



Escuela Superior de
Ingeniería
Programa de Doctorado en
Ingeniería y Arquitectura
Itinerario en Ingeniería de Fabricación



Metodología dirigida por modelos para las pruebas de un sistema distribuido multiagente de fabricación

Autor: Antonio García Domínguez
Directores: Inmaculada Medina Buló, Mariano Marcos Bárcena

Conformidad de los directores

D^a María Inmaculada Medina Bulo, profesora del Departamento de Ingeniería Informática de la Universidad de Cádiz, y D. Mariano Marcos Bárcena, profesor del Departamento de Ingeniería Mecánica y Diseño Industrial, siendo Directores de la Tesis titulada *Metodología dirigida por modelos para las pruebas de un sistema distribuido multiagente de fabricación*, realizada por el doctorando D. Antonio García Domínguez dentro del Programa de Doctorado en Ingeniería y Arquitectura bajo el Itinerario en Ingeniería de Fabricación, para proceder a los trámites conducentes a la presentación y defensa de la tesis doctoral arriba indicada, en aplicación de la Normativa Reguladora de Estudios de Tercer Ciclo de la Universidad de Cádiz, informan que se autoriza la tramitación de la tesis.

Los directores de tesis

Inmaculada Medina Bulo

Mariano Marcos Bárcena

Cádiz, España, a 6 de noviembre de 2013

Agradecimientos

Desearía dar las gracias a todas las personas que han hecho posible esta tesis:

- A mis padres, por su apoyo sacrificado e incondicional y por inculcarme la curiosidad ante todo y el gusto por el trabajo bien hecho.
- A mis hermanas y mis amigos y amigas, por los buenos ratos compartidos y todos los ánimos recibidos.
- A Inma y Mariano, por su ayuda durante esta tesis y su paciencia.
- A Paco, por enseñarme el placer de impartir conocimientos y aportar mi granito de arena.
- A la comunidad de desarrolladores de software de fuentes abiertas, que ha posibilitado el desarrollo de esta tesis con sus sistemas operativos, herramientas, metodologías y conocimientos.

Muchas gracias a todos.

Antonio García Domínguez
Cádiz, España, 6 de noviembre de 2013

Agradecimientos institucionales

Este trabajo fue financiado por la beca de investigación PU-EPIF-FPI-C 2010-065 de la Universidad de Cádiz, por el proyecto MoDSOA (TIN2011-27242) del Programa Nacional de Investigación, Desarrollo e Innovación del Ministerio de Ciencia e Innovación y por el proyecto PR2011-004 del Plan de Promoción de la Investigación de la Universidad de Cádiz.

Resumen

Las presiones del mercado han empujado a las empresas de fabricación a reducir costes a la vez que mejoran sus productos, especializándose en las actividades sobre las que pueden añadir valor y colaborando con especialistas de las otras áreas para el resto. Estos sistemas distribuidos de fabricación conllevan nuevos retos, dado que es difícil integrar los distintos sistemas de información y organizarlos de forma coherente. Esto ha llevado a los investigadores a proponer una variedad de abstracciones, arquitecturas y especificaciones que tratan de atacar esta complejidad. Entre ellas, los sistemas de fabricación holónicos han recibido una atención especial: ven las empresas como redes de *holones*, entidades que a la vez están formados y forman parte de varios otros holones. Hasta ahora, los holones se han implementado para control de fabricación como agentes inteligentes autoconscientes, pero su curva de aprendizaje y las dificultades a la hora de integrarlos con sistemas tradicionales han dificultado su adopción en la industria. Por otro lado, su comportamiento emergente puede que no sea deseable si se necesita que las tareas cumplan ciertas garantías, como ocurren en las relaciones de negocio a negocio o de negocio a cliente y en las operaciones de alto nivel de gestión de planta.

