# Simulation Model of TTP/C Protocol v1.0

Petr Grillinger
University of West Bohemia
Research group DSS

5. 4. 2004

# Outline

- Motivation
  - FIT project, 1st TTP/C model (v0.1)
- General TTP/C properties
  - Principles, versions
- New features of version 1.0
- Design decisions
  - Process communication
  - Modular structure
- Implementation in Python
  - Benefits and issues
- Model availability and restrictions

# Motivation

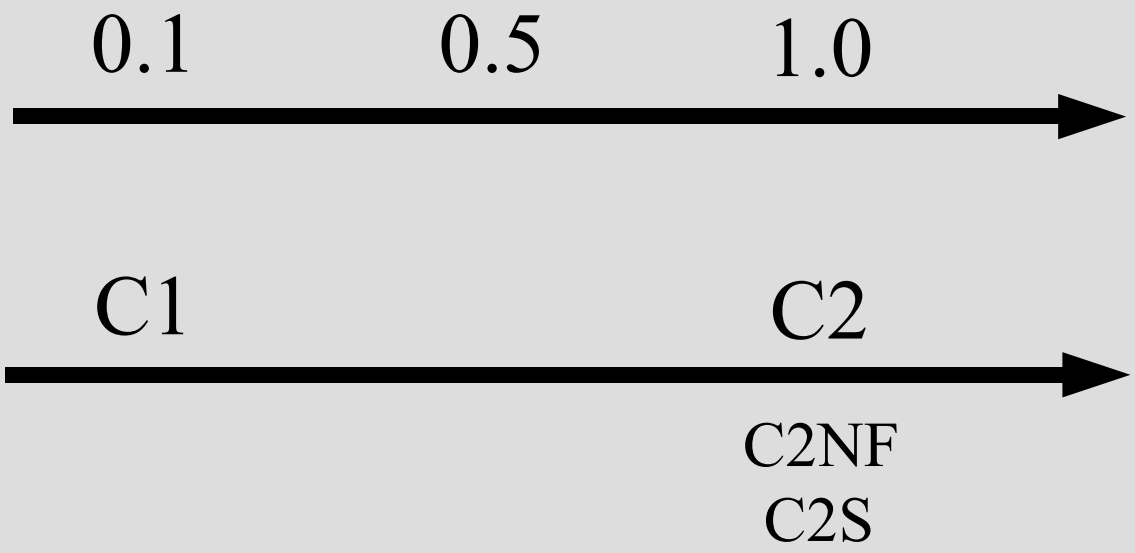
- Successful model of TTP/C 0.1 in C-Sim (FIT project, 1999-2002)
- New protocol version surpasses the verified 0.1 but so far no equivalent model exists
- TTTech (provider of TTP/C) wants the model of newest protocol v1.0
- Verification of safety critical protocol that one day may be used in “my car”
- Demonstration/Evaluation of new modeling approach

# TTP/C in General

- Distributed communication protocol for safety critical deployment (cars, planes, etc.)
- Separation of communication *(controller)* and application *(host)*
- FT provided by communication controller:
  - active/passive replication of nodes
  - replication of messages (channel, node, time)
  - node reintegration after failure
  - error detection mechanism (EDM)
  - strict timing (deadlines), synchronized global time
- Defined interface between controller and host

# TTP/C Versions

- Two independent version lines:
  - General protocol version (high-level specification)
  - Implementation version (low-level specification)



# New Features of TTP/C 1.0

- *The core remains the same as in version 0.1*
- Extension of some limits: message length, cluster size, communication period length
- New message type (X-Frame), better CRC
- Improved reintegration algorithm safety
- Better documentation: precise and complete
- Direct support of star architecture and central bus guardian (BG)
- More general MEDL structure

# Inter-Process Communication

- 0.1 model uses direct function calls and function hooks – not very clear
- Option 1 – message passing:
  - Language independent, allows distribution
  - Simple and stable object interface
  - High overhead, no language support
  - Slightly more complicated
- Option 2 – interface (virtual methods)
  - No overhead, very simple
  - Run-time name resolution in Python
  - Less readable and robust

# Message Handling Example

```
while self._receive(deadline):
   if self._message(id=FRAME):
      continue
   elif self._message(id=INTERRUPT):
      break
   elif self._message(id=CLOCK):
      continue
else:
   self._deadline_exceeded()
```

# Modular Structure

- *Original 2D modular structure works, but there may be something better?*
- Changed module and file name conventions:
  - Like Java (one class – one module)
  - Different functional levels in different directories (packages), e.g. *protocol* or *application*
- Separated simulation and functionality:
  - 2D directory structure seems too complex
  - Currently: functionality in base class, simulation added through multiple inheritance
  - So far no visualization

# Modular Structure Example

- psim
- Protocol
  - Functionality: *Controller, CNI, MEDL, Frame*
  - Simulation: *Controller, Clock*
- Application
  - Functionality: *Host*
  - Simulation: *Host, Cluster*
- Channel
  - Simulation: *Bus, Star*

```
from C2NF.Protocol.Functionality import *
from C2NF.Protocol.Simulation import *
```

# Benefits of Python Implementation

- Configuration files are scripts
  - More powerfull than simple data files
- Package-based modular structure
  - Namespaces, clear hierarchy
- Dynamic run-time name resolution:
  - No abstract methods / classes
- Direct access to data produced by TTPplan and TTPbuild (serialized Python classes)
- Much shorter source codes
- Possibility of portable GUI (tkinter)

# Issues of Python Implementation

- Low performance but can be improved:
  - Use JIT compiler
  - Use thread-free P-Sim implementation
- Not portable to most HW platforms
  - Can be used for algorithm-level verification
  - Cannot be used to design whole applications on model and transfer them later to HW
- Relatively unknown language
- Model developed under NDA with several restrictions

# State of the Art

- What is complete:
  - C2NF model without local BG
  - Bus channel implementation
  - Cluster setup framework
- What is under development:
  - Local bus guardian
  - Import from TTPplan data files
- Near future
  - C2S model
- Far future
  - Star architecture

# Availability of the Model

- The model is developed under NDA and sponsored by TTTech:
  - Is may not be published (source or otherwise)
  - Some of the specifications used during development are not public (most are)
- The model can be used for research and obtained results may be published
- Verification of FT is not part of agreement with TTTech

# Conclusions

- Python is superior prototyping language
  - Short development and debugging time
- Next model will be probably in C or Java:
  - Possible transition to industrial HW platform
  - Performance
  - No restriting NDA
- No FT verification has been done so far
  - It will be performed using FI after the model is finished
  - Especially star architecture is worth testing