Methodology for the tests of a distributed multiagent manufacturing system

Antonio García Domínguez

December 10th, 2013
Background

Convergence of two research lines

- TEP-027 Materials and Manufacturing Engineering and Technology group: manufacturing systems
- TIC-025 UCASE Software Engineering group: testing of Web Services and model-driven engineering
Outline

1. Introduction
2. SODM+T: extension of SODM for performance testing
3. Generation of performance test artefacts
4. Case study: the Keraben tile manufacturing plant
5. Conclusions and future lines of work
Outline

1 Introduction
   - Overall context
   - Holonic manufacturing systems
   - Service-oriented systems

2 SODM+T: extension of SODM for performance testing

3 Generation of performance test artefacts

4 Case study: the Keraben tile manufacturing plant

5 Conclusions and future lines of work
Context: ideal situation

- Raw material suppliers
- Manufacturer
- Distributor
Context: real situation

How to handle this complexity?
Tackling complexity in manufacturing systems

Enterprise architectures
- Provide a structure for describing the enterprise
- GIM, PERA, CIMOSA, GERAM, ISA-95...

Manufacturing process notations
- Indicate steps to be performed and flows between them
- IDEF, PSL, VSM, BPMN...

Process control paradigms
- Coordinate work between participants
- Centralised, hierarchical, heterarchical...
Process control paradigms

Centralised
Process control paradigms

Centralised

Hierarchical

distribute
Process control paradigms

Centralised

Hierarchical

distribute

decentralise

Heterarchical
Process control paradigms

Centralised

Hierarchical

Heterarchical

Problem: flat structures are hard to coordinate.
Holonic manufacturing systems

What is a “holon”?

- Coined by Koestler (1967)
- A whole (hol-) and a part (-on)
Applications of HMS

Popular areas
- Supply chains (Ulieru and Cobzaru, 2005)
- Process control: FABMAS (Mönch et al., 2006), ANEMONA (Giret Boggino, 2005)...

Common implementation of a holon: “intelligent agent”
- A piece of software that pursues a certain goal while collaborating with others
- Besides implementing holons, agents have been used for enterprise integration (e.g. Lea et al., 2005 or Nahm and Ishikawa, 2005) and many other applications
Intelligent agents: reasons for limited adoption?

Observations by (Monostori, 2006) and (Leitão, 2009)

- Agents are flexible and powerful
- However, they have not been widely accepted in industry

Some possible reasons

- Higher initial costs
- Hard to integrate with legacy systems and physical devices
- Missing industrial-strength software tooling
- Emergent behaviour complicates offering guarantees

Proposal

Not all holons or manufacturing agents need to be implemented as intelligent agents.
Alternative implementation of holons as services

**Services and Web Services**

- A service is a reusable business task which is exposed through machine-oriented interfaces
- A Web Service is a particular implementation through Web standards (e.g. HTTP, SOAP and WSDL)

**Holons: intelligent agents, or Web Services?**

- Intelligent agents focus on proactiveness but current implementations are complex and hinder integration
- Web Services focus on interoperability and are simpler to implement: proactiveness can be added later
Service-oriented architecture

Definition
A way to organise information systems as catalogues of services.

Creating an SOA is a non-trivial task
- Can impact the entire organisation
- Development costs are initially higher
- Requires both business and technology expertise

Methodologies in the literature
- Textual descriptions: Erl, Papazoglou
- Model-driven (stepwise refinement of abstractions):
  - Specific steps: GAMBUSE
  - Component-based: Stojanović
  - Comprehensive: IBM SOMA, SODM, BPSOM, Hoyer
Software quality is especially important in SOAs

- Increased reuse places higher demands on the reliability of the service-based holons: a failure can have a much larger impact
- However, existing model-driven methodologies for SOA do not assist users with testing their services

Overall goal for this Thesis

Define and implement a model-driven SOA methodology that assists in developing and testing the information systems of holonic manufacturing enterprises.
Existing methodologies lack testing facilities