Esta tesis propone una visión más flexible del concepto de holón, permitiendo que se sitúe en un espectro más amplio de niveles de inteligencia, y defiende que sea mejor implementar los holones de negocio como *servicios*, componentes software que pueden ser reutilizados a través de tecnologías estándar desde cualquier parte de la organización. Estos servicios suelen organizarse como catálogos coherentes, conocidos como Arquitecturas Orientadas a Servicios ('Service Oriented Architectures' o SOA). Una iniciativa SOA exitosa puede reportar importantes beneficios, pero no es una tarea trivial. Por este motivo, se han propuesto muchas metodologías SOA en la literatura, pero ninguna de ellas cubre explícitamente la necesidad de probar los servicios. Considerando que la meta de las SOA es incrementar la reutilización del software en la organización, es una carencia importante: tener servicios de alta calidad es crucial para una SOA exitosa.

Por este motivo, el objetivo principal de la presente Tesis es definir una metodología extendida que ayude a los usuarios a probar los servicios que implementan a sus holones de negocio. Tras considerar las opciones disponibles, se tomó la metodología dirigida por modelos SODM como punto de partida y se reescribió en su mayor parte con el framework Epsilon de código abierto, permitiendo a los usuarios que modelen su conocimiento parcial sobre el rendimiento esperado de los servicios. Este conocimiento parcial es aprovechado por varios nuevos algoritmos de inferencia de requisitos de rendimiento, que extraen los requisitos específicos de cada servicio. Aunque el algoritmo de inferencia de peticiones por segundo es sencillo, el algoritmo de inferencia de tiempos límite pasó por numerosas revisiones hasta obtener el nivel deseado de funcionalidad y rendimiento. Tras una primera formulación basada en programación lineal, se reemplazó con un algoritmo sencillo *ad hoc* que recorría el grafo y después con un algoritmo incremental mucho más rápido y avanzado. El algoritmo incremental produce resultados equivalentes y tarda mucho menos, incluso con modelos grandes.

Para sacar más partidos de los modelos, esta Tesis también propone un enfoque general para generar artefactos de prueba para múltiples tecnologías a partir de los modelos anotados por los algoritmos. Para evaluar la viabilidad de este enfoque, se implementó para dos posibles usos: reutilizar pruebas unitarias escritas en Java como pruebas de rendimiento, y generar proyectos completos de prueba de rendimiento usando el framework The Grinder para cualquier Servicio Web que esté descrito usando el estándar Web Services Description Language.

La metodología completa es finalmente aplicada con éxito a un caso de estudio basado en un área de fabricación de losas cerámicas rectificadas de un grupo de empresas español. En este caso de estudio se parte de una descripción de alto nivel del negocio y se termina con la implementación de parte de uno de los holones y la generación de pruebas de rendimiento para uno de sus Servicios Web.

Con su soporte para tanto diseñar como implementar pruebas de rendimiento de los servicios, se puede concluir que SODM+T ayuda a que los usuarios tengan una mayor confianza en sus implementaciones de los holones de negocio observados en sus empresas.

Abstract

Market pressures have pushed manufacturing firms to reduce costs while improving their products by specialising in their main value-adding activities and collaborating with other specialist firms for the others. These distributed manufacturing systems bring new challenges, as it is difficult to integrate their disparate information systems and organise them in a coherent manner, prompting researchers to provide a number of abstractions, architectures and specifications that attempt to tackle this complexity. Among them, holonic manufacturing systems have achieved considerable attention: these view enterprises as networks of *holons*, entities which are at the same time formed by and part of several other holons. So far, holons have been implemented as intelligent self-aware agents for manufacturing control, but their steep learning curve and difficulties in integrating with legacy systems has hindered their industrial adoption. Additionally, their emergent behaviour may not be desirable in contexts with tasks that need specific guarantees in their behaviour, such as business-to-business or business-to-customer relations and high-level plant management operations.

This Thesis proposes a more flexible view that allows a holon to have varying degrees of self-awareness and emergent behaviour, and defends that business holons may be better implemented as *services* or software components that can be reused through standards-based technologies from anywhere in the organisation. These services are usually organised into coherent catalogues, known as Service Oriented Architectures. A successful SOA initiative may provide large benefits to the organisation, but it is not a trivial task. For this reason, many methodologies have been proposed in the literature, but none of them cover the need to test the services that are finally developed. Considering that the goal of a SOA is to increase reuse of the available software throughout an organisation, this is an important oversight: high-quality services are crucial to a successful SOA.

