Formal methods for embedded software systems: Two problems

Meenakshi D'Souza

IIIT-Bangalore.

24th January 2019.

Embedded Control Software

Figure: Robotics

Figure: Avionics

Figure: IoT

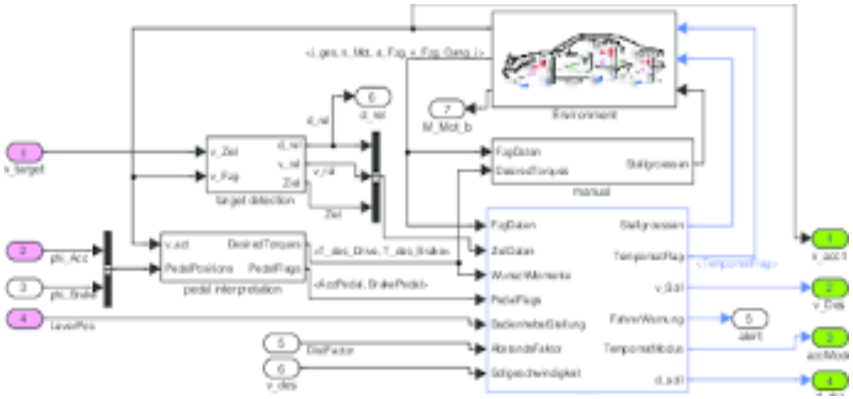


Figure: Simulink: Embedded Control Design

Embedded control software: Characteristics

- Runs on a proprietary real-time platform.
- Software tightly coupled with its environment.
- Distributed, real-time.
- Typically safety critical— subject to certification and regulatory requirements.
- Not feasible to shut down a malfunctioning system to restore safety or functionality.

One of the areas where formal methods is used by the industry.

Software for Control Automation

Software for automation engineering systems

- Software is used in automation systems to monitor and control various operations like batch processing, arc welding etc.
- Such software is implemented using domain-specific languages, most are proprietary in nature.
- Usually safety critical in nature, certification standards demand use of formal methods techniques.

Program analysis tools for automation software

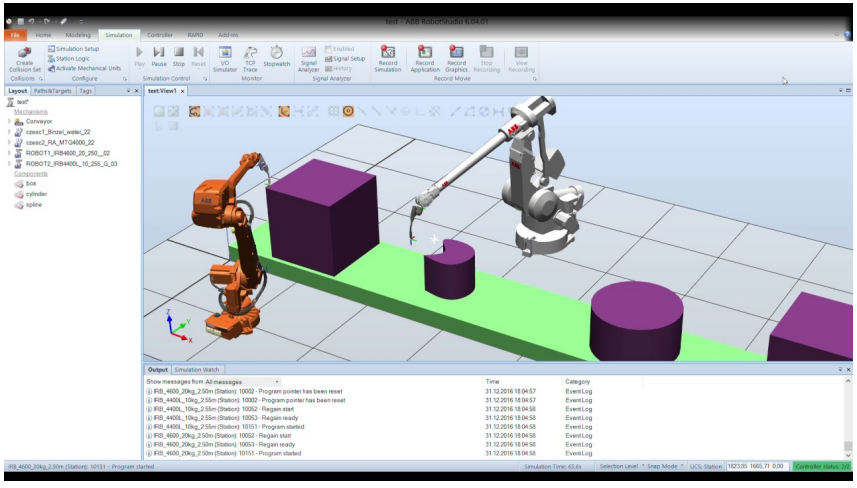
- Standard program analysis tools (Coverity for C, PolySpace for C/C++/Ada, Klocwork etc.) are not known to work for such languages.
  - Complex data types, task-based asynchronous/parallel execution, real-time system interrupts.
  - Tools do not port well across various development environments.
  - Many tools deploy pattern based matching to detect code violations, not known to scale for industrial automation tools.

Industry automation languages

We work with three programming languages used in industry automation:

- Rapid, a domain-specific language for programming industrial robot arms.
- IEC 61131-3 for PLC programming.
- Electronic Device Description Language (EDDL) used for configuration of field devices.

Rapid: A robotics programming language



- ABB robots are multi-axes industrial robots/robot manipulators.
- Typical actions done include welding, painting, picking, placing etc.
- Rapid is ABB proprietary language to program their robots.
- Robot instructions can be programmed using a teach pendant that generates Rapid code or directly using a textual interface.

Rapid in a nutshell: Data types

- Many standard datatypes are included in Rapid: `num`, `string`, `bool`, `array` etc.
- In addition, complex datatypes support co-ordinates in 3-D space, target positions for the robot arm etc.
  - `VAR pos := [500, 0, 940];`: Position in 3-D space.
  - `VAR robtarget p15 := [[600, 500, 225.3], [1, 0, 0, 0], [1, 1, 0, 0], [11, 12.3, 9E9, 9E9, 9E9, 9E9]];`: Position of a robot.
    First tuple: position in 3-D space,
    Last three tuples specify orientation of the tool, axis-configuration of the robot and the position of external axes respectively.

Rapid in a nutshell: Instructions

- `WaitTime 200;`: Instructs the robot to wait for 200 seconds before doing any assigned work.
- `IDelete intr;`: Disables the interrupt variable `intr`.
- `MoveL p1,v500,z10,tool1;`: Moves the position of the robotic tool `tool1` linearly to the position `p1`, with velocity `v500` and zone data `z10`.
  - This internally calculates the torque that needs to be applied to each axis (motor) to move linearly to the position `p1`.

Rapid in a nutshell: Program control flow

- Program flow: Written using standard imperative language constructs including relational and logical expressions, IF-THEN-ELSE statements and FOR and WHILE loops.
- Procedure calls are available, makes execution semantics complex.
- System generated interrupts and exceptions and their handling can alter the control flow of a program.

