

Real Time / Embedded Components & Contracts

Prof. Jean-Marc Jézéquel

(Univ. Rennes 1 & INRIA)

Triskell Team @ IRISA

Campus de Beaulieu

F-35042 Rennes Cedex

Tel : +33 299 847 192 Fax : +33 299 847 171

e-mail : jezequel@irisa.fr

<http://www.irisa.fr/prive/jezequel>

Outline



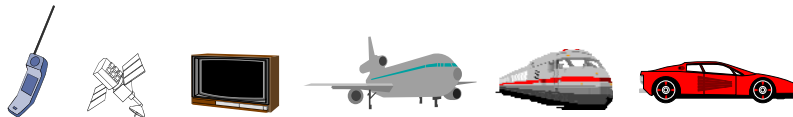
- Interest & Challenges of RTE Components
 - The Software Engineering Context
- Assembling Components: the Dangers
 - Towards trusted components
- Components and Extra-Functional Contracts
 - Reasoning about component & contract composition
 - Weaving contract monitoring
- Wrap up & Perspectives

Outline

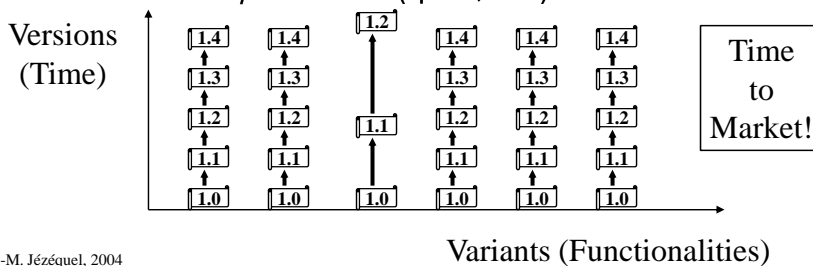


- **Interest & Challenges of RTE Components**
 - *The Software Engineering Context*
- **Assembling Components: the Dangers**
 - Towards trusted components
- **Components and Extra-Functional Contracts**
 - Reasoning about component & contract composition
 - Weaving contract monitoring
- **Wrap up & Perspectives**

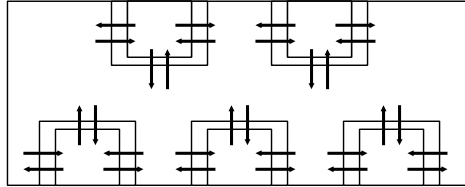
Building RT/Embedded Systems



- **Importance of non-functional properties**
 - distributed reactive systems, parallel & asynchronous
 - quality of service : reliability, timeliness, latency, performance...
- **Flexibility of functional aspects**
 - notion of *product lines* (space, time)



Components & Models



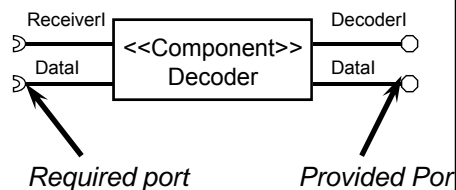
- A *component* is a unit of composition with contractually specified interfaces and fully explicit context dependencies that can be deployed independently and is subject to third-party composition (Szyperski)
- A *model* is a simplified representation of an aspect of reality for a specific purpose (Rothenberg)
 - E.g.; holistic reasoning on some aspects (correctness, performance...)

© J.-M. Jézéquel, 2004

5

From Objects to Components

- Object = instance of a class
- Class = reusable unit of software
 - focus on structural and functional properties
 - development concept
- Component = deployment unit
 - focus on non-functional properties
 - Installation concept
 - » Explicit dependencies
 - » Configuration and connection



© J.-M. Jézéquel, 2004

6

General Expected Benefits of Component Technology

- Structuring of Development process
 - Complexity,
 - Interfacing,
 - Integration,
- Maintenance Support
 - Modularization,
- Reuse
 - Used for standard system components
 - Within some development organizations
 - Exchange of Components between Organizations
 - Market for Software Parts

© J.-M. Jézéquel, 2004

7

Overview of existing component technologies

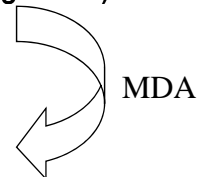
- 3 lines of widely adopted component technologies:
 - JavaBeans/EJB from Sun.
 - COM/DCOM/COM+/.NET from Microsoft
 - CORBA/CCM from OMG.
- Early component technologies offer protocols whereby independently developed components can be deployed and collaborate
 - JavaBeans, COM, and CORBA
- The technologies have subsequently developed into sophisticated platforms (e.g., EJB, .Net, CCM)
 - offer a variety of runtime services for managing component activation, concurrency, security, persistency, distribution, and transactions.

© J.-M. Jézéquel, 2004

8

Component models & components technologies

- "composition level": Fractal Model, UML
 - How to reason at the architecture level (design time)?



- "connection level": Java, EJB, .NET, Corba, Web Services
 - How to practically "wire" the pieces of code (run time)?

© J.-M. Jézéquel, 2004

9

Example

- Modeling a (simplified) GPS device
 - Get position, heading and speed
 - » by receiving signals from a set of satellites
 - Notion of Estimated Position Error (EPE)
 - » Receive from more satellites to get EPE down
 - User may choose a trade-off between EPE & saving power
 - » Best effort mode
 - » Best route (adapt to speed/variations in heading)
 - » PowerSave



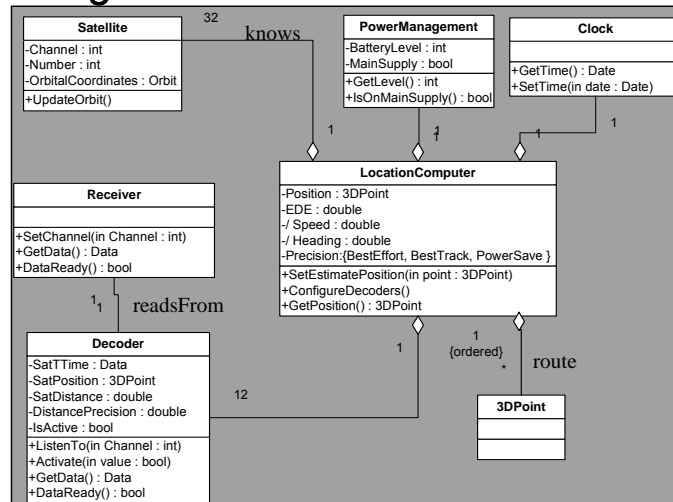
(Case Study borrowed from N. Plouzeau, K. Macedo & JP. Thibault. Big thanks to them)

© J.-M. Jézéquel, 2004

10

Modeling a (simplified) GPS device

■ Class diagram

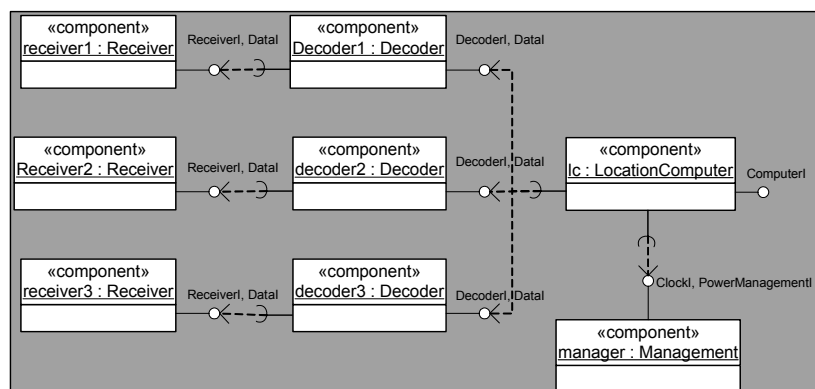


© J.-M. Jézéquel, 2004

11

Modeling a (simplified) GPS device

■ Deployment diagram



© J.-M. Jézéquel, 2004

12

Obstacles to RTE Development

- **Lack of Wide Adoption**
 - COM, .NET, CORBA, ... do not address extra-functional properties,
- **Techniques for extra-functional properties not enough developed, e.g.,**
 - Supporting component technologies have deficiencies
 - Lack of uniform specification techniques
 - QoS prediction still an art, often done after design,
 - Systematic timing analysis applicable under certain restrictions, otherwise often ad hoc

Challenges and Work Directions

- **Integration of Large Open Systems**
 - Integration across many levels (devices → enterprise)
 - Many Communication protocols, Interoperability standards
 - Open to other segments (supervision, business)
 - Long-life products – survive many changes of technology
- **Current Trend: Adaptation of existing component technologies**
 - Use selected parts of COM, .NET, CORBA.
- **Future Directions**
 - Rich Interfaces (resources, QoS, ...)
 - Support for Analysis and Prediction of global QoS properties
 - Model Driven Development (MDA)
 - Standardization

Challenges for System Integration

- System-wide constraints (timing, resources)
 - Current practices:
 - » System testing and ad hoc engineering
 - » State-of-practice schedulability analysis: restrictions not always satisfied
- System requirements: availability, etc.
- Component interoperability
- Component non-interference

Outline

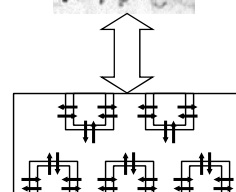
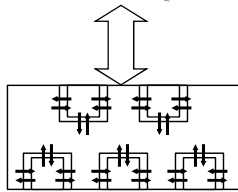


- Interest & Challenges of RTE Components
 - The Software Engineering Context
- *Assembling Components: the Dangers*
 - *Towards trusted components*
- Components and Extra-Functional Contracts
 - Reasoning about component & contract composition
 - Weaving contract monitoring
- Wrap up & Perspectives

Validity of component integration

How can we (re)use a component?

