Foundation of Contract for Things

C. Sofronis, O. Ferrante, A. Ferrari, L. Mangeruca
ALES S.r.l. – Rome

The Internet of System Engineering
INCOSE-IL Seminar, Herzliya, Israel
15 September, 2011

Software Platform for Integration of Engineering and Things
ICT-2009.1.3 Project Number: 257909
www.sprint-iot.eu
• SPRINT Use Cases
• Contracts introduction & SPRINT application
  – Role of Contracts in Action
• Contract based verification in SPRINT
• Contract Specification: Patterns language
• Conclusions
Heterogeneous System Design & Verification

- Distributed Design: Components developed and integrated over the cloud
- Heterogeneous: Components represented at different levels of abstractions
  - Models
  - Physical devices as prototypes: for internet-based integration
  - Physical devices as target components

Internet in the system: asynchronous execution of components
- E.g., a set of temperature sensors that communicate values over a TCP/IP protocol

employ

Contract Based Design
Contracts: Introduction

- Contracts are a Requirements formalization mechanism

**Contracts allow to reason on what the environment should guarantee to a component for its correct execution**

- **Contract**: pair of assertions (set of behaviors)
  - **Assumption**: Determine boundary conditions on design context under which component is promising its behavior
  - **Promise**: Guarantees of the component if used in assumed design context

- Contracts cover several aspects (multi-view)
  - Assumptions and Promises are organized in **viewpoints**
    - Behavior, Safety, Real-Time, Power ....
Contracts in Natural Language

**Contract of Client**

- **Client Assumptions**
  - Ack is received at most every two msg

- **Client Promises**
  - Msg are sent in bursts of at least two

**Contract of Server**

- **Server Assumptions**
  - At most two msg are received before ack is sent

- **Server Promises**
  - Ack is sent at most once for each msg
  - Ack is sent only after at least one msg is received

---

Client → msg → Server

Client → ack ← Server

Client → msg → Server

Client → ack ← Server
Contracts in Formal Language

Contract of Client

Contract of Server

SPRINT Consortium - All rights reserved 15-sep-2011
• Formal relations as methodology enablers
  
  – **Satisfaction**: Implementation M satisfies a contracts C iff
    \[ A \cap M \subseteq P \]
    
    • Intuitively: “All behaviors of the implementation that do not violate the assumption will not violate the promise”

  – **Compatibility**: Contract C1 is compatible with C2 iff
    \[ P_1 \cap P_2 \subseteq A_1 \cap A_2 \]
    
    • Intuitively: “There is no behavior that is possible but not acceptable by the assumptions”
Contracts role in SPRINT

Internet of Models
Internet of PD

Distributed HiL Simulation

Physical Device
Model

Contracts space
Implementations space

compatibility

satisfaction

C_{car}

C_{contr}

M_{car}

M_{contr}
Contract Verification by Simulation
- Generate *executable monitors* from Contracts to check satisfaction & compatibility
- Monitored component are on different implementation abstraction
  - Models of implementations
  - Physical devices: When physical devices are used, we cannot have a full virtual simulation
- **On-target (operation time)** verification
  - Embed the verification method in the operation-time implementation

Support **asynchronous** composition operator: RichConnector
- Describe different synchronization patterns between components
- Extend the current fully synchronous theoretical framework with native primitives

Verification by HiL simulation cannot guarantee preservation of design semantics. Enhance methodology: Distinguish contracts in:
- **Latency-sensitive**: over the internet verification cannot be guaranteed
- **Latency-insensitive**: over the internet verification cannot be influenced
System verification with HiL simulation

- Distributed HiL simulation
  - Component K1
    - M1: physical device prototype
    - C1: decomposed as sync and async parts
  - Component K2
    - M2: model
    - C2: decomposed as sync and async parts
- Verification by monitor synthesis
  - Executable specification from contracts’ FSMs
  - Contract composition respects the sync vs. async composition operator

Satisfaction check:
- M1 Satisfies \( C_1^{async} \)
- M2 Satisfies \( C_2^{async} \)
- Coverage of M1 Satisfies \( C_1^{sync} \)
- Coverage of M2 Satisfies \( C_2^{sync} \)
Requirement to contracts: ATS example