Rapid: An example

```
PROJECT TestProject                                              Project
TASK T1                                                          Task
MODULE TestModule ! ------ The main module                       Module
VAR num h; VAR num l1; VAR intnum sig1;
VAR robtarget r1:=[[502,500,500],[1,0,0,0],[1,1,0,0],[500,9E9,9E9,9E9,9E9,9E9]];
VAR robtarget r2:=[[402,400,400],[1,0,0,0],[1,1,0,0],[500,9E9,9E9,9E9,9E9,9E9]];
VAR robtarget r3:=[[302,300,300],[1,0,0,0],[1,1,0,0],[500,9E9,9E9,9E9,9E9,9E9]];
VAR robtarget r4; VAR signaldi di1; VAR tooldata tool1;
  PROC main()  !---- The main procedure ----                     Procedure
  SimpleBug;
  l1 := h - 10;
  SettoZero;
  CONNECT sig1 with SimpleError;
  ISignalDI di1, 1, sig1;
  r4 := [[h,h,h], [1, 0, 0, 0], [1, 1, 0, 0], [500, 9E9, 9E9, 9E9, 9E9, 9E9]];
  DrawShape;
  ENDPROC
  PROC SimpleBug()                                               Procedure
  !---- Illustration of a potential bug
  WHILE h < 3.4E+38 - 2 DO
      h := h+1;
  ENDWHILE
  ENDPROC
  PROC DrawShape()                                               Procedure
  MoveL r1, v500, fine, tool1;
  MoveL r2, v500, fine, tool1;
  MoveL r3, v500, fine, tool1;
  MoveL r4, v500, fine, tool1;
  ENDPROC
  PROC SettoZero()                                               Procedure
  l1 := 0;
  ENDPROC
  TRAP SimpleError                                               Procedure
  h := h+2;
  ENDTRAP
ENDMODULE
ENDTASK
ENDPROJECT
```

IEC 61131-3

- IEC 61131 is an open international standard for Programmable Logic Controllers (PLC).
- IEC 61131-1 deals with architecture and programming languages of the control program within PLC.
- Several standard data types, user defined data types including a kind of strongly typed pointer, I/O variables amongst others, program organization units that structure the code in a modular way.

IEC 61131-3: An example

```
CreateArray (Array => Array_var,
             FirstIndex := 1,
             LastIndex := 2,
             ArrayElement := array_element,
                             tatus => status);
index_val := status;
index_val := index_val +1;
putArray (Array := Array_var,
          Index := index_val,
          ArrayElement := array_element,
          Status => status);
index_val := index_val + 3;
putArray (Array := Array_var,
          Index := index_val,
          ArrayElement := array_element,
          Status => status);
while (run < 3) do
if (run < 3 ) then
   Mountain_Height := 5;
else
   Mountain_Height := 7;
end_if;
exit;
run := run + 1;
end_while;
```

EDDL

- EDDL (Electronic Device Description Language), an IEC standard, is a language for describing the service and configuration of field devices for process and factory automation.
- EEDL has data, communication (e.g. addressing information), user interfaces and operations (e.g., calibration).

EEDL: An example

```
MANUFACTURER HCF, DEVICE_TYPE SAMPLE1,
            DEVICE_REVISION 1, DD_REVISION 1
VARIABLE temperatureUSL {
      LABEL "Temp USL";
      HELP [upper_sensor_limit_help];
      HANDLING READ;
      TYPE FLOAT {DISPLAY_FORMAT ".3f";}
}
METHOD Sample_Method {
      LABEL "Sample Method";
      DEFINITION {
            int len1, x, y;
            char a[10], b[20], c[15];
            b=a+c;
            x=6;
            if(x<10) {
                  len1=6;
                  y=x/(x-len1);
            } else {
                  if(y>10)
                        len1=20;
            }
      }
}
```