Component => Information Hiding
Trade-off on how much information is hidden



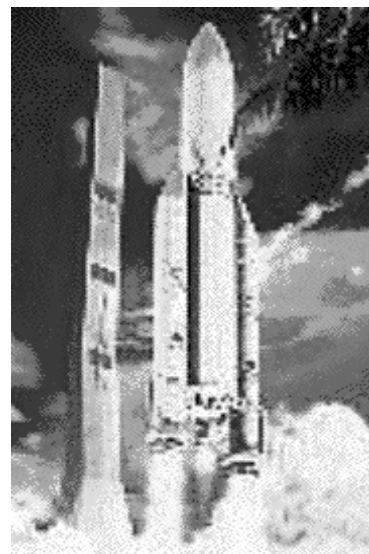
© J.-M. Jézéquel, 2004

17

Ariane 501 Maiden Launch

Kourou, ELA3 -- June, 4 1996, 12:34 UT

- H0 -> H0+37s : nominal
- Within SRI 2:
 - BH (Bias Horizontal) > 2¹⁵
 - convert_double_to_int(BH) fails!
 - exception SRI -> crash SRI2 & 1
- OBC disoriented
 - Angle > 20°, huge aerodynamics constraints
- boosters separating...



© J.-M. Jézéquel, 2004

Ariane 501 Maiden Launch

Kourou, ELA3 -- June, 4 1996, 12:34 UT

- H0 + 39s: Self destruction (cost: € 500M)



Why? (cf. IEEE Comp. 01/97)

- Not a programming error
 - unprotected conversion = design decision (~1980)
- Not a design error
 - Justified vs. Ariane 4 trajectory & RT constraints
- Problem with integration testing
 - As always, could have theoretically been caught. But huge test space vs. limited resources
 - Furthermore, SRI useless at this stage of the flight!

Why: (cf. *IEEE Comp. 01/97*)

- Reuse of a component with a hidden constraint!
 - Precondition : $\text{abs}(\text{BH}) < 32768.0$
 - Valid for Ariane 4, but no longer for Ariane 5
 - » *More powerful rocket*
 - *Ariane 5 flight envelope violated the implicit contract offered by SRI*

Specification = Contract between the client and the component

- In real life, many kinds of contracts
 - *From Jean-Jacques Rousseau's "Social Contract" to "cash & carry"*



- Likewise, many issues for software contracts in a distributed setting
 - Make them explicit to make it possible to:
 - » Reason about them (holistically)
 - » Check them (locally)

Four levels of Software Contracting

- **Basic (syntactic)**
 - the program compiles...
- **Behavioral**
 - Eiffel like pre/post conditions
- **Synchronization**
 - e.g. path expressions, etc. [McHale]
- **Quality of service (quantitative)**
 - Possible dynamic negotiation

*Cf. IEEE Computer
July 1999*

Level 2 Contracts in UML: OCL (Object Constraint Language)

- **Constraint = Boolean expression (no side effect) on**
 - Usual operations on basic types (Boolean, Integer...)
 - attributes of class instances
 - « query » operation (functions side-effect free)
 - associations from the UML class diagram
 - States from StateCharts

Behavioral Contracts

- Inspired by the notion of Abstract Data Type
- Specification = Signature +
 - Preconditions
 - Postconditions
 - Class Invariants
- Behavioral contracts are inherited in subclasses

Precondition: *Burden on the client*

- Specification on what must be true for a client to be allowed to call a method
 - example: amount > 0
- Notation in UML
 - {«precondition» OCL boolean expression}
 - Abbreviation: {pre: OCL boolean expression}

Postcondition:

Burden on the implementor

- Specification on what must be true at completion of any successful call to a method
 - example: `balance = balance @pre + amount`
- Notation in UML
 - {«postcondition» *OCL boolean expression*}
 - Abbreviation: {post: *OCL boolean expression*}
 - Operator for previous value (idem old Eiffel):
 - » *OCL expression @pre*

Interest of Behavioral Contracts

- Specification, documentation
 - Not a software fault tolerance gadget
 - Might help system fault tolerance...
- Help V&V
 - When assertions are monitored
 - » Must go from model to instrumented code: *Transformations*
 - Never doing debugging again
- Help allocate responsibilities during integration
 - No longer have to find a scapegoat ;-)

Outline

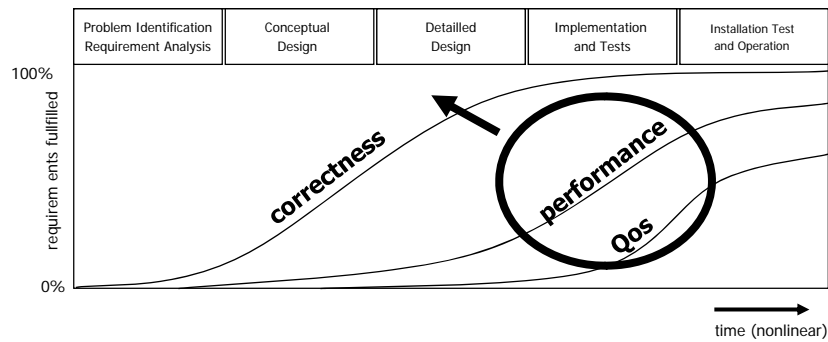


- Interest & Challenges of RTE Components
 - The Software Engineering Context
- Assembling Components: the Dangers
 - Towards trusted components
- *Components and Extra-Functional Contracts*
 - Reasoning about component & contract composition
 - Weaving contract monitoring
- Wrap up & Perspectives

Work Directions

- Build on existing composition mechanism,
- Standardize rich interfaces (w. timing, resources)
- Techniques for analysis of extra-functional properties
 - Scheduling Theory, Simulation, Probabilistic Analysis, Model Checking.
- Develop platform w. predictable resource/timing behavior (lightweight component framework)
- Model resource and timing properties of platform
- Non-interference → temporal and resource protection
- Services for run-time composition and replacement, failure handling, ...
- Develop standard system architectures,

Integrated consideration (from Ed Brinksma)

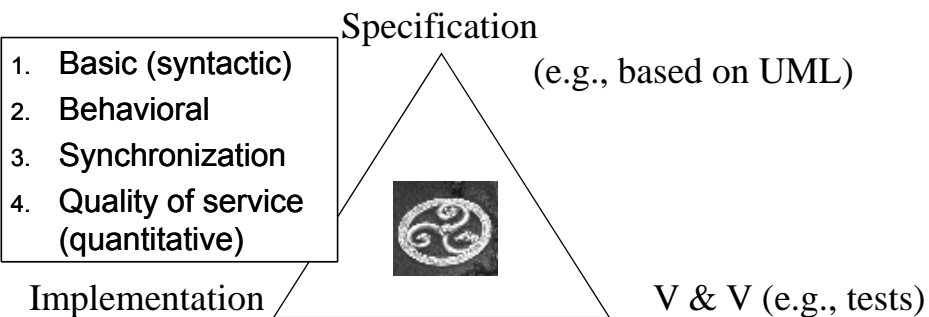


© J.-M. Jézéquel, 2004

31



Components and Contracts



Making Components Contract Aware

A Beugnard, JM Jézéquel, N Plouzeau, D Watkins
COMPUTER, 1999

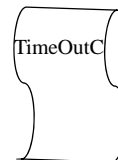
QoS Contracts (Level 4)

- QoS Dimension (from QML)
 - Name (responseTime, throughput...)
 - Type (float, int, bool, enum...)
 - Direction (up, down)
 - Unit (seconds, bytes, none ...)
- QoS Categories
 - To group a set of QoS Dimensions
- Contracts
 - Inherit one or more QoS categories
 - Bound to ports

QML: A Language for Quality of Service Specification
HP Labs Technical Reports
<http://www.hpl.hp.com/techreports/98/HPL-98-10.html>

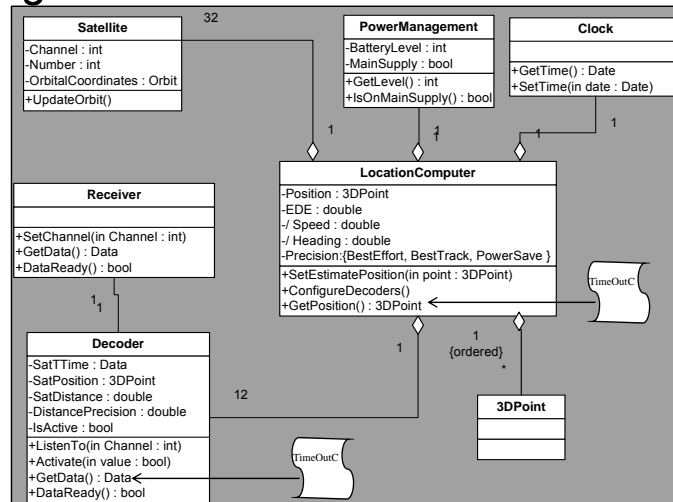
Example for the GPS

- Getting location data from a receiver should be done quickly enough
 - Can take a long time in case of radio reception problems
 - Big power consumption while the receiver is active
- TimeOut contracts for the GPS
 - Just one QoS dimension
 - » Name = responseTime
 - » Type = int
 - » Direction = down
 - » Unit = us