Software quality is especially important in SOAs
- Increased reuse places higher demands on the reliability of the service-based holons: a failure can have a much larger impact
- However, existing model-driven methodologies for SOA do not assist users with testing their services

Overall goal for this Thesis
Define and implement a model-driven SOA methodology that assists in developing and testing the information systems of holonic manufacturing enterprises.
Outline

1 Introduction

2 SODM+T: extension of SODM for performance testing
   - Selection of the base methodology: SODM
   - Changes from SODM to SODM+T
   - Algorithms for early performance requirement design
   - Evaluation of the algorithms

3 Generation of performance test artefacts

4 Case study: the Keraben tile manufacturing plant

5 Conclusions and future lines of work
Selection of the base methodology

Candidates

Stojanović, IBM SOMA, SODM, BPSOM and Hoyer.

Requirements

1. Extensible to the entire company
2. Provides (partial) automation through model transformations
3. Publicly available, ideally with a reference implementation
4. Well-known notations and simple models to reduce costs

Results: SODM was selected

- Stojanović: no automation
- IBM SOMA: high cost, proprietary
- Hoyer: too focused on technical aspects
- BPSOM: requires costly tools due to SoaML
Initially proposed extensions: performance tests
Specifying how Web Service-based holons should perform with performance annotations
Initially proposed extensions: performance tests
Specifying how Web Service-based holons should perform with performance annotations

Evaluate order

Close order

Create shipping order

Create invoice

Pay

«sppc»
concurrentUsers = 5
timeLimit = 1

«sapc»
concurrentUsers = 5
timeLimit = 0.2
manual = true

«sapc»
concurrentUsers = 5
timeLimit = 0.1
manual = true

«sapc»
concurrentUsers = 4
timeLimit = 0.7
manual = false

«sapc»
concurrentUsers = 4
timeLimit = 0.35
manual = false

«sapc»
concurrentUsers = 4
timeLimit = 0.35
manual = false
Initially proposed extensions: performance tests
Specifying how Web Service-based holons should perform with performance annotations

- «sppc»
  - concurrentUsers = 5
  - timeLimit = 0.2
  - manual = true

- «sapc»
  - concurrentUsers = 5
  - timeLimit = 0.1
  - manual = false

- «sapc»
  - concurrentUsers = 4
  - timeLimit = 0.35
  - manual = false

- «sapc»
  - concurrentUsers = 4
  - timeLimit = 0.7
  - manual = false

- Evaluate order
- Close order
- Create shipping order
- Create invoice
- Pay

Evaluate order → Close order
Evaluate order → Create shipping order
Close order → Pay
Create shipping order → Pay
Create invoice → Pay
Create invoice → Pay
Initially proposed extensions: performance tests
Specifying how Web Service-based holons should perform with performance annotations

- **«sapc»**
  - concurrentUsers = 5
  - timeLimit = 0.2
  - manual = true

- **«sppc»**
  - concurrentUsers = 5
  - timeLimit = 1

- **«sapc»**
  - concurrentUsers = 5
  - timeLimit = 0.1
  - manual = true

- **«sapc»**
  - concurrentUsers = 4
  - timeLimit = 0.7
  - manual = false

- **«sapc»**
  - concurrentUsers = 4
  - timeLimit = 0.35
  - manual = false

- **«sapc»**
  - concurrentUsers = 4
  - timeLimit = 0.35
  - manual = false
Initially proposed extensions: performance tests
Specifying how Web Service-based holons should perform with performance annotations

Evaluate order
- [rejected] (p=0.2)
- [else] (p=0.8)