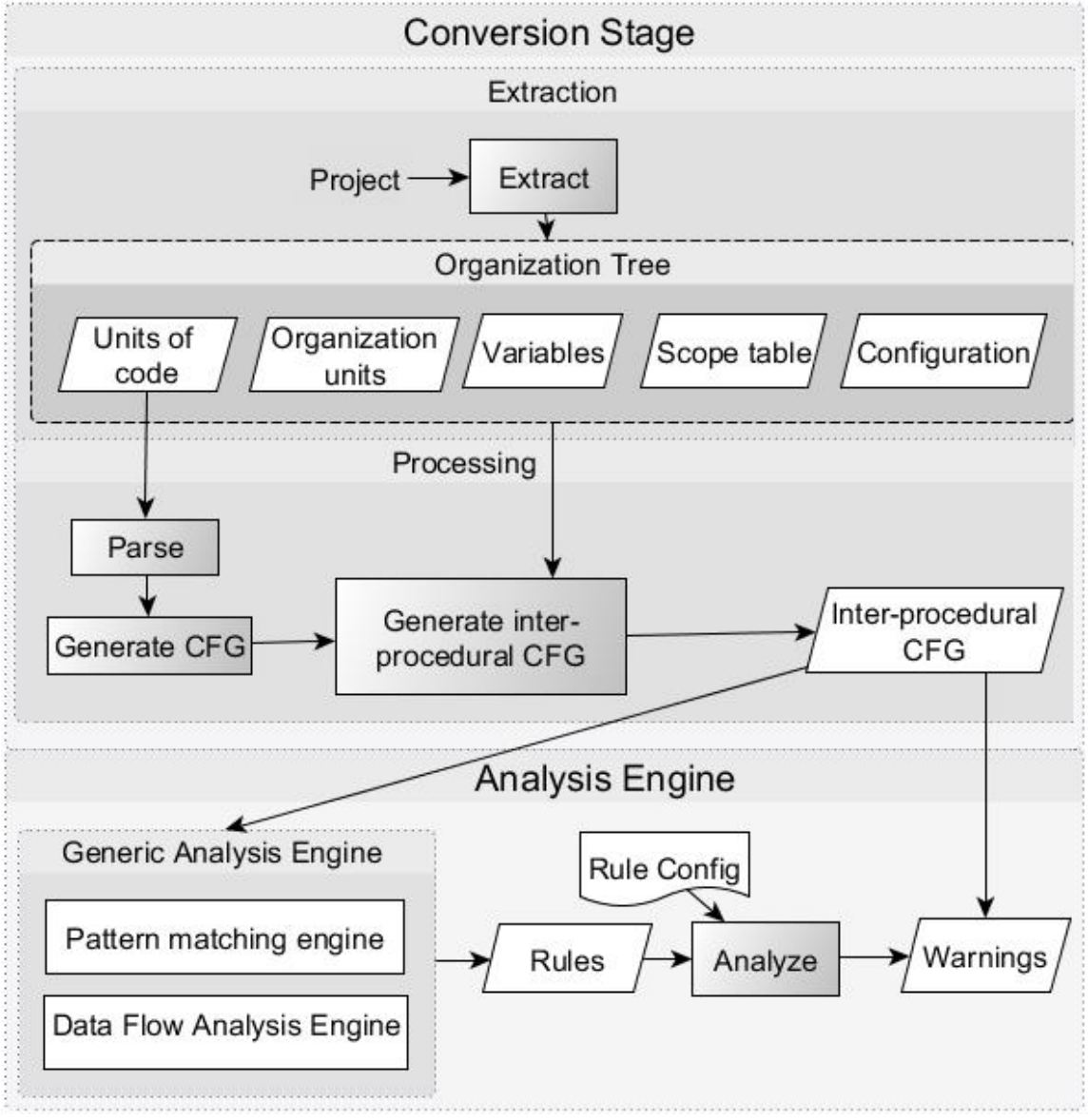
Common characteristics

- All three languages, although from disparate domains, characterize domain-specific languages for automation engineering.
- All of them support
  - Task based execution and use a modular structure for code organization. Execution can change based on interrupts.
  - Code is used to monitor and control various devices and controllers. Interrupts come from the platform.
  - Variables can be primitive as well as structured data types.
- Other languages like PLCopen, KRL, etc. support similar programming structures, and can be analyzed as well.

Program analysis framework: Key contributions

- Generic datatype to represent the parsed information for the three languages.
- Flexible Data Flow Analysis (DFA) engine to encode data flow rules as needed by varying the domain.
- Flexible rule engine to process data for further analysis.

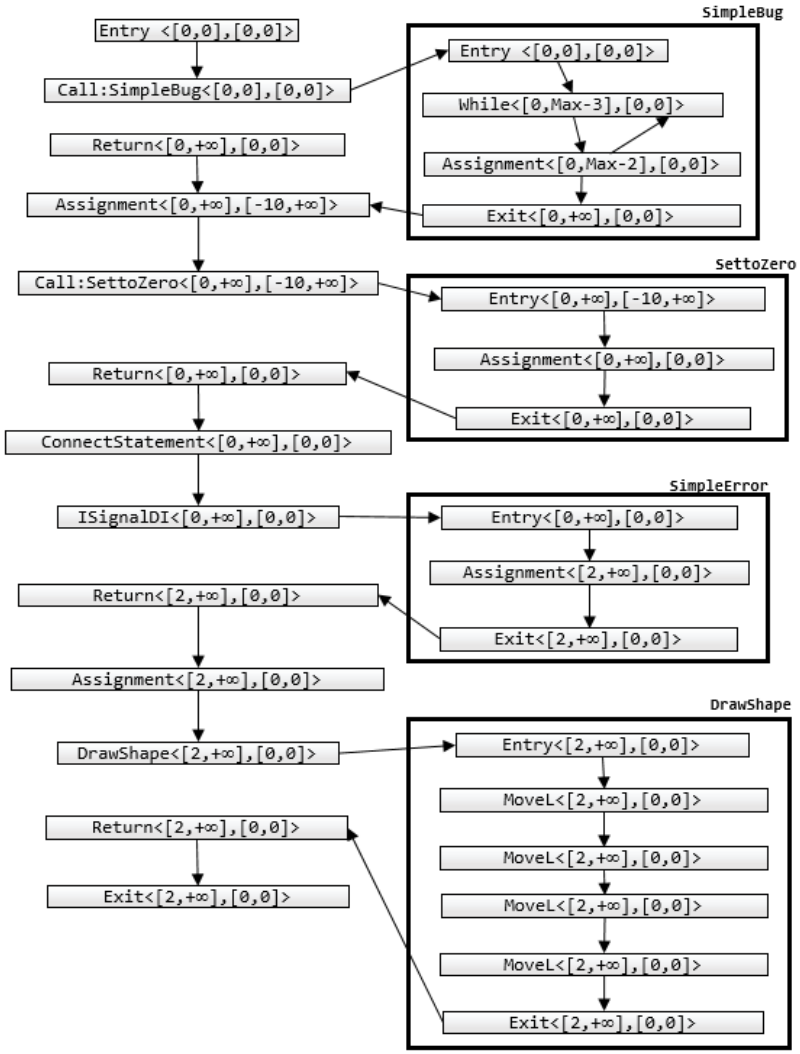
Rapid: Program analysis framework



Generic analysis engine

- Abstract-Syntax Tree (AST) and Control Flow Graph (CFG) are generated the usual way.
- Inter-procedural CFG needs one or more executions as input if procedure calls are not clear from the code.
  - CFG and inter-procedural CFG are annotated with data flow results.
- Data Flow Analysis (DFA) uses abstract interpretation based on interval domain abstraction to define transfer functions that support all standard arithmetic, logical and relational operations.
  - Interval domain semantics defined for all special datatypes.
- Several standard syntactic errors can be detected using AST and CFG.