Example

■ Adding QoS contracts to our GPS device



© J.-M. Jézéquel, 2004

35

Outline

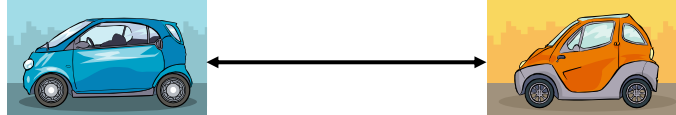


- Interest & Challenges of RTE Components
 - The Software Engineering Context
- Assembling Components: the Dangers
 - Towards trusted components
- Components and Extra-Functional Contracts
 - Reasoning about component & contract composition
 - Weaving contract monitoring
- Wrap up & Perspectives

© J.-M. Jézéquel, 2004

36

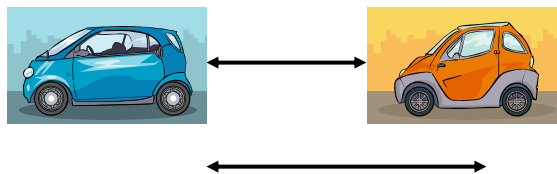
Quality of Service: Safety distance



© J.-M. Jézéquel, 2004

37

Quality of Service: Safety distance

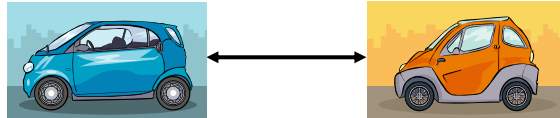


- Service : slow down when the safety distance is violated
- Quality of Service : don't stay too long with a violated safety distance

© J.-M. Jézéquel, 2004

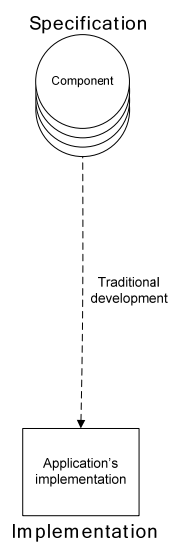
38

Quality of Service: Safety distance

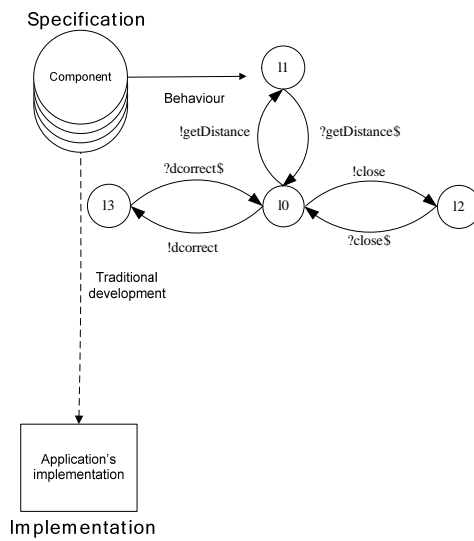


- Property: have a correct distance after a close distance with no more than 4 units of time