Create shipping order
- Create invoice
- Pay

Close order

«sapc»
concurrentUsers = 5
timeLimit = 0.2
manual = true

«sppc»
concurrentUsers = 5
timeLimit = 1

«sapc»
concurrentUsers = 5
timeLimit = 0.1
manual = true

«sapc»
concurrentUsers = 4
timeLimit = 0.35
manual = false

«sapc»
concurrentUsers = 4
timeLimit = 0.7
manual = false

«sapc»
concurrentUsers = 4
timeLimit = 0.35
manual = false
Initially proposed extensions: functional tests
Specifying what Web Service-based holons should do with service contracts

- Evaluate order
- Close order
- [rejected]
- [else]
- Order
- Create shipping order
- «WS» Create invoice
- Pay
Initially proposed extensions: functional tests
Specifying what Web Service-based holons should do with service contracts

- «precondition» order.open = true
- «postcondition» order.open = false

- «localPrecondition»
  order.accepted = true
  and order.open = true
  and order.entries.count() > 0
  and not Invoice.allInstances()
  → exists(i | i.order.id = order.id)

- «localPostcondition»
  invoice.oclIsNew()
  and invoice.entries.count() = order.articles.count()
  and invoice.entries forall (e | e.price = e.article.currentPrice)
  and invoice.total = invoice.entries
  → collect(e | e.price * (1 + e.tax) * e.qty) → sum()
Extended metamodels and editors in SODM+T

**Elements brought back from UML (originally missing in SODM)**
- Structured activity nodes (can contain other nodes)
- Merge nodes

**New elements for performance testing**
- Control flows could now have a *condition* and a *probability*
- New nodes: global constraints and local annotations

**New model editors based on Epsilon**
- Created using a model-driven approach (Eugenia)
- Models are automatically validated (EVL)
- Transformations from SP to SC models are automated (Flock)
- EVL rules tested using the new EUnit framework
Antonio García Domínguez  Methodology for the tests of a distributed multiagent manufacturing system
Antonio García Domínguez

Methodology for the tests of a distributed multiagent manufacturing system

Screenshot of the new Epsilon EUnit testing framework

- 1 difference in resource compare2311004039551262013.model
  - 1 change(s) in model
    - 1 change(s) in Graph
      - 1 change(s) in Node z2

Attribute name: EString in Node t2 has changed from z2 to t2

```
Obtained model

- Node t1
- Edge
- Node t2

Expected model

- Node t1
- Edge
- Node z2
```

Differences Properties

Differences Properties

 Failures: 1
Errors: 0

Runs: 5/5
Performance inference algorithms

General requirements
- Users provide global requirements and a description of their partial knowledge about the desired performance
- The algorithms “fill in the blanks” and define performance requirements for every action in the model

Developed algorithms
- A throughput inference algorithm that computes how many requests each WS will need to handle
- Three time limit inference algorithms that compute in how many seconds the WS needs to reply
Throughput inference

**General concept**

Requests propagate from the initial node to the final nodes.

- $T(I) =$ global requirement
- $T(n) =$ minimum of $T(e_i)$
- $T(n) =$ sum of $T(e_i)$
- $T(e) = \mathcal{P}(e)T(a)$
- $T(b) = T(e)$
Time limit inference

Basic concepts

- Nodes have minimum times and weights
- Slack of a path: difference between global time limit and its total minimum time

Optimisation-based approach for distributing the slack

Maximise the available time over all paths while ensuring that:

1. Minimum times and global time limit are upheld
2. Every node with non-zero weight gets some slack
3. Slack should be distributed evenly and proportionally to weight

Alternative graph-based algorithms

- Naive algorithm (equivalent to above approach, simpler)
- Incremental algorithm (more efficient)
### Evaluation of the algorithms: research questions

**Questions**

1. Do the algorithms compute the desired results?
2. What are their limitations?
3. How is the performance affected by:
   1. the shape and size of the model?
   2. the proportion of annotated nodes?
   3. the values used in the annotations?