Rapid code: Annotated inter-procedural CFG



Rules: Classification

- **Generic Programming rules:** Depend solely on the analysis framework, generic across all languages.
  - Division by zero, array out of bounds.
- **Language specific rules:** Based on a specific language, may not exist for all languages.
  - Boundary violation check for a robot arm (Rapid), variable re-definition rule (EDDL).
- **User specified rules.** Rules defined by the user based on a specific project or application.
  - Nesting levels of code, based on quality requirements.
  - Conformance to NORSOK standards for oil and gas applications.

Distribution of rules

Rules	Language	Generic errors	Language based	User specific
Pattern-based	IEC 61131-3	11	5	2
	EDDL	4	12	1
	RAPID	11	7	0
	Total	34	24	3
Semantic	IEC 61131-3	0	0	0
	EDDL	0	1	0
	RAPID	5	0	0
	Total	5	1	0
DFA	IEC 61131-3	5	1	1
	EDDL	4	2	0
	RAPID	1	4	1
	Total	10	7	2

Errors and rules: IEC 61131-3

Rule name	Category	Error	Warning
Incorrect Attribute	language specific	10	0
Uninitialized variable	generic	0	76
Datatype mismatch	language specific	0	9
Divide-by-zero	generic	0	7
Duplicate identifier	generic	0	1

Errors and rules: EDDL

Rule name	Category	Error	Warning
Divide-by-zero	generic	0	5
Missing Mandatory Menus	language specific	0	23
Unused variables	generic	0	24
Assignment for comparison	generic	3	0

Errors and rules: Rapid

Rule name	Category	Error	Warning
Illegal wait statement	language specific	4	0
Function side effects	language specific	0	4
Routines not used	generic	12	52
Unused variable	project specific	41	15
Arithmetic overflow	generic	2	2
Constant	project specific	0	30
Unreachable code	generic	3	7

Program analysis framework: Drawbacks

- Framework generates false positives.
  - Interval domain based abstraction is known to have this problem.
  - We are working on other abstraction techniques.
- Tool cannot handle recursion efficiently, we assume that worst case is reached when the function call stack exceeds a set limit.
- Works really well for Rapid, has handled code with ≥ 11000 statements and high cyclomatic complexity. Yet to be tested on large EDDL code.

Publications

- Joint work with Avijit Mandal and Raoul Jetley (ABB).
- 28th IEEE ISSRE 2017, industry track.
- 23rd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2018.

Modelling and verification of IoT protocols [1]

[1] Joint work with Maithily Diwan, Michael Butler

Introduction: Internet of Things

- Connects different computing devices, sensors, actuators, people and virtually any object.
- Prevalent in various industries like health care, automotive, manufacturing, power grid, domotics, etc.
- Gartner has predicted that there will be over 20 billion devices by 2020.
- Communication between these devices is an important aspect of IoT.

Introduction: IoT Communication Protocols

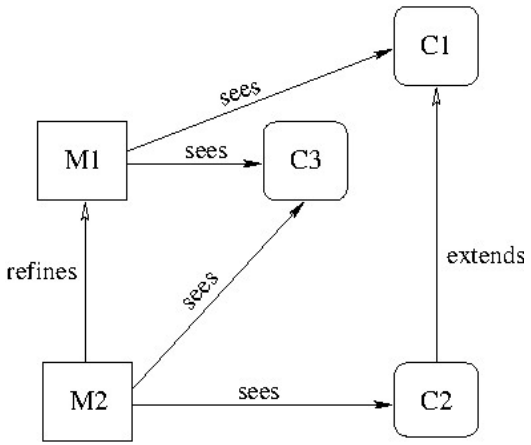
- Various protocols are used for communication in an IoTsystem. TCP/IP is a popular protocol used in lower layers.
- Protocols used in IoT systems posses properties like bandwidth efficient, light-weight and small code foot-print.
- Common IoT Protocol features: publish-subscribe, messaging layer, QoS(Quality of Service) levels, resource discovery, re-transmission, etc.
- Some protocols adapted for use in application layer in an IoT system - MQTT, MQTT-SN, CoAP, XMPP, AMQP.

Comparison of IoT communication protocols

Sl.No.	Protocol Feature	MQTT	MQTT-SN	CoAP
1	Architecture	Asynchronous Message exchange	Asynchronous Message exchange	REST architecture Layered Approach
2	Transport Layer	TCP	Any	UDP
3	Communication type	UniCast	UniCast/Multicast	UniCast/Multicast
4	Addressing	ClientID Server address	ClientID Server address	Uri Based
5	Messaging pattern	Publish Subscribe	Publish Subscribe	Request-Response Publish-Subscribe
6	QoS Levels	AtmostOnce, AtleastOnce, ExactOnce	AtmostOnce, AtleastOnce, ExactOnce	AtmostOnce, AtleastOnce
7	Persistent Session	Yes	Yes	Yes
8	Retained Message /Offline/Caching	Yes	Yes	Yes
9	Proxying/Caching	No	Yes	Yes
10	Resource Discovery	No	Yes	Yes
11	Sleep Mode	No	Yes	Yes
12	Security	Optional TLS	Optional TLS	Optional DTLS

Introduction to Event-B

- Uses set theory as a modeling notation and first order predicate calculus for writing axioms and invariants.
- Step by step refinement to represent systems at different abstraction levels and provides proofs to verify consistency of refinements.
- Has two types of components: contexts and machines.