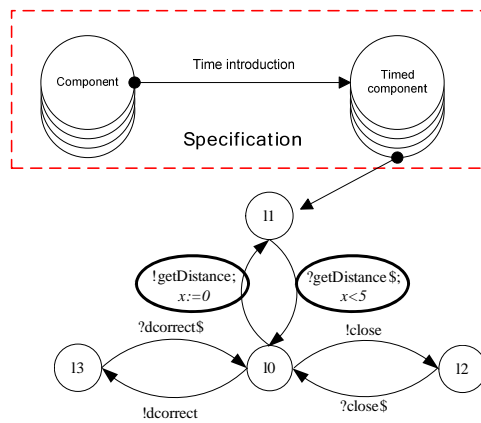
Process overview



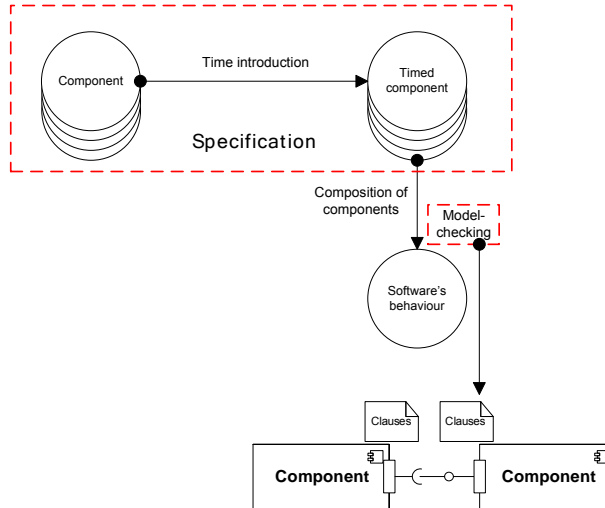
Process overview



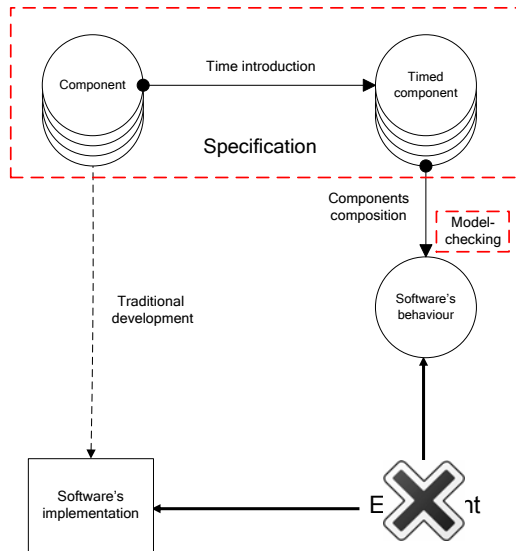
Process overview



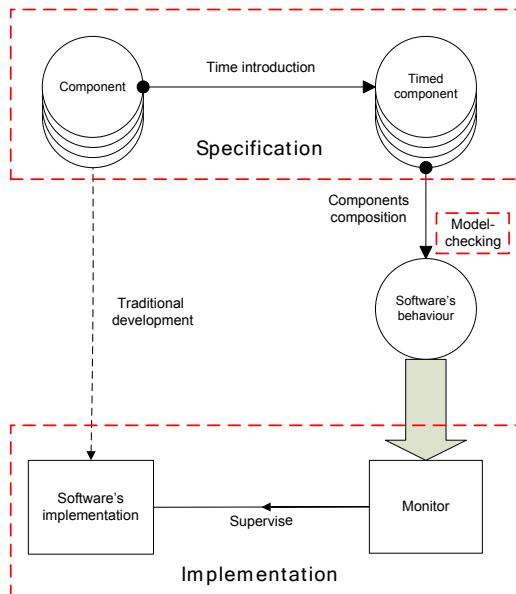
Process overview



Process overview



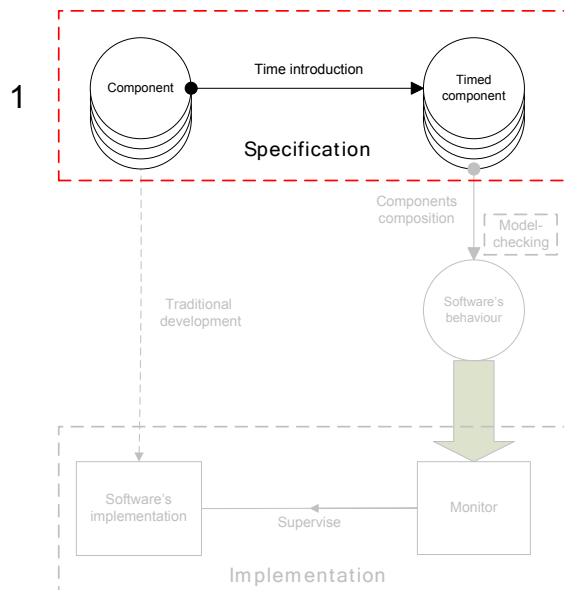
Process overview



© J.-M. Jézéquel, 2004

45

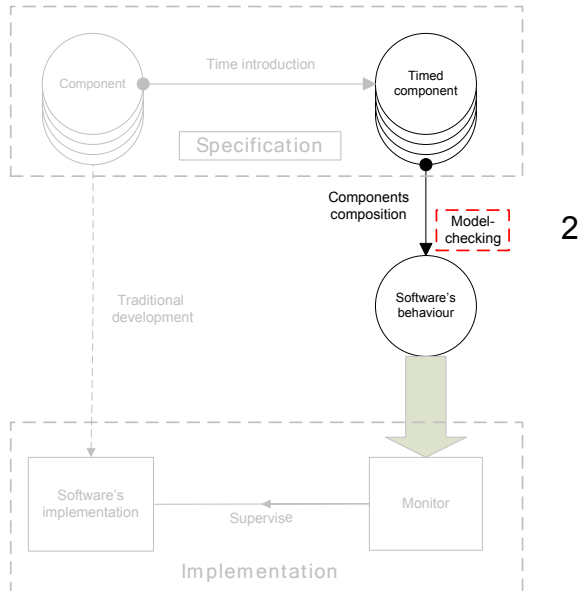
Process overview



© J.-M. Jézéquel, 2004

46

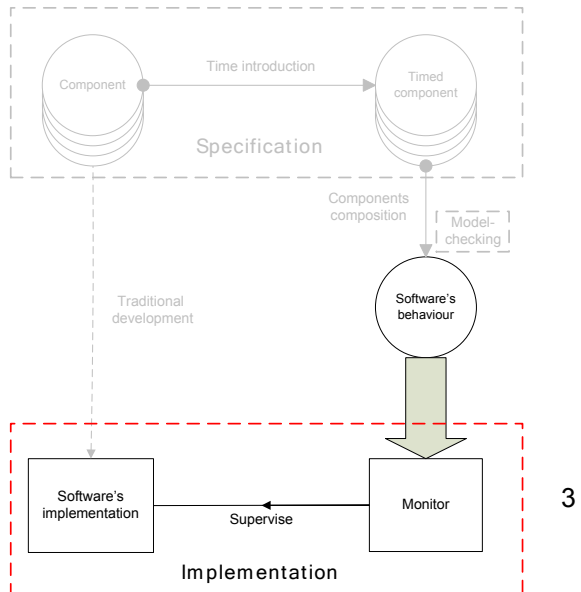
Process overview



© J.-M. Jézéquel, 2004

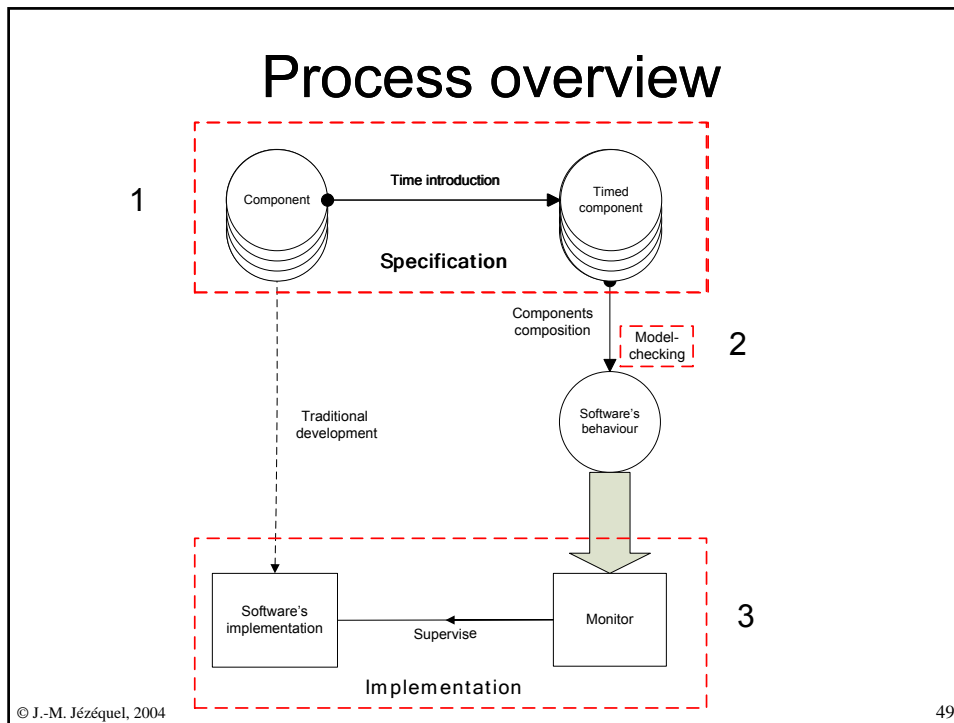
47

Process overview



© J.-M. Jézéquel, 2004

48



Add timed properties

- Introduction of time during specification
- Help the architect who does not know timed formalisms
- The properties must be used during the composition
- Introduction into clauses
 - Provided: the behaviour, automaton
 - Required: logical formula

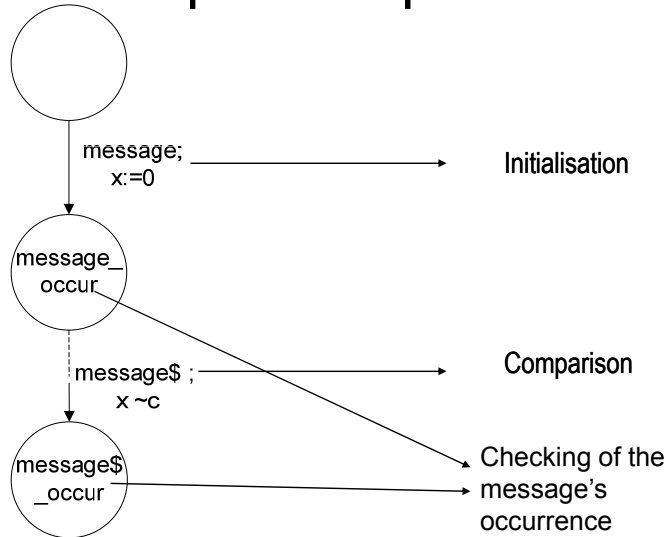
Patterns

- Reusability
- Separation time/functionality
- Easy to use
 - No knowledge on formalisms
 - Hidden semantic
- 2 types of patterns
 - For the behaviour
 - For the clauses

Timed patterns

- Add in the behaviour of component
- 1 property \rightarrow 1 pattern
- The pattern has parameters:
 - The messages
 - The operator
 - The value
 - The use

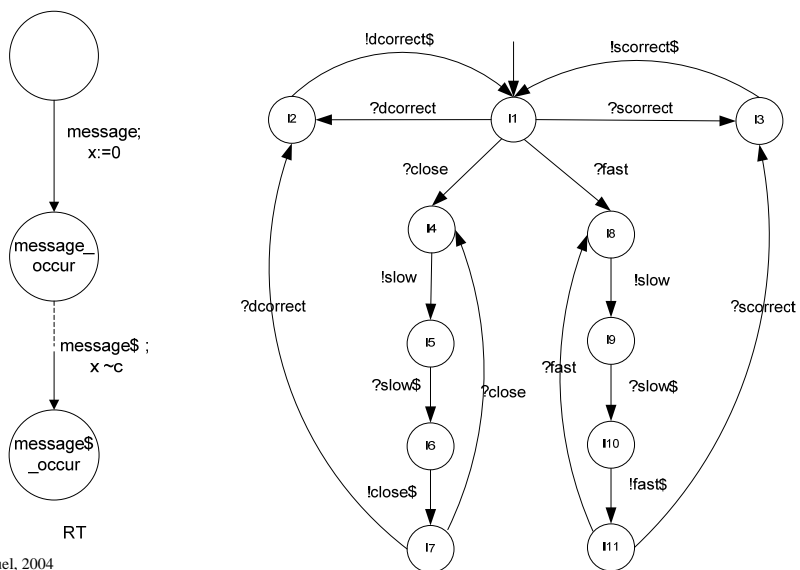
Example: Response time



© J.-M. Jézéquel, 2004

53

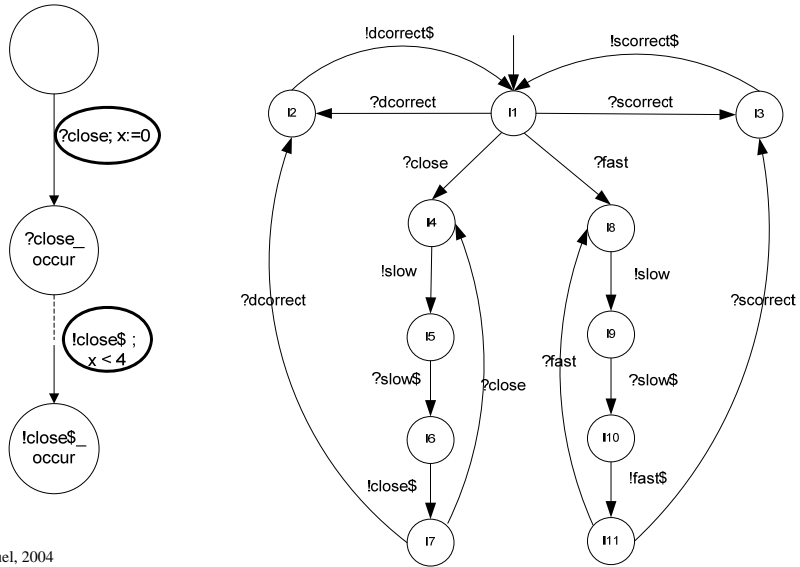
Applying a pattern



© J.-M. Jézéquel, 2004

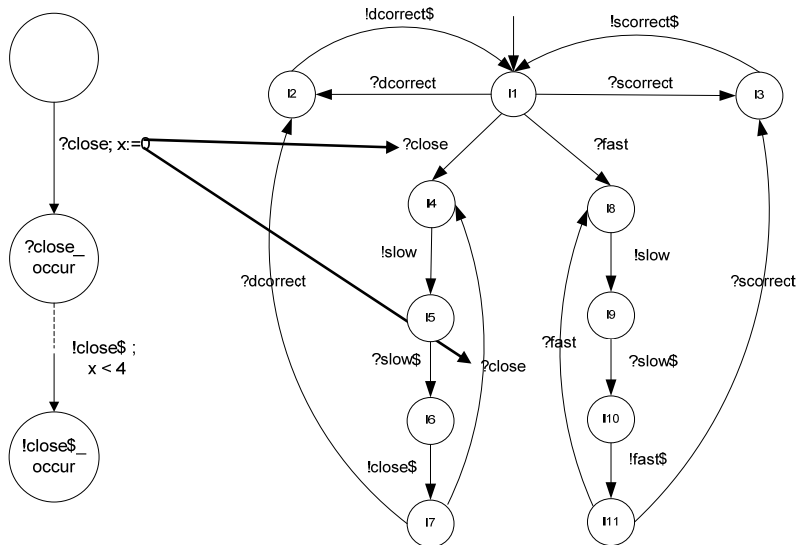
54

Applying a pattern(2)