**Approaches**

- Theoretical analysis of the algorithms
- Empirical studies with EUnit and a custom Eclipse plug-in for performance comparisons
Input model limitations and testing approach

Validation of the algorithms through testing

- Hand-crafted representative cases, with known results
- Large sets of automatically generated models, checking that the time limit inference algorithms produce equivalent results (within $\pm 0.1\%$)

Identified limitations

- Cycles are not allowed (but bounded repetition is available)
- Multiple instances of the same task are treated as different tasks (but the strictest constraint can be taken)
Performance of the throughput inference algorithm

Empirical results confirm the quadratic theoretical bound
Performance of the time limit inference algorithms
The incremental algorithm shows much better performance

![Graph showing performance comparison](image_url)

- **Opt.-based**
- **Naive**
- **Incremental**

Time (s)

Size (actions)
Performance of the time limit inference algorithms
The incremental algorithm shows much better performance
Performance of the time limit inference algorithms
The incremental algorithm shows much better performance

- Opt.-based
- Naive
- Incremental

![Graph showing time vs. size for different algorithms](image)
Performance impact from annotation density
More annotations can imply a higher cost, but it depends on the shape

![Graph showing performance impact from annotation density](image)
Performance impact from annotation density
More annotations can imply a higher cost, but it depends on the shape
Performance impact from annotation density

More annotations can imply a higher cost, but it depends on the shape
1. Introduction

2. SODM+T: extension of SODM for performance testing

3. Generation of performance test artefacts
   - Overall approach for generation of performance test artefacts
   - Applications: Java and WSDL

4. Case study: the Keraben tile manufacturing plant

5. Conclusions and future lines of work
Outline of the proposed generic approach

- Code
- Performance model
- Weaving model
  - Test artefacts (code + scripts + textual DSL-based models)
- Model discovery
- Design/impl. model
- M2T transformation
Outline of the proposed generic approach

- Code
- Performance model
- Weaving model
- Test artefacts (code + scripts + textual DSL-based models)
- Model discovery
- Design/impl. model
- M2T transformation
Outline of the proposed generic approach

- Code
- Performance model
- Weaving model
- Model discovery
- Design/impl. model
- M2T transformation
- Test artefacts (code + scripts + textual DSL-based models)
Outline of the proposed generic approach

- Code
- Performance model
- Weaving model
- Model discovery
- Design/impl. model
- M2T transformation
- Test artefacts (code + scripts + textual DSL-based models)
Repurposing Java unit tests as performance tests
Many WS are developed in Java (Apache Axis, Apache CXF) and tested with JUnit

```
public class MyUnitTests {
    @Test
    public void runRejected() {
        /* ... test code ... */
    }

    @Test
    public void runAccepted() {
        /* ... test code ... */
    }
}
```
Repurposing Java unit tests as performance tests
Many WS are developed in Java (Apache Axis, Apache CXF) and tested with JUnit

MoDisco is used to extract a model from the Java code.
Repurposing Java unit tests as performance tests
Many WS are developed in Java (Apache Axis, Apache CXF) and tested with JUnit

The weaving model is created using Epsilon ModeLink.
Repurposing Java unit tests as performance tests
Many WS are developed in Java (Apache Axis, Apache CXF) and tested with JUnit

Extraction  Weaving  Generation

```java
@RunWith(ContiPerfSuiteRunner.class)
@SuiteClasses(MyUnitTests.class)
@PerfTest(invocations=100, threads=10)
@Required(max=1000)
public class InferredLoadTest {}
```

The new JUnit test suite uses the ContiPerf performance testing library to generate less code.
Generating new tests from WSDL documents

**Extraction**
Use model discovery with the new ServiceAnalyzer tool to produce message catalogues from the WSDL descriptions.