Therefore, the main goal of this Thesis is defining an extended methodology that assists users in testing the services that implement their business holons. After considering the available options, the model-driven SODM methodology was selected as a starting point and was then largely rewritten using the open source Epsilon framework, allowing users to model their partial knowledge about the expected performance of the services. This partial knowledge is used by several novel performance requirement inference algorithms that extract the specific performance requirements of each service. While the throughput inference algorithm is based on a simple traversal of the graph, the time limit inference algorithm went through several revisions before reaching the desired level of functionality and performance. After a first formulation based on linear programming, it was replaced with a simple *ad hoc* algorithm and then with a much faster and more advanced incremental algorithm. The incremental algorithm produces equivalent results to the others while taking much less time, even in large models.

In order to extract more value from the models, this Thesis also provides a general approach for generating test artefacts for multiple technologies out of the models annotated by the previous algorithms. To evaluate the viability of this approach, it has been

implemented for two different applications: repurposing unit tests written in the Java programming language as performance tests, and generating performance testing projects in the The Grinder framework for any Web Service described using the language-agnostic Web Services Description Language standard.

Lastly, the full methodology is successfully applied on a case study based on a rectified tile manufacturing firm of a Spanish manufacturing enterprise group. This case study starts with a high-level description of the business and concludes with the partial implementation of one of the holons involved and the generation and execution of performance test artefacts for one of its Web Services.

With its support for designing performance requirements and testing them, SODM+T can therefore help users obtain a higher degree of confidence in the implementation of the business holons observed in their enterprises.

Contents

1. Introduction	1.1
1.1. Goals and scope	1.3
1.2. Context	1.3
1.3. Hypotheses	1.4
1.4. Document structure	1.4
2. Concepts of next-generation manufacturing systems	2.1
2.1. Challenges in manufacturing information systems	2.1
2.1.1. Evolution of production control systems	2.4
2.1.2. Holons, agents and services	2.6
2.1.3. Issues with existing methodologies	2.9
2.2. Extended enterprises	2.10
2.3. Enterprise integration	2.12
2.3.1. GRAI Integrated Methodology (GIM)	2.13
2.3.2. Purdue Enterprise Reference Architecture (PERA)	2.15
2.3.3. Computer-Integrated Manufacturing Open System Architecture (CIMOSA)	2.15
2.3.4. Generalised Enterprise Reference Architecture and Methodology (GERAM)	2.17
2.3.5. EN/ISO 19439 and EN/ISO 19440	2.21
2.3.6. The Open Group Architecture Framework (TOGAF)	2.21
2.3.7. IEC 62264 / ISA-95	2.23
2.4. Process modelling	2.26
2.4.1. Integrated DEFinition for Process Description Capture Method (IDEF)	2.28
2.4.2. Process Specification Language (PSL)	2.29
2.4.3. Value Stream Mapping (VSM)	2.30
2.4.4. Business Process Modelling Notation (BPMN)	2.30
2.4.5. Comparison through a case study	2.32
2.5. Multi-agent systems	2.39
2.5.1. Applications	2.39
2.5.2. Agent platforms	2.43
3. Concepts of software and service engineering for distributed manufacturing	3.1
3.1. Service-oriented architectures	3.1
3.1.1. Definitions and goals	3.1
3.1.2. Web Services	3.2
3.2. Performance engineering	3.8
3.2.1. Notations	3.8
3.2.2. Algorithms	3.9