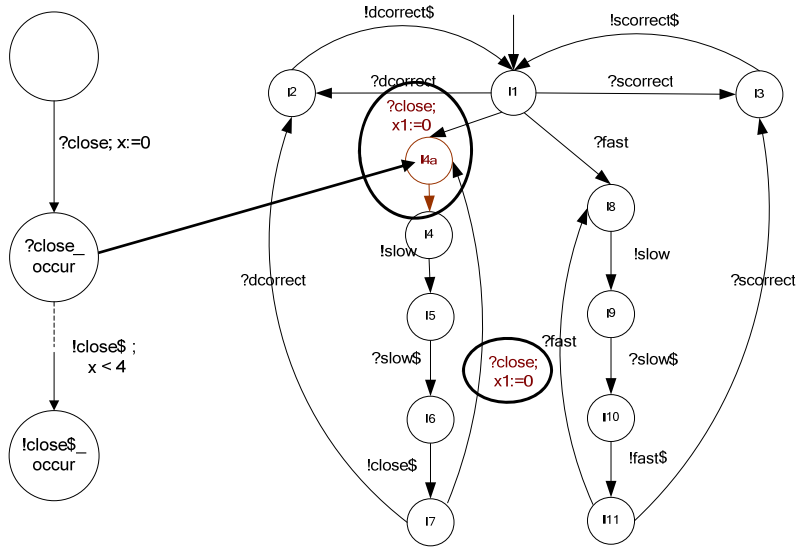
55

Applying a pattern(3)



56

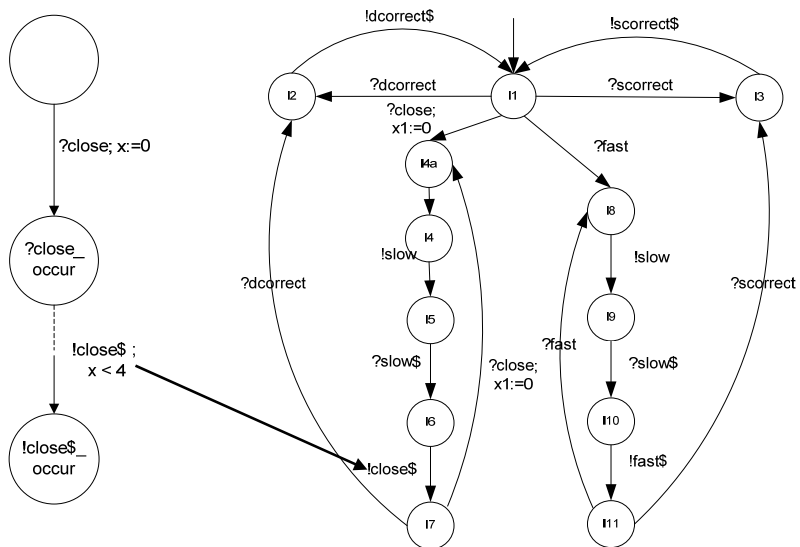
Applying a pattern(4)



© J.-M. Jézéquel, 2004

57

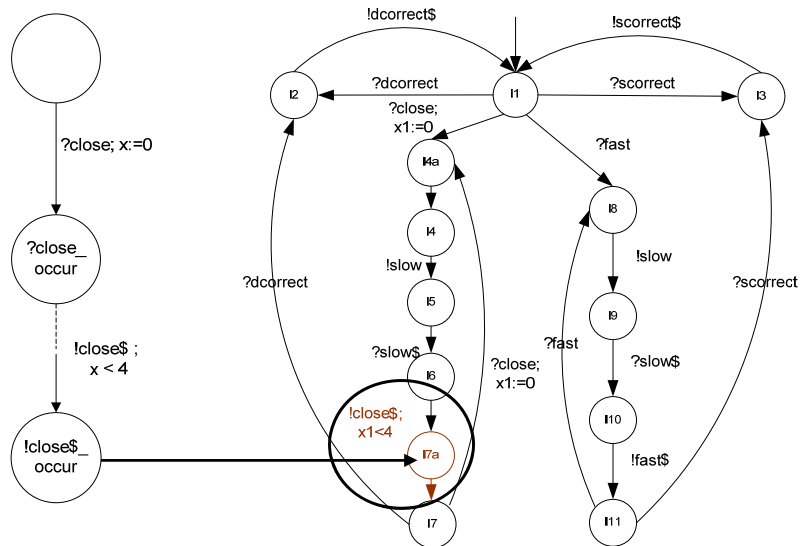
Applying a pattern(5)



© J.-M. Jézéquel, 2004

58

Applying a pattern(6)



© J.-M. Jézéquel, 2004

59

Times contracts

- Add time into clauses of component
- On messages exchange between components
- Parameters:
 - Messages
 - Operator
 - Value

© J.-M. Jézéquel, 2004

60

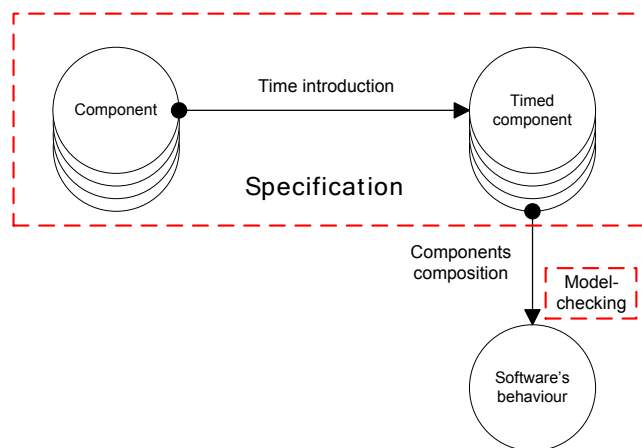
Examples

- The property p has a period of c units

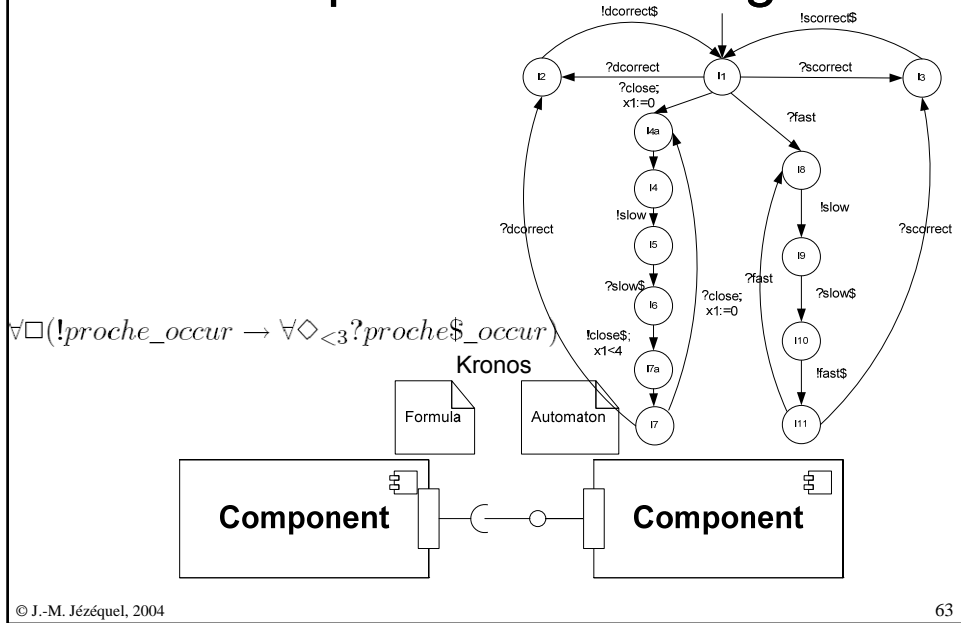
- There are c units ^{$\forall \square (\forall \diamond \sim c p)$} between $p1$ and $p2$

