Language - Lotos, etc.
    - → Verification Language - Promela, etc.



5. Final remarks

- Activity nodes defined by simple cTLA process
- Functional Semantics for activity diagram – compositional cTLA process
- Activity to cTLA process generator
  - A tool to formalize the UML activities is introduced
- Problem(s) we are addressing at the moment
  - Automatically transform of the cTLA process onto a canonical form.



Thank you for your attention !