- **ATS system**
  - provides towing services in a given service area
  - centralized control subsystem (C4I)
  - user vehicles can request a tow-bot to the C4I
    - sending information such as its location, weight etc.
  - each tow-bot receives commands from the C4I
    - traveling autonomously to the user vehicle’s location
    - using a cruise-control sub-system
  - Communication between C4I, user vehicles and tow-bots is based on wireless channel

\[
F_T(kT) = k_p e(kT) + k_i \sum_{k=0}^{\infty} e(kT)
\]

- Cruise control sub-system
- Road slope
- Rolling friction
- Aerodynamic drag

- Proportional-integral control law (T = 40 ms)

\[
F_x = mg \cos(\alpha) \text{ sign}(v)
\]

\[
F_r = \frac{1}{2} \rho C_s A (v + w)^2
\]

\[
F_a = mg \sin(\alpha)
\]
• Example of natural language requirements for the C4I sub-systems:
  – When a user-vehicle request is received, if the vehicle has always been in the service area, a message will be send to a tow-bot to serve the request
• Example of natural language requirements for the tow-bot cruise control sub-system
  – If the road slope changes in the range -8% and 8%, the cruise speed is equals to the reference speed with a maximum error of ±5.5%.
  – The cruise control shall tolerate variations of the wind speed between -15 m/s (headwind) and +15 m/s (tailwind) with a maximum variation of 5 m/s every sampling period (T=40 ms)

Pattern-based contract specification allows the designer to formally capture plain-English contract using a semi-formal graphical or textual language
Pattern-based Contract Specification

- Graphical and textual language for the specification of contracts
  - Formalization of requirements in terms of assertions
    - Using a set of native library elements (atomic assertions)
    - Composing assertions using predefined assertion composition operators
- Patterns are logically layered
  - Assertion semantics is based on the classical temporal logic (LTL/CTL) semantics
  - Atomic assertions are provided as building blocks
    - Helper blocks allows for the definition of events, timers, etc...
  - Assertion composition layer
  - Assertion quantification layer provides blocks for the specification of repetitive assertions composition
  - Contract Specification layer allow for identification and composition of contracts
C4I Contract formalization

• Natural language requirements are organized in contracts first identifying assumptions and promises

“When a user-vehicle request is received, if the vehicle has always been in the service area, a message will be send to a tow-bot to serve the request”

• C4I contract
  – Assumption: the user vehicle position is inside the service area and the vehicle has been in the service area
  – Promise: when a user vehicle request message is received, a message will be send to a tow-bot to serve the request

• Each assertion is then expressed using the pattern language
• Finally the contract is identified combining an assumption and a promise
This block constraint the position of the user vehicle – received as part of the message – to be inside the service area of the C4I.

The atomic pattern assert that the condition expressed by the previous block must be true at every execution step.

The blocks emits an event when a message is sent or received.


This pattern constrains the event E2 (tow-bot message sent) to happen every time the event E1 (user vehicle request received) happens.
Conclusions

- Extend the basic synchronous semantics
  - Asynchronous composition of components and contracts
- System verification with contracts
  - Assumption-Promise reasoning
  - Satisfaction, Compatibility relations
  - Verification by Simulation
    - Monitors from contracts
  - Heterogeneous: different level of abstractions of implementations
    - Models
    - Physical devices HiL over the internet integration
  - Methodology for contract verification in the case of HiL over the net
    - Latency-sensitive
    - Latency-insensitive
- Patterns for Contract specification
  - High-level specification mechanism
Thank you!  תודה

Questions?
Andreas Keis
Engineering & Architecture
Manager Software Engineering
EADS Innovation Works
Quadrant Campus, Newport NP10 8FZ
United Kingdom
Tel: +44 (0) 1633 71 4760
Fax: +44 (0) 1633 71 3300
Mobile: +44 (0) 7970381972

Email: andreas.keis@eads.com
Linkedin: http://www.linkedin.com/in/andreaskeis