$$p1 \rightarrow \forall \square (\forall \diamond \sim c p2)$$

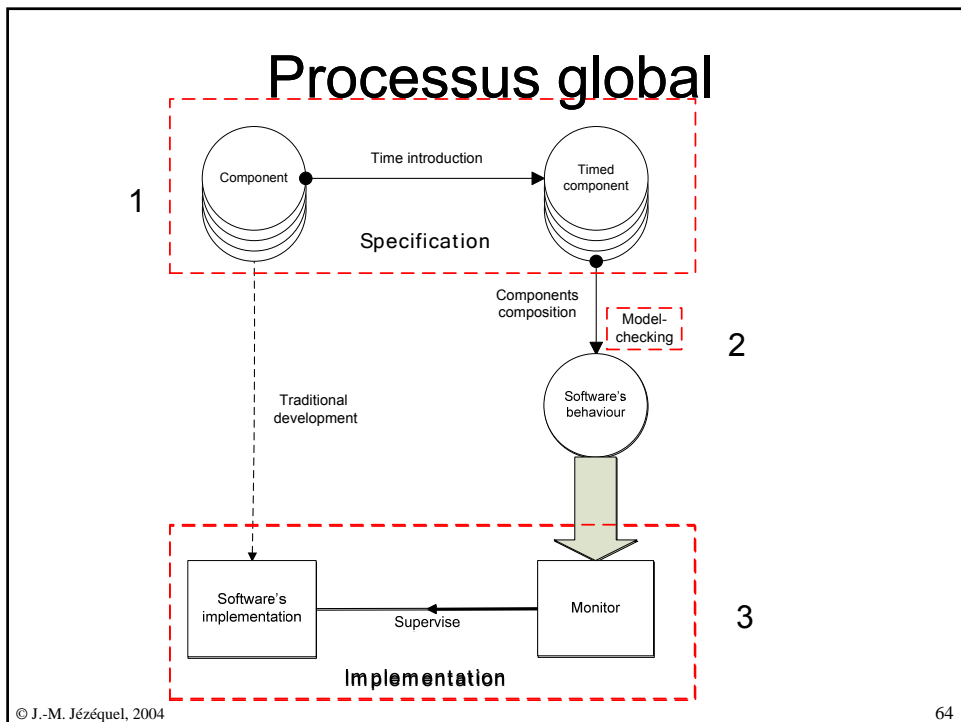
Process overview



Composition checking



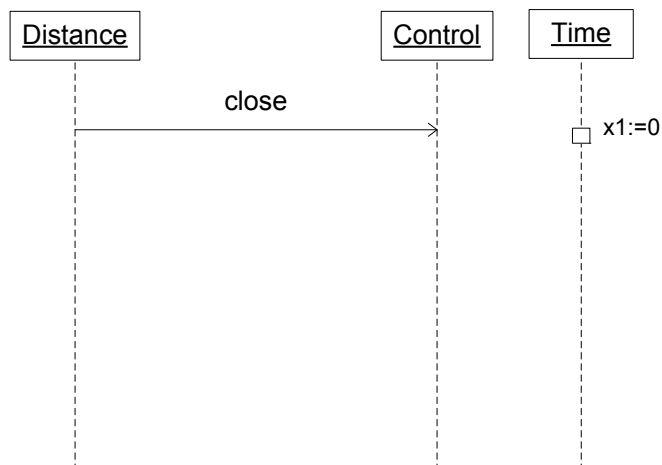
Processus global



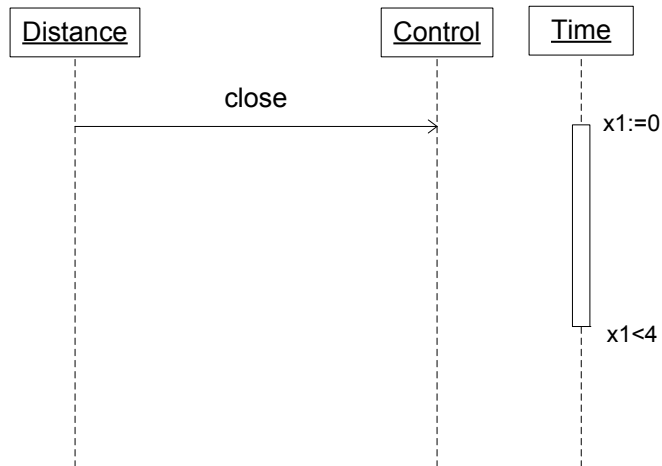
Generation of QoS monitor

- Checking properties during the execution
- Notification of violations
- Use of the specification's result: the expected behaviour
- The monitor is developed on a different branch than the traditional development

QoS Violation



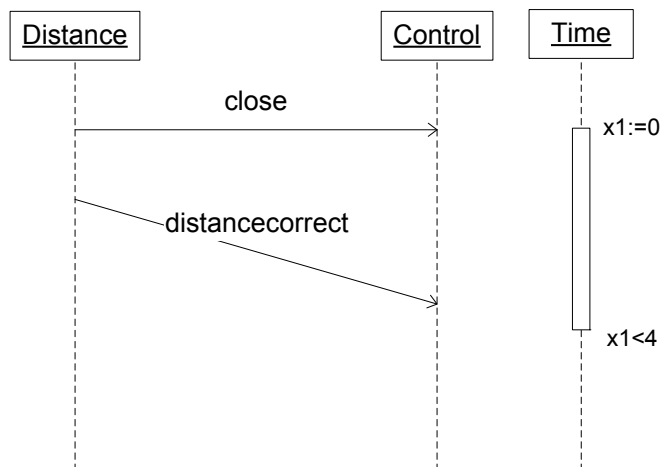
QoS Violation



© J.-M. Jézéquel, 2004

67

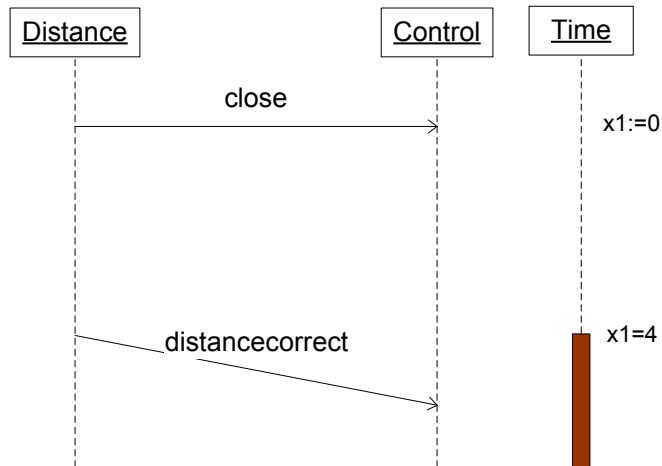
QoS Violation



© J.-M. Jézéquel, 2004

68

QoS Violation



© J.-M. Jézéquel, 2004

69

Transformation

- Automatic generation process:
 - Less errors due to manual coding
 - Automatic → easy redo
- 2 algorithms
 - Discrete time
 - Dense time

© J.-M. Jézéquel, 2004

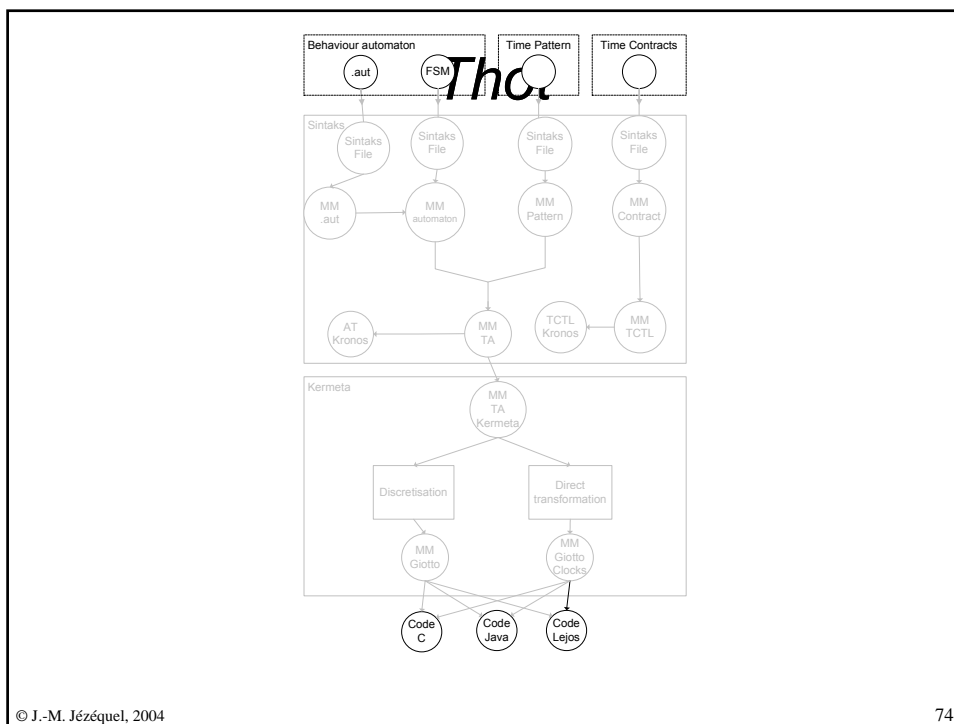
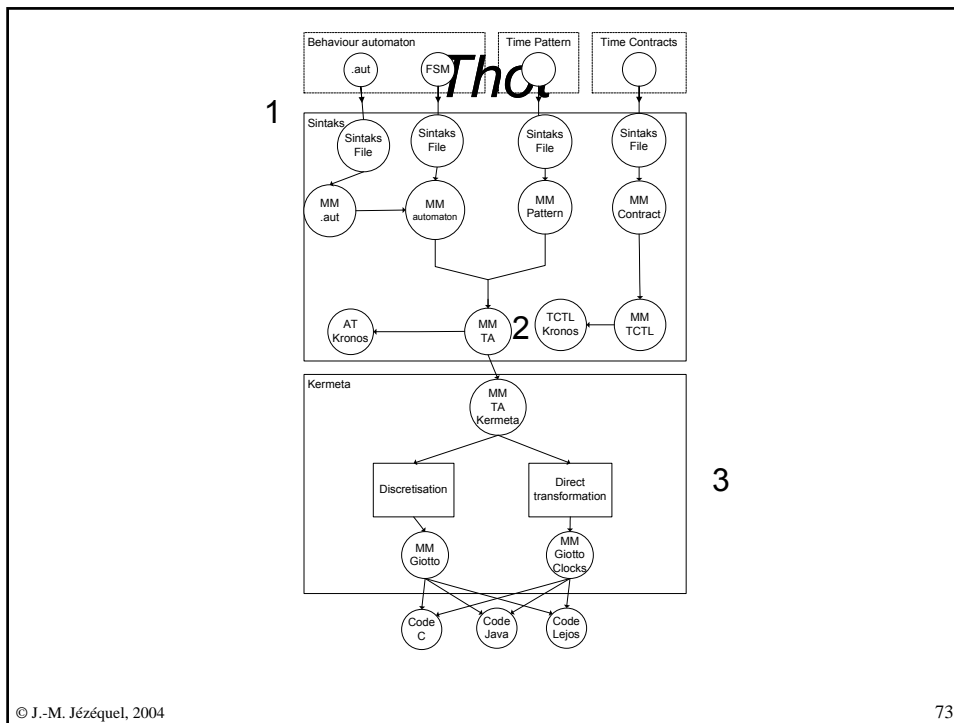
70

Giotto [Heinzinger-Horowitz]