**Weaving**
Link UML actions with catalogue operations (Epsilon Modelink).

**Generated artefacts**
- Automation support (Eclipse, Maven, Jython)
- Customisable templates for input data and messages (TestSpec, Velocity)
Outline

1. Introduction
2. SODM+T: extension of SODM for performance testing
3. Generation of performance test artefacts
4. Case study: the Keraben tile manufacturing plant
   - Description of the enterprise
   - Modelling with SODM+T
   - Partial implementation of a service-based holon
   - Performance testing
5. Conclusions and future lines of work
Description of the Keraben Group

Keraben

Size

- > 250 000 m²
- > 1 000 employees
Description of the Keraben Group

Size

- > 250,000 m²
- > 1,000 employees
Description of the Keraben Group

Size

- > 250,000 m²
- > 1,000 employees
Description of the Keraben Group

Keraben

Cerámica Gres

Size

- \( > \ 250 \ 000 \ m^2 \)
- \( > \ 1 \ 000 \) employees
Description of the tile manufacturing process

1. Raw materials
2. Grinding and atomisation
3. Pressing
4. Drying
5. Glazing
6. Packaging
7. Classification
8. Grinding
9. Firing
Description of the tile manufacturing process

1. Raw materials
2. Grinding and atomisation
3. Pressing
4. Drying
5. Glazing
6. Packaging
7. Classification
8. Grinding
9. Firing
Modelling scope

Main focus
Manufacturing scheduling (make-to-order and make-to-stock).

Reasons for the present focus
- Ideal for the service-based holons that focus on interoperability
- One of the links between levels 3 and 4 of the ISA-95 activity hierarchy (manufacturing operations and business planning)
CIM: Gordijn value model
Specifies the value exchanges between business participants (holons)
CIM: business process model
Specifies the business model of the enterprise under study
CIM: business services list
Collects the business services required by the enterprise

**Business services for “Customer”**
- Order submission
- Order status reporting
- Order issue notification and response

**Business services for “Sales and Marketing”**
- Estimated demand submission
- Production status reporting
- Production issue notification and response
PIM: use case model
Lists the ways in which the system will be used

Customer

<<CLASSIFIER>>
SOA Keraben

<<BS>>
Order submission

<<BS>>
Order status reporting

<<BS>>
Order issue notification

<<BS>>
Estimated demand submission

<<BS>>
Production issue notification

<<BS>>
Production status reporting

Sales Department
PIM: extended use case model for “Order submission”
Decomposes the use case into easier to implement steps
PIM: service process model for "Order submission"
Arranges the EUC steps into a certain flow
PIM: service composition model for “Order submission”
Splits work among the participant actors or holons

«gaScenario, gaAnalysisContext»
Order Submission Composition

<Customer> Provide production specs

 structured> Review order feasibility

(B2B) Check product availability

[discontinued or missing]
(B2B) Report bad product

[else]
(MES) Estimate prod. dates

[unfeasible or unacceptable deadline]
(B2B) Report cancelled order

[else]

(Customer) Provide shipping address

(Logistics) Check shipping info

[else]

(B2B) Compute order total

(Customer) Review order total

[order confirmed]

(Customer) Provide payment details

(Finances) Perform Payment

[failed]

[succeeded]

(B2B) Confirm order
Actions marked with «WS» to be implemented as Web Services

- (B2B) Check product availability
- (MES) Estimate production dates
- (Logistics) Check shipping information
- (Finances) Perform payment
PSM: performance annotations for the ESC model

Annotation of the models
- Global time limit: 5 s handling 5 requests per second
- Some nodes received minimum times and weights
- Conditional edges received probability estimations

Generation of the requirements
- The algorithms were run to annotate the models
- For example, “Estimate production dates” had to reply within 2.28 s while handling 4.5 requests per second
PSM: Web Service interface model

Implementation

UML class diagram annotated with a Papyrus UML profile.