3.3.	Model-driven software engineering	3.9
3.3.1.	Definitions	3.10
3.3.2.	Existing approaches	3.12
3.3.3.	Available technologies	3.14
4.	Existing service-oriented methodologies	4.1
4.1.	State of the art	4.1
4.1.1.	Prior work on component-based systems	4.1
4.1.2.	IBM SOMA	4.4
4.1.3.	SODM	4.4
4.1.4.	BPSOM	4.5
4.1.5.	Hoyer	4.7
4.2.	Selection of a base methodology	4.9
4.3.	Detailed description of SODM	4.10
4.3.1.	UML subset used by SODM	4.10
4.3.2.	Computation-independent models	4.16
4.3.3.	Platform-independent models	4.20
4.3.4.	Platform-specific models	4.21
4.4.	Extending SODM for testing	4.25
4.4.1.	System tests: performance requirements	4.25
4.4.2.	Function and integration tests: service contracts	4.27
4.5.	Conclusion	4.31
5.	SODM+T: extension of SODM for performance testing	5.1
5.1.	Introduction	5.1
5.2.	Extended metamodels	5.2
5.2.1.	Extended service process metamodel	5.2
5.2.2.	Extended service composition metamodel	5.4
5.3.	Extended model editors	5.4
5.3.1.	Computing least common ancestors	5.6
5.3.2.	Model validation	5.8
5.3.3.	Migration of service processes to service compositions	5.13
5.4.	Performance inference algorithms	5.13
5.4.1.	Input and output values	5.14
5.4.2.	Basic definitions	5.15
5.4.3.	Running example	5.16
5.4.4.	Throughput inference	5.17
5.4.5.	Time limit inference	5.18
5.5.	Evaluation	5.27
5.5.1.	Limitations	5.27
5.5.2.	Implementation	5.28
5.5.3.	Theoretical performance	5.29
5.5.4.	Empirical performance	5.32
5.6.	Conclusions	5.37

6. Generation of test artefacts with SODM+T and MARTE	6.1
6.1. The MARTE profile	6.1
6.1.1. Architecture	6.1
6.1.2. GQAM	6.3
6.1.3. VSL	6.3
6.2. Changes in SODM+T for MARTE	6.3
6.2.1. Revised annotations	6.3
6.2.2. Revised algorithms	6.6
6.3. Overall approach for test artefact generation	6.7
6.4. Reusing Java unit tests as performance tests	6.8
6.4.1. Model extraction	6.10
6.4.2. Weaving metamodel	6.11
6.4.3. Code generation	6.11
6.5. Generating performance tests for WSDL-based Web Services	6.13
6.5.1. Target performance testing tool: The Grinder	6.14
6.5.2. Model extraction	6.14
6.5.3. Weaving metamodel	6.17
6.5.4. Test data generation	6.18
6.5.5. Test code generation	6.19
6.5.6. Test infrastructure and report generation	6.20
6.6. Conclusion	6.22
7. Case study	7.1
7.1. Overall description	7.1
7.1.1. Enterprise profile	7.1
7.1.2. Manufacturing process for porcelain stoneware	7.2
7.1.3. Manufacturing facilities for porcelain stoneware	7.3
7.1.4. Providers	7.4
7.1.5. Information and material flows	7.4
7.2. Computation-independent models	7.5
7.2.1. Value models	7.7
7.2.2. Business process model	7.7
7.2.3. Business service list	7.10
7.3. Platform-independent models	7.10
7.3.1. Use case model	7.11
7.3.2. Extended use case models	7.11
7.3.3. Service process models	7.14
7.3.4. Service composition models	7.16
7.4. Platform-specific models	7.17
7.4.1. Extended service composition models	7.17
7.4.2. Web Service interface models	7.19
7.5. Implementation	7.20
7.5.1. Persistence layer: adaptation of the ISA-95 object model	7.23
7.5.2. Web interface: specification of rectification processes with ISA-95	7.25
7.5.3. Web service: provision of a scheduler WS	7.32
7.6. Performance test generation and execution	7.32
7.7. Conclusion	7.35

8. Conclusions and future work	8.1
8.1. Obtained results	8.1
8.2. Future work	8.4
8.3. Publications	8.5
8.3.1. Journal articles	8.5
8.3.2. Conference papers	8.6
8.3.3. Book chapters	8.7
A. Related proofs	A.1
A.1. Path ordering simplification	A.1
A.2. Path ordering as a partial order	A.3
B. The Epsilon EUnit testing framework	B.1
B.1. Motivation	B.1
B.1.1. Common issues	B.1
B.1.2. Testing with JUnit	B.2
B.1.3. Selected approach	B.3
B.2. Test organisation	B.3
B.2.1. Test suites	B.3
B.2.2. Test cases	B.5
B.3. Test specification	B.5
B.3.1. Ant buildfile	B.6
B.3.2. EOL script	B.7
B.4. Examples: testing a model transformation with EUnit	B.11
B.4.1. Models and tasks in the buildfile	B.11
B.4.2. Models and tasks in the EOL script	B.13
B.5. Extending EUnit	B.15
B.5.1. Adding model management tasks	B.15
B.5.2. Integrating model generators	B.17
B.6. Case studies	B.17
B.6.1. Regression tests for EuGENia	B.18
B.6.2. Unit testing for SODM+T	B.18
C. List of acronyms	C.1
D. Bibliography	D.1