- Programming language
 - For implementing control systems
 - On distributed sensors, actuators, CPUs, and networks
- Based on principles:
 - Time-triggered, periodic tasks
 - Time-triggered mode switches
- Concepts :
 - Ports : interactions with environment
 - Tasks : link with functions
 - Drivers : ports update
 - Modes : 'program' with period, time structure

The tool *Thot*

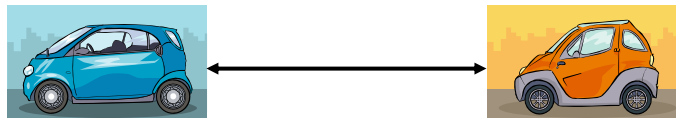
- MDE
- Kermeta
 - Creation of models
 - Model transformations
- Sintaks
 - Text to model



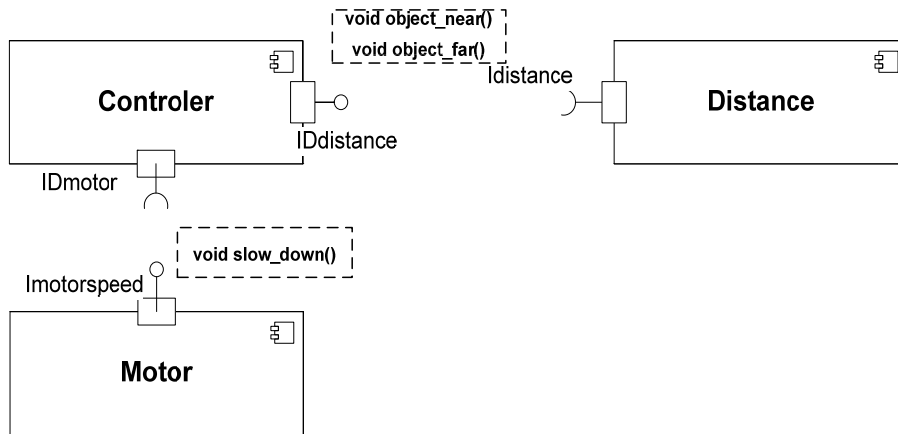
Case Study

- Using the approach on a concrete example
- Made with Lego Mindstorms

Example: Safety distance



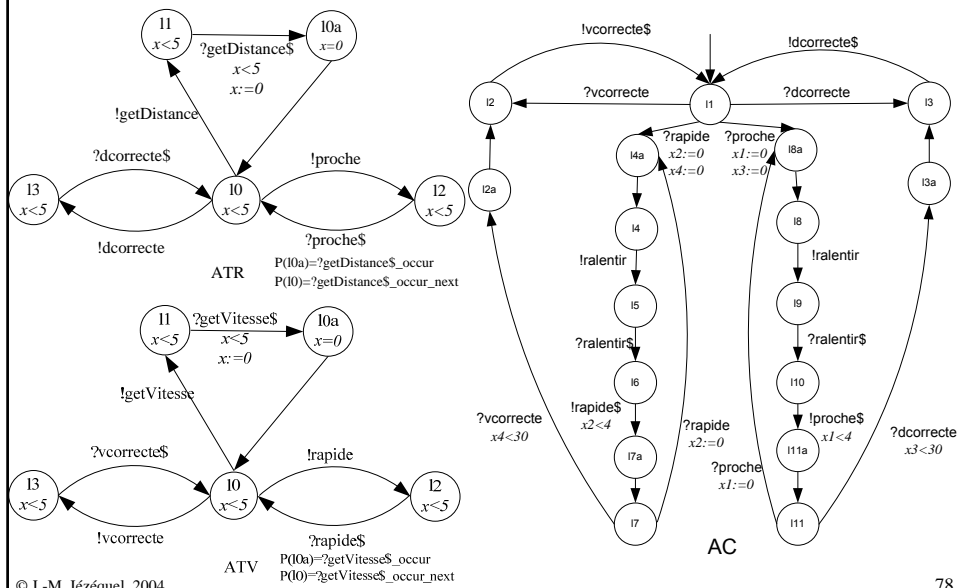
Example



© J.-M. Jézéquel, 2004

77

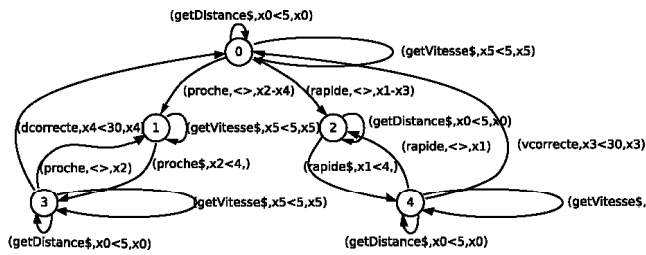
Timed automata



© J.-M. Jézéquel, 2004

78

Monitor generation

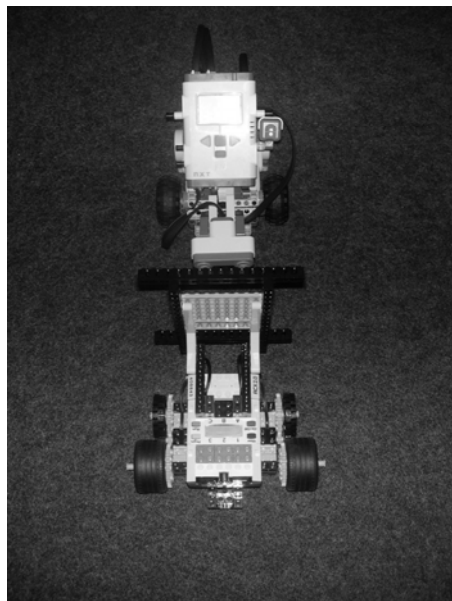


```

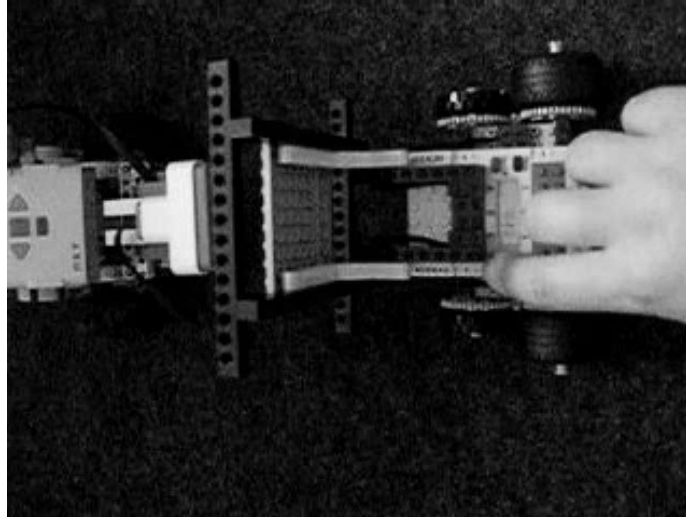
start 0 {
  mode 0() period 1000 {
    exitfreq 1 do 0(driver_getDistance$0);
    exitfreq 1 do 0(vdriver_getDistance$5);
    exitfreq 1 do 1(driver_proche$24);
    exitfreq 1 do 0(driver_getVitesse$5);
    exitfreq 1 do 0(vdriver_getVitesse$5);
    exitfreq 1 do 2(driver_rapide$13);
    taskfreq 1 do In();
  }
  mode 1() period 1000 {
    exitfreq 1 do 3(driver_proche$);
    exitfreq 1 do 3(vdriver_proche$);
    exitfreq 1 do 1(vdriver_getVitesse$5);
    exitfreq 1 do 1(driver_getVitesse$5);
    taskfreq 1 do In();
  }
  mode 2() period 1000 {
    exitfreq 1 do 4(driver_rapide$);
    exitfreq 1 do 4(vdriver_rapide$);
    exitfreq 1 do 2(vdriver_getDistance$0);
    exitfreq 1 do 2(vdriver_getDistance$0);
    taskfreq 1 do In();
  }
  mode 3() period 1000 {
    exitfreq 1 do 3(driver_getDistance$0);
    exitfreq 1 do 3(vdriver_getDistance$5);
    exitfreq 1 do 1(driver_proche$2);
    exitfreq 1 do 3(driver_getVitesse$5);
    exitfreq 1 do 3(vdriver_getVitesse$5);
    exitfreq 1 do 0(driver_dcorrecte$4);
    exitfreq 1 do 0(vdriver_dcorrecte$);
    taskfreq 1 do In();
  }
  mode 4() period 1000 {
    exitfreq 1 do 4(driver_getDistance$0);
    exitfreq 1 do 4(vdriver_getDistance$5);
    exitfreq 1 do 0(vdriver_vcorrecte$3);
    exitfreq 1 do 0(vdriver_vcorrecte$);
    exitfreq 1 do 4(vdriver_getVitesse$5);
    exitfreq 1 do 2(vdriver_rapide$1);
    taskfreq 1 do In();
  }
}

```

The robots



Execution



© J.-M. Jézéquel, 2004

81

Conclusion

- Method for QoS of time
 - Introduction of time
 - Formal semantic
 - Generation of a QoS monitor
- Separation of concerns
- Development of a tool *Thot* (Kermeta and Sintaks)

© J.-M. Jézéquel, 2004

82

Beyond atomic contracts

- **Contracts can provide *trust***
 - *But you cannot (completely) hide the platform*
 - abstraction ≠ hiding
- **Components have offered *and* required interfaces**
 - need to express dependencies between interfaces
- **Component contracted interfaces:**
 - Implies dependencies between offered and required contracts

© J.-M. Jézéquel, 2004

83

Component contracts

- **Component Based Systems are not layers of functionalities**
 - networks of interdependent pieces
- ***Provided* but also *required* contracts**
 - Engagements valid only if *clients and providers* observe their own ones
- **Most offered contracts explicitly depend upon required ones**
 - E.g. response time depends on platform spec
 - And even for objects, this can happen (callback)

© J.-M. Jézéquel, 2004

84

Examples of contract dependencies in the GPS

- The *TimeOutContract* on the **LocationComputer** depends on *TimeOutContracts* from the active **Decoders**
- The *TimeOutContract* on the **Decoder** depends on a *ReceptionQuality* contract on the **Receiver**
 - Monitoring the quality of the reception of satellite data
 - Known at runtime only in this case

Contract space

- A component actually offers a *range of contracts*
 - One contract will be enforced (hopefully)
 - Depending on the obtained required contracts
 - At binding time or at run-time
- Many possible ways to compute:
 - Logical deduction
 - Functionally dependent parameters
 - ...