Contents of the model for “Estimate production dates”

- 1 service (“SchedulerService”) + endpoint/binding/interface
- The interface has 1 operation, “EstimateProductionDates”:
  - Request message takes the ISA-95 product production rule ID and the requested quantity
  - Response message includes the production schedule ID, the earliest start date and the latest end date
  - Error message provides a string with a description
Components for the “MES” service-based holon

**Data model**
- Translation of the ISA-95 object model using Spring Roo
- Adjustments to reduce duplication and improve data integrity
- Detected other issues in the ISA-95 data model:
  - The supply chain is not represented
  - No distinction between make-to-order and make-to-stock

**Internal intelligence**
- Greedy scheduler
- Additional facilities for data entry

**Public interfaces**
- For humans: web interface
- For machines: Web Service exposing the greedy scheduler
## ISA-95 Model Administration Panel

### List all Process Segments

<table>
<thead>
<tr>
<th>Segment Id</th>
<th>Location</th>
<th>Duration</th>
<th>Duration Unit</th>
<th>Parent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG-R-Drying</td>
<td>EQ-AreaProd</td>
<td>120.0000 Minutes</td>
<td>Minutes</td>
<td></td>
<td>Drying the rectified pieces, reducing their water content.</td>
</tr>
<tr>
<td>SG-R-Inspection</td>
<td>EQ-AreaProd</td>
<td>45.0000 Seconds/Pieces</td>
<td>Seconds/Pieces</td>
<td></td>
<td>Inspection of the rectified pieces for cracks and inadequate calibrator.</td>
</tr>
<tr>
<td>SG-R-HRectifying-M</td>
<td>EQ-AreaProd</td>
<td>5.0000 Minutes/Pieces</td>
<td>Minutes/Pieces</td>
<td></td>
<td>Horizontal rectifying on a medium tile.</td>
</tr>
<tr>
<td>SG-R-VRectifying-M</td>
<td>EQ-AreaProd</td>
<td>5.0000 Minutes/Pieces</td>
<td>Minutes/Pieces</td>
<td></td>
<td>Vertical rectifying on the piece.</td>
</tr>
<tr>
<td>SG-R-Setup-LM</td>
<td>EQ-AreaProd</td>
<td>10.0000 Minutes</td>
<td>Minutes</td>
<td></td>
<td>Line setup for standard tile sizes.</td>
</tr>
<tr>
<td>SG-R-Setup-C</td>
<td>EQ-AreaProd</td>
<td>15.0000 Minutes</td>
<td>Minutes</td>
<td></td>
<td>Line setup time for custom sizes.</td>
</tr>
<tr>
<td>SG-R-HRectifying-L</td>
<td>EQ-AreaProd</td>
<td>7.0000 Minutes/Pieces</td>
<td>Minutes/Pieces</td>
<td></td>
<td>Rectifying the horizontal edges of a large tile.</td>
</tr>
<tr>
<td>SG-R-VRectifying-L</td>
<td>EQ-AreaProd</td>
<td>7.0000 Minutes/Pieces</td>
<td>Minutes/Pieces</td>
<td></td>
<td>Rectifying the vertical edges of a large tile.</td>
</tr>
<tr>
<td>SG-R-HRectifying-C</td>
<td>EQ-AreaProd</td>
<td>7.0000 Minutes/Pieces</td>
<td>Minutes/Pieces</td>
<td></td>
<td>Rectifying the horizontal edges of a custom-sized tile.</td>
</tr>
<tr>
<td>SG-R-VRectifying-C</td>
<td>EQ-AreaProd</td>
<td>7.0000 Minutes/Pieces</td>
<td>Minutes/Pieces</td>
<td></td>
<td>Rectifying the vertical edges of a custom-sized tile.</td>
</tr>
</tbody>
</table>
Generation of the performance test

Process

1. Link UML actions to WS operations
2. Generate test artefacts
3. Customise to produce meaningful data

Customised TestSpec document

typedef int (min=1, max=100) TQuantity;
typedef string (values={"PPR−RC2", ...}) ProductProductionRule;
typedef string (values={"Pieces"}) Unit;
typedef tuple (element={
    string, ProductProductionRule, TQuantity, Unit
}) TGenerateSchedule;