Introduction to Event-B

Machine

- A machine has several events and can also define variables and its types.
- Can refine another machine to introduce new events, refine events, split or merge events.

Event

- An event consists of guards which need to be satisfied before the actions in events are executed.
- When an event is enabled and executed, the variables are updated as per the actions in the event.
- An invariant is a condition on the state variables that must hold permanently.

Rodin and ProB

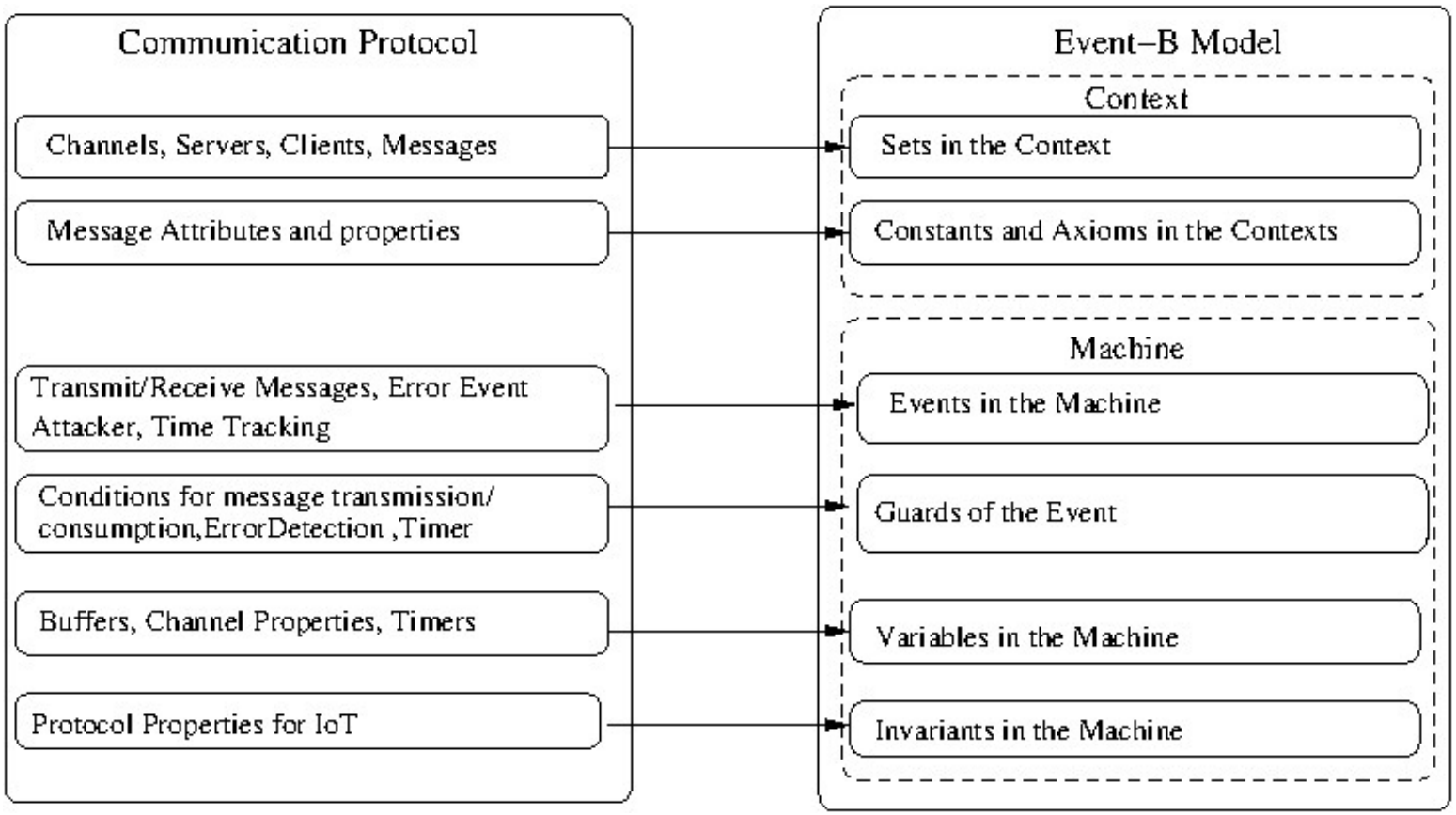
Rodin

- Implements Event-B and is based on Eclipse platform.
- Has sophisticated automatic provers like PP, ML and SMT.
- Provides interactive proving mechanism for manual proofs.
- Offers various plug-ins: text editors, decomposition/modularization tools, simulator ProB, etc.

ProB

- Provides a simulation environment through animation.
- Run executes a sequence of events.
- System state gives values of variables, evaluates invariants, axioms and guards for all the events.
- Deadlocks, invariant violation and errors can be detected.

Mapping between communication protocol and Event-B model



Protocol Modeling and Decomposition Using Event-B

The protocol modeling is done in two major steps:

- Building a common abstract model encompassing the common features of various protocols.
- Refining this common abstract model into a concrete model of a particular IoT protocol.

Our modeling is done using the techniques of machine decomposition, refinement and atomicity decomposition in Event-B.