QoSCL: a QoS-aware component metamodel*

► OUTLINE

- ❑ Includes QML concepts [Frolund, Koistinen - 98]
 - Dimension, Contract, ContractType
- ❑ Includes and extends QoS dependency relationships [Reussner, Schmidt - 01]
- ❑ Extends the UML2 metamodel

► MOTIVATION

- ❑ Weaving QoS monitors (@ run time)
 - QoS monitor, contracts (re-)negotiation
- ❑ Compositional reasoning about QoS properties (@ design/binding time)
 - Performance evaluation, dimensioning

Specifications of QoS properties and contracts

- ❑ Dimensions are Operations
 - Formal parameters
 - Types are totally ordered sets
- ❑ Dependencies and quality requirements are pre/post conditions
 - Dependency relationships are equations or rules
 - Provided/required quality levels are constraints

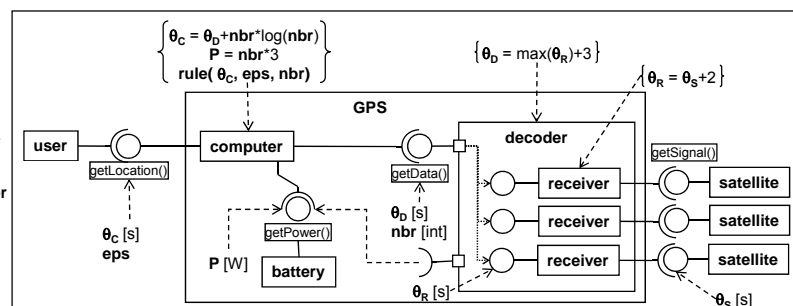
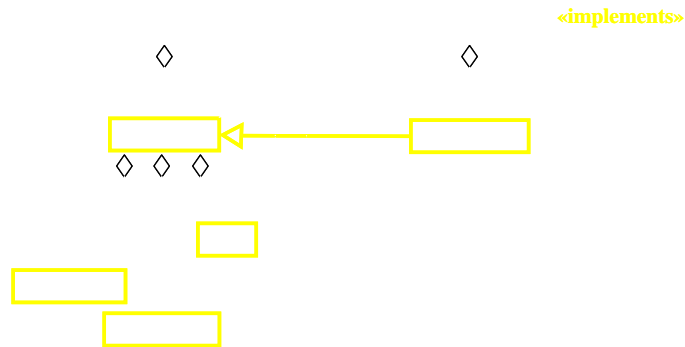


Fig. 1- Specifications of QoS properties for a GPS component-based model

The QoSCL metamodel



© J.-M. Jézéquel, 2004

89

Management of QoS properties & contracts

□ CLP(R) is a general framework

- QoS aspect of a single component is a system of constraints
- Connection of two components implies:
 - ⇒ the conjunction of their QoS constraints (global system)
 - ⇒ the unification between the required and provided QoS properties

□ Model Transformation

- Translation from QoSCL elements into CLP(R) predicates

?- **decoder**([ThetaD, Nbr, ThetaS]) :-
 member(Nbr, [3,5,12]), ThetaD \$>= 0, ThetaS \$>= 0, } ← **Quality levels**
 ThetaD \$= ThetaR + 3,
 receiver([ThetaR, ThetaS]). } ← **Dependency**

?- **gps**([ThetaC, Eps, ThetaS]) :-
 computer([ThetaC, Eps, P, ThetaD, Nbr]), } ← **Composition**
 decoder([ThetaD, Nbr, ThetaS]), } ← **Connections**
 battery([P]). }

© J.-M. Jézéquel, 2004

90

Prediction of QoS behavior

□ Performance evaluation and dimensioning

➤ QoS levels of the GPS component only

?- **gps**([ThetaC, Eps, ThetaS]).
 ThetaC = {6.4313 .. 25.0},
 Eps = medium,
 ThetaD = {0.0 .. 18.5686};
 (more solutions...)

ThetaC	Eps	ThetaS
[25 .. ∞]	low	[18.56 .. ∞]
[6.43 .. 30]	medium	[0 .. 21.5]
[8.49 .. 24]	high	[0 .. 15.5]

Tab. 1- QoS levels for the GPS component only

➤ QoS levels of the GPS component connected to satellite

?- **gps**([ThetaC, _, ThetaS]),
satellite([ThetaS]).
 ThetaC = {21.4313 .. 25.00},
 ThetaS = {15.00 .. 18.5686};
 (more solutions...)

ThetaC	Eps	ThetaS
[25 .. 38.49]	low	[18.56 .. 30]
[21.43 .. 30]	medium	[15 .. 21.5]
[23.5 .. 24]	high	[15 .. 15.5]

Tab. 2- QoS levels after the connection

Conclusion on QoSCL

□ Contribution of CLP(R) in QoS reasoning

- Framework to handle QoS properties and contracts
- Check the QoS (in-)validity of assemblies
- Predict possible quality levels

□ Limitation

- CLP(R) gets the following results:
 - ⇒ the existence of solutions (or not)
 - ⇒ and the closest approximated set of solution

$$\begin{aligned}
 &?- X * X + Y * Y \leq 1. \\
 &X = \{-1.0 .. 1.0\}, \\
 &Y = \{-1.0 .. 1.0\}.
 \end{aligned}$$

□ Perspective

- Use CLP(R) results as QoS oracles in test cases (trusted components)
- Taking into account the dynamic behavior in the QoS reasoning

Outline



- Interest & Challenges of RTE Components
 - The Software Engineering Context
- Assembling Components: the Dangers
 - Towards trusted components
- Components and Extra-Functional Contracts
 - Reasoning about component & contract composition
 - *Weaving contract monitoring*
- Wrap up & Perspectives

© J.-M. Jézéquel, 2004

93

Contract Management

- Contract Description
- Contract Subscription, Termination
- Contract Checking
 - static/dynamic, sequential/concurrent/distributed...
 - Level of Service actually provided
- Dealing with Contract Violations
 - ignore, reject, wait, negotiate ...
- Model Transformations Needed
 - To go from PIM to PSM

Crosscutting Concerns!

© J.-M. Jézéquel, 2004

94

Weave contract management

- **Problem: it depends on the semantics of each contract type**
 - QML does not capture the semantics
 - Sometimes quite complicated
 - » E.g. bounded throughput variation implies non-instantaneous monitoring and the collecting of statistics
 - » May heavily depends on the platform!
- **There exist known solutions to these problems**
 - Apply design patterns?
 - Weave design pattern applications into the PSM model

Outline



- **Interest & Challenges of RTE Components**
 - The Software Engineering Context
- **Assembling Components: the Dangers**
 - Towards trusted components
- **Components and Extra-Functional Contracts**
 - Reasoning about component & contract composition
 - Weaving contract monitoring
- ***Wrap up & Perspectives***

Contract Aware Components: Summary



- Components must have explicitly defined contracts
 - Four levels of contracts
 - » Incl. Contracts to declaratively express non-functional aspects
 - Dependencies between contracts
- Modeling in UML based on e.g. QML
 - Reasoning on models with components & contracts
 - » Bottom-up or top-down
- Monitoring of Contracts is
 - Complex, Cross-cutting, Platform dependant => AOSD

© J.-M. Jézéquel, 2004

97

Perspective

- Component-oriented formalisms
 - with rigid semantics (\Rightarrow gives trustworthiness),
 - More than one semantic domain needed (aspect weaving)
 - well integrated in design cycle (\Rightarrow UML),
 - well supported by tools (\Rightarrow industrial applicability),
 - enabling design-by-contract (\Rightarrow for extra-funtional/QoS properties).

© J.-M. Jézéquel, 2004

98

Perspective



- **Required/Offered QoS contracts**
 - with stochastics, and
 - providing metrics in terms of stochastic models.
- **Seamless QoS design process requires**
 - not a shallow annotational extension of the UML,
 - but a deep semantical embedding.
 - and work on component-based analysis.
- **Industrial-strength tool support.**
 - Performance evaluation and dimensioning
 - simulation
 - verification (model-checking)
 - Testing

© J.-M. Jézéquel, 2004

99

References

- **Components Models**
 - .NET www.microsoft.com/net
 - CCM corbaweb.lifl.fr/OpenCCM/CCM.html
 - EJB: java.sun.com/products/ejb
 - Fractal: fractal.objectweb.org
- **QoS aspect weaving**
 - The IST-QCCS project <http://www.qccs.org>
- **Component-based Design and Development**
 - The IST-ARTIST project <http://www.artist-embedded.org>
- **IST IP project SPEEDS: SPEculative and Exploratory Design in Systems Engineering.**
 - <http://www.speeds.eu.com/>

© J.-M. Jézéquel, 2004

100