TGenerateSchedule generateSchedule;
Results from the performance test: graphs

All Transactions

Performance

Elapsed time (s)

Throughput (tx/s)

Time (s)

transactions/second

mean response time
## Results from the performance test: summary tables

### Transaction Name

<table>
<thead>
<tr>
<th>Transaction Name</th>
<th>Tests Passed</th>
<th>Tests W/ Errors</th>
<th>Pass Rate</th>
<th>Mean Response Time</th>
<th>Response Time Standard Dev.</th>
<th>Mean Response Length</th>
<th>Bytes per Sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>SchedulerImplService-generateSchedule</td>
<td>1,000</td>
<td>0</td>
<td>1.000</td>
<td>193.9 ms</td>
<td>70.54 ms</td>
<td>402.92 bytes</td>
<td>10,221.57</td>
</tr>
</tbody>
</table>

### Transaction Name

<table>
<thead>
<tr>
<th>Transaction Name</th>
<th>Mean Time Resolve Host</th>
<th>Mean Time Establish Connection</th>
<th>Mean Time to First Byte</th>
<th>Under 1 sec</th>
<th>1 to 3 sec</th>
<th>3 to 10 sec</th>
<th>Over 10 sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>SchedulerImplService-generateSchedule</td>
<td>0.02 ms</td>
<td>0.19 ms</td>
<td>192.68 ms</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>
End of the case study

Process

1. Started with high-level descriptions of the business
2. Continued with increasingly detailed processes
3. Annotated processes with performance requirements
4. Implemented the holons and tested them

Summary

SODM+T can cover the analysis and high-level design and also help with testing the WS.
Outline

1. Introduction
2. SODM+T: extension of SODM for performance testing
3. Generation of performance test artefacts
4. Case study: the Keraben tile manufacturing plant
5. Conclusions and future lines of work
   - Results
   - Future lines of work
Contributions: main results (I)

**Define a methodology for developing and testing IS for HMS**
- Selected SODM from available candidates
- Proposed extensions for performance and functional testing
- Extended its metamodel for performance testing
- Revamped the editors, adding validation and transformations

**Adopt techniques for generating and running tests**
- Four algorithms for deriving performance requirements:
  - Validated through manual and random testing
  - Throughput and incremental time limit inference algorithms have good performance
- Two workflows for generating performance test artefacts:
  - Repurposing Java unit tests
  - Creating new tests for WSDL-based Web Services
Contributions: main results (II)

Define rules for implementing a holon

Implemented a service-based holon using current technologies:

- Data model inspired on ISA-95 with improvements
- Greedy scheduler based on ISA-95 concepts
- Human and machine interfaces (web interface and WS)

Implement and validate the methodology

- Performed an end-to-end case study based on Keraben
- Started with descriptions of the environment and business
- Ended with generating performance test artefacts for a WS
Contributions: complementary results

<table>
<thead>
<tr>
<th>Improvements in the Epsilon framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contributions since 2009, direct involvement since 2011</td>
</tr>
<tr>
<td>Over 380 code contributions and 250 forum messages</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other manufacturing contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed broadening the concept of a holon in HMS</td>
</tr>
<tr>
<td>Compared several notations for manufacturing processes</td>
</tr>
</tbody>
</table>
Future lines of work

### Related to manufacturing
- Extend the case study to multiple WS and holons
- Define an improved notation for describing manufacturing processes, simplifying data entry
- Add more autonomy to some holons by integrating other technologies (e.g. Complex Event Processing)

### Related to software engineering
- Provide additional guidelines for minimum times and weights
- Extend models to support functional testing: some work has already been done regarding WSCoL contracts
Thank you for your attention

Source code:
https://neptuno.uca.es/redmine/projects/sodmt
A proposal for generalised software holons

Revised concept

The concept of a holon could be broadened to a reusable piece of software that has the following features:

- can be integrated into other holons while retaining its identity,
- performs a well-defined task,
- provides machine-oriented interfaces through open standards,
- uses communication primitives (e.g. request/response or auctions) across a cooperation domain,
- may include a human-facing interface,
- may interact with a physical control layer and
- may include some degree of intelligence to reason about its work (expert systems, business rules, schedulers, and so on).
Performance impact from annotation values (I)
Larger models make the worst case less common

Observation about the incremental time limit algorithm
- Average case seems to be sub-exponential
- How common is the exponential worst case?
- Approach: sample the space of all fork-join nodes

First study: influence of complexity ($L = 1.5$ s)
- 1-level binary fork-join models: 31.8% in worst case
- 2-level, with 1-level “worst case”: 2.5% in worst case
- 3-level, with 2-level “worst case”: 0.05% in worst case
- Conclusion: worst case becomes less common as models grow
Stricter time limits make the worst case less common.

Sampled 3-level fork-join models by top-level incomparable paths

- $L = 0.50\text{s}$
- $L = 1.50\text{s}$
Nodes (default: minimum time 0 and weight 1)

- Global time limit: 5 s, while handling 5 requests per second
- “Customer” actions: minimum time 0 and weight 0
- “Review order feasibility”: minimum time 0 and weight 5, as generating a schedule can be expensive
- “Estimate production dates” originally had minimum time 0 and weight 3, later revised to minimum time 0.4 s and weight 2

Probability estimations for conditional edges

- 10% of all orders refer to a missing or discontinued product
- 20% of remaining: bad manufacturing deadlines
- 20% of remaining: bad shipping deadlines
- 10% of remaining: cancelled after reviewing the order total
- 5% of remaining: problems during payment
Results from the performance inference algorithms

- “Check product availability”: 0.94 s while handling 5 requests/s.
- “Check shipping information”: 0.44 s, 4 requests/s.
- “Compute order total”: 0.44 s, 3.2 requests/s.
- “Confirm order”: 0.44 s, 2.74 requests/s.
- “Estimate production dates”: 2.28 s, 4.5 requests/s.
- “Perform payment”: 0.44 s, 2.88 requests/s.
- “Report bad product”: 2.28 s, 0.5 requests/s.
- “Report cancelled order”: 1.33 s, 0.8 requests/s.
A. García-Domínguez, I. Medina-Bulo and M. Marcos-Bárcena.

A. García-Domínguez, I. Medina-Bulo and M. Marcos-Bárcena.

A. García-Domínguez, I. Medina-Bulo and M. Marcos-Bárcena.

A. García-Domínguez, I. Medina-Bulo and M. Marcos-Bárcena.

A. García-Domínguez, I. Medina-Bulo and M. Marcos-Bárcena.
Hacia la Integración de Técnicas de Pruebas en Metodologías Dirigidas por Modelos para SOA. Novática, no. 204, March 2010. ISSN 0211-2124.
A. García-Domínguez, I. Medina-Bulo, M. Marcos-Bárcena.
An Approach for Model-Driven Design and Generation of Performance Test Cases with UML and MARTE.

A. García-Domínguez, I. Medina-Bulo and M. Marcos-Bárcena.
Performance Test Case Generation for Java and WSDL-based Web Services from MARTE.

A. García-Domínguez, I. Medina-Bulo and M. Marcos-Bárcena.
An Approach for Performance Test Artefact Generation for Multiple Technologies from MARTE-Annotated Workflows.
Procedia Engineering, no. 63, September 2013. ISSN 1877-7058.

A. García-Domínguez, M. Marcos-Bárcena and I. Medina-Bulo.
A Comparison of BPMN 2.0 with Other Notations for Manufacturing Processes.
Key Engineering Materials, no. 502, February 2012. ISSN 1013-9826.

Proposal of a Methodology for Implementing a Service-Oriented Architecture in Distributed Manufacturing Systems.