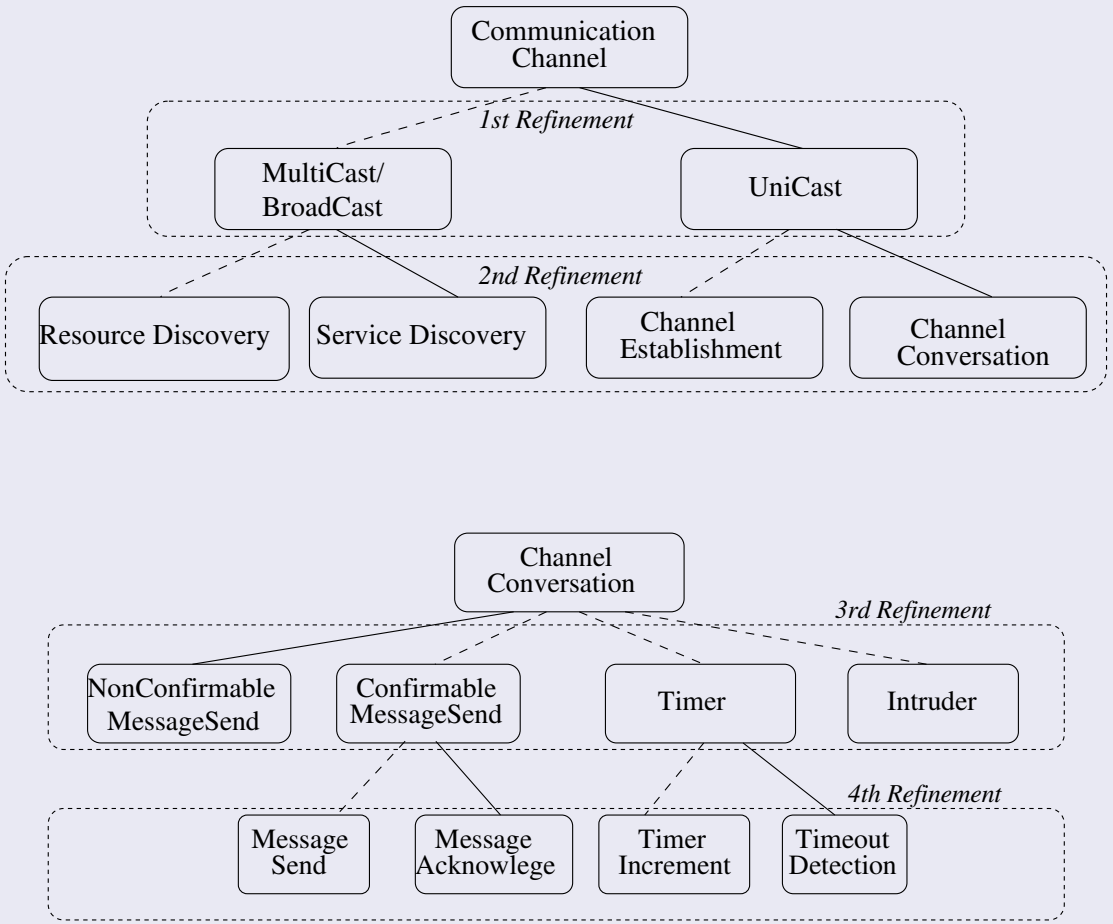
Common Abstract Model

Context

- Set MSG represents a message which is a basic communication entity.
- Attributes of a message are defined as relations over the set message and the sets defined for the attributes.
- A projection function is used to extract the value of an attribute for a given message.