List of Figures

2.1. Phases of a Collaborative Network	2.2
2.2. Evolution of manufacturing control structures in the 1980s and early 1990s	2.5
2.3. General architecture of a holon	2.6
2.4. An example of a holarchy	2.7
2.5. Concept of a Holonic Manufacturing System	2.11
2.6. A physical holon	2.12
2.7. GRAI model for a Decision System	2.14
2.8. Purdue Method	2.16
2.9. PERA task module	2.16
2.10. CIMOSA views	2.17
2.11. CIMOSA process modelling	2.18
2.12. GERAM framework components	2.19
2.13. GERA Modelling Framework	2.20
2.14. EN/ISO 19439 framework for enterprise modelling	2.21
2.15. TOGAF Architecture Development Cycle	2.22
2.16. ISA-95 activity hierarchy	2.25
2.17. ISA-95 equipment hierarchy	2.26
2.18. ISA-95 functional enterprise/control model	2.27
2.19. ISA-95 generic activity model	2.28
2.20. Selected subset of the IDEF3 notation	2.29
2.21. Selected subset of the Value Stream Mapping notation	2.30
2.22. Selected subset of the BPMN 2.0 notation	2.31
2.23. IDEF3 model for the notation comparison case study	2.33
2.24. BPMN 2.0 model for the notation comparison case study	2.36
2.25. VSM model for the notation comparison case study	2.37
2.26. Architecture of a MASCOT agent	2.41
2.27. Contract net for scheduling steel milling and casting	2.42
3.1. Language and model levels in MDE/MDSE	3.11
3.2. QVT and ATL transformation pattern	3.15
3.3. Screenshot of the GMF Dashboard view	3.17
4.1. Outline of the models used by SODM	4.6
4.2. Integration of BPSOM into the existing software development process . . .	4.6
4.3. Overview of the model-driven development process by Hoyer et al.	4.8
4.4. WSDL 2.0 metamodel	4.24
4.5. SODM WS interface metamodel	4.26
4.6. Model for the “Handle Order” service process, with non-functional testing extensions	4.28

4.7. Service composition model for “Handle Order” with functional and non-functional testing extensions	4.30
4.8. Example of a visual contract based on graph transformations for “Handle Order”	4.31
5.1. UML class diagram with the extended service process metamodel	5.3
5.2. UML class diagram with the extended service composition metamodels	5.5
5.3. Finding the least common ancestor of D and F	5.8
5.4. Annotated model for the running example	5.16
5.5. Execution trace of the exhaustive time limit algorithm	5.25
5.6. Execution traces for the incremental graph-based time limit algorithm	5.26
5.7. Screenshot of the Eclipse-based model editor	5.28
5.8. Graph shapes used in the performance analyses	5.30
5.9. Screenshot of the automated Eclipse-based performance comparison tool	5.33
5.10. Execution times of the throughput inference algorithm	5.34
5.11. Execution times of the time limit inference algorithms	5.35
5.12. Execution times of the incremental graph-based time limit inference algorithm by percentage of annotated nodes	5.35
5.13. Sampled 3-level fork-join models by top-level incomparable paths	5.37
6.1. Architecture of the MARTE profile	6.2
6.2. Simple example model annotated by the performance inference algorithms	6.5
6.3. Screenshot of the Eclipse Papyrus editor	6.6
6.4. Overall approach for generating performance test artefacts from abstract performance models	6.9
6.5. Instance of the above approach for wrapping JUnit tests into ContiPerf tests	6.9
6.6. MoDisco model browser showing a model generated from a Java project	6.10
6.7. Simplified subset of the MoDisco Java metamodel	6.11
6.8. Java-MARTE weaving metamodel	6.12
6.9. Screenshot of the Epsilon ModeLink editor weaving the MARTE performance model and the MoDisco model	6.12
6.10. Instance of the above approach for WSDL-based WS	6.15
6.11. ServiceAnalyzer service catalogue metamodel	6.16
6.12. ServiceAnalyzer-MARTE weaving metamodel	6.17
6.13. Example of an overall performance graph produced by Grinder Analyzer	6.21
7.1. Organisational chart of Keraben	7.2
7.2. Manufacturing process for porcelain stoneware	7.3
7.3. Map of the Keraben manufacturing firm	7.4
7.4. Map of the Keraben porcelain stoneware manufacturing plant	7.5
7.5. Information and material flows within Keraben	7.6
7.6. Gordijn value model for Keraben	7.8
7.7. BPMN 2.0 diagram of the high-level business process for Keraben	7.9
7.8. Use case model for Keraben	7.11
7.9. Extended use case model: “Order submission”	7.12
7.10. Extended use case model: “Order status reporting”	7.12
7.11. Extended use case model: “Order issue notification”	7.13