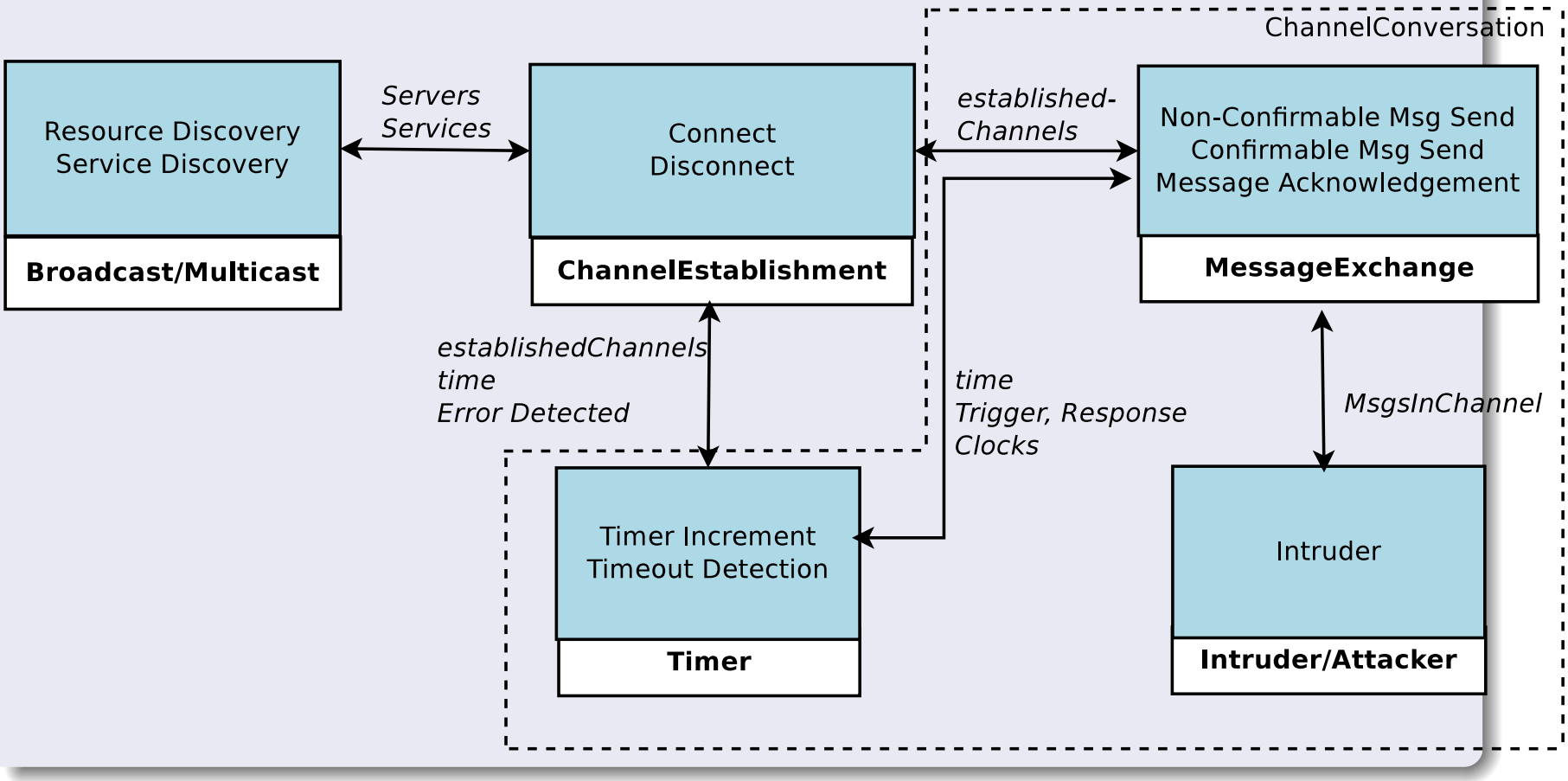
Common Abstract Model

Atomicity Decomposition



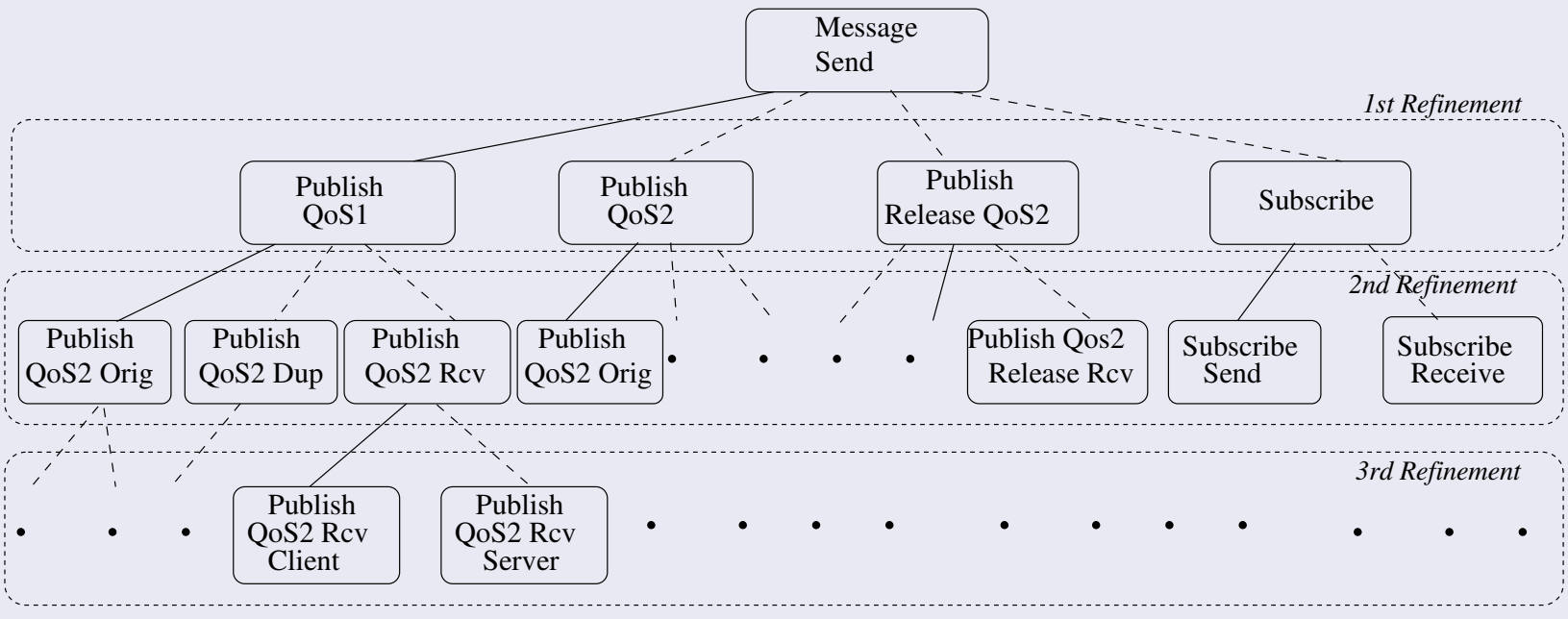
Common Abstract Model

Machine Decomposition



Concrete Protocol Models

Atomicity Decomposition



Concrete Protocol Models

Event for publishing message with QoS0

Event Client_Publish_QoS0 ⟨ordinary⟩ $\widehat{=}$

any

M10
ch
Topic

where

grd1: $Msg\_Topic(M10) = Topic$
grd2: $IncrementTime = FALSE$
grd3: $M10 \in MSG \land Topic \in TOPIC$
grd4: $PayloadCounter \in 0..9$
grd5: $(M10 \mapsto ((Publish \mapsto AtmostOnce) \mapsto PayloadCounter)) \in Msg\_Type\_QoS$

then

act1: $Client\_timer(ch) := 1$
act2: $IncrementTime := TRUE$
act3: $PayloadCounter := PayloadCounter + 1$
act4: $LimitTimer := 1$
act5: $Channel\_Direction(ch) := Client\_To\_Server$
act6: $MsgsInChannel(ch) := MsgsInChannel(ch) \cup \{M10 \mapsto PayloadCounter\}$

end

Model Validation

- ProB is used for validating our model through simulation of events and checking LTL properties.
- Accuracy of the model can be obtained by executing different runs and observing the sequence of events and variable values in each of these events.
- ProB also reports invariant violation or error in events which is then corrected in the model.
- Model validation is also done by writing and verifying invariants.

Verification of IoT Properties Using Event-B

Properties are verified by writing them as invariants that have to be satisfied for all the events in model.

Message Ordering

If both client and server make sure that no more than one message is "in-flight" at any one time, then no QoS1 message will be received after any later one.

$$\begin{aligned}
&\forall ch \cdot \forall pc1 \cdot \forall pc2 \cdot ((pc1 \in 0 \cdot \cdot 9 \land pc2 \in 0 \cdot \cdot 9 \land ch \in establishChannel \\
&\quad \land (pc1 \in Client_MsgSentQoS2(ch) \lor pc1 \in Client_MsgSentQoS1(ch)) \\
&\quad \land (pc2 \in Client_MsgSentQoS2(ch) \lor pc2 \in Client_MsgSentQoS1(ch)) \\
&\quad \land (time > SendTRange(pc2) + Response_Timeout) \\
&\quad \land pc1 \neq pc2 \land (SendTRange(pc1) < SendTRange(pc2)) \\
&\quad \Rightarrow (RcvTRange(pc1) \leq RcvTRange(pc2))
\end{aligned} \tag{1}$$

Verification of IoT Properties Using Event-B

Persistent Session

When a client reconnects with "CleanSession" set to 0, both the client and server must re-send any unacknowledged publish packets (where QoS > 0) and publish release packets using their original packet Identifiers. The variable RcvTRange is updated with current time only after the message is received. Hence it should be greater than the SendTRange time.

$$\begin{aligned}
&\forall ch \cdot \forall pc \cdot ((pc \in 0 \cdot \cdot 9 \wedge ch \in establishChannel \\
&\quad \wedge Channel_CleanSess(ch) = FALSE \\
&\quad \wedge ((pc \in Client_MsgSentQoS1(ch)) \vee (pc \in Client_MsgSentQoS2(ch)) \\
&\quad \wedge (time > (SendTRange(pc) + Response_Timeout)) \\
&\quad \Rightarrow (RcvTRange(pc) > SendTRange(pc)))
\end{aligned} \tag{2}$$

Verification of IoT Properties Using Event-B

QoS of a message from Client1 to Client2

The effective QoS of any message received by the subscriber is minimum of QoS with which the publishing client transmits this message and the QoS set by the subscriber while subscribing for the given topic.

$$\begin{aligned}
&\forall ch \cdot \forall pc \cdot \forall chnl \cdot \forall msg \cdot ((pc \in 0 \cdot \cdot 9 \land ch \in establishChannel \\
&\quad \land msg \in MSG \land chnl \in establishChannel \\
&\quad \land (pc \in Client_MsgSentQoS1(ch) \\
&\quad \land (msg \mapsto ((PUBLISH \mapsto AtleastOnce) \mapsto pc)) \in Msg_Type_QoS \\
&\quad \land ((Msg_Topic(msg) \mapsto ExactOnce) \in Channel_TopicQoS(chnl)) \\
&\quad \land ((time - SendTRange(pc)) \geqslant Response_Timeout))) \\
&\quad \Rightarrow (\exists QC \cdot ((QC \geq 1) \land Client_MsgReceived_2(chnl) = QC)))
\end{aligned} \tag{3}$$

Proof Obligations Results

Sl.No.	Protocol Property	Proof Obligations	Result
1	Duplicate Channel	10	Passed
2	Message Ordering	34	Passed
3	Persistent Session	34	Passed
4	QoS1 in single channel	26	Passed
5	QoS2 in single channel	26	Passed
6	Retained QoS1 message	24	Passed
7	Retained QoS2 message	24	Passed
8	Effective QoS0 in Multi channel(3 cases)	66	Passed
9	Effective QoS1 in Multi channel(3 cases)	66	Passed
10	Effective QoS2 in Multi channel(3 cases)	72	Passed
11	Request-Response Matching and Timeout	39	Passed
12	Confirmable Message ID Matching and Timeout	39	Passed
13	Exponential Backoff	39	Passed

Related Work

- In his paper published in 2014, the author Aziz. B., shows that there are scenarios where MQTT has failed to adhere to the QoS requirement.
- Gawanmeh. A's paper published in 2011, shows that a protocol used for IoT - Zigbee is verified for properties related to connection establishment propertie using Event-B.
- Authors Lee, S., Kim, H., Hong, D. K., Ju, H, of paper written in 2013, give methods to evaluate performance of MQTT protocol with regards to different QoS levels used and compare with other IoT protocol CoAP.
- In their paper of 2013, Authors Che, X., Maag, S, test connection properties using passive testing for XMPP protocol in IoT.

Conclusion and Future Work I

Conclusion

- Proposed and demonstrated use of a framework based on Event-B for modelling some of the widely used IoT protocols MQTT, MQTT-SN and CoAP.
- Properties verified: QoS, persistent session, will, retain messages, resource discovery, two layered request-response architecture, caching, proxying and message deduplication.
- We show that the protocols work as intended in an uninterrupted network as well as with an intruder which consumes messages in the network.
- Proposed and used a model of time based on intervals of time points, a new feature in Event-B.

Thank you!

Thank you! Questions?