7.12. Extended use case model: “Estimated demand submission”	7.13
7.13. Extended use case model: “Production status reporting”	7.14
7.14. Extended use case model: “Production issue notification”	7.14
7.15. Service process model: “Order submission”	7.15
7.16. Service process model: “Order status reporting”	7.15
7.17. Service process model: “Order issue notification”	7.16
7.18. Service composition model: “Order submission”	7.18
7.19. Revised Web Service interface metamodel	7.21
7.20. Web Service interface model for “Estimate production dates”	7.22
7.21. UML class diagram for the <i>domain</i> base package	7.25
7.22. UML class diagram for the <i>people</i> subpackage	7.26
7.23. UML class diagram for the <i>equip</i> subpackage	7.26
7.24. UML class diagram for the <i>material</i> subpackage	7.27
7.25. UML class diagram for the <i>psegment</i> subpackage	7.27
7.26. UML class diagram for the <i>product</i> subpackage	7.28
7.27. UML class diagram for the <i>capabilities</i> subpackage	7.29
7.28. UML class diagram for the <i>schedule</i> subpackage	7.29
7.29. UML class diagram for the <i>perform</i> subpackage	7.30
7.30. Screenshot of the web administration panel	7.31
7.31. Performance testing results for “Estimate production dates”	7.34
A.1. General situation when comparing two paths	A.1
B.1. Example of an EUnit test tree	B.4
B.2. Comparison between parametric testing and theories	B.5
B.3. Screenshot of the EUnit graphical user interface	B.12

List of Tables

2.1. Zachman Framework for Enterprise Architecture	2.24
2.2. Differences between the selected process modelling notations	2.38
2.3. A comparison of the currently available agent platforms	2.44
3.1. 2D grid projection of a simple software factory schema	3.13
4.1. Feature comparison of the SOA methodologies under review	4.2
4.2. UML class diagram concepts used by SODM	4.11
4.3. UML use case diagram concepts used by SODM	4.14
4.4. UML activity diagram concepts used by SODM	4.15
4.5. Abstract and concrete syntax of Gordijn value models	4.17
4.6. New or changed elements in the SODM use case models	4.20
4.7. New or changed elements in the SODM extended use case models	4.22
4.8. Transformation rules from SODM extended use case models to service process models	4.23
5.1. Common restrictions for service process and service composition models	5.8
5.2. Additional constraints for service process models	5.12
5.3. Additional constraints for service composition models	5.13
5.4. Migration strategy from service processes to service compositions	5.14
5.5. Restriction counts and individual and total generation times for the LP- based algorithm, by component.	5.30
5.6. Restriction counts and generation costs for the LP-based algorithm, by graph shape, using the results from Table 5.5.	5.31
6.1. Mapping from the SODM+T custom annotations to MARTE	6.4
6.2. Example test metrics produced by Grinder Analyzer (overall results, through- put and message sizes)	6.20
6.3. Example test metrics produced by Grinder Analyzer (timing information)	6.21
B.1. Extra operations and variables in EUnit	B.9
B.2. Assertions in EUnit	B.10
B.3. Available options by model comparator	B.12

List of Listings

2.1.	Process Specification Language fragment describing the activity sequences dedicated to preprocessing tobacco	2.34
2.2.	Process Specification Language fragment describing the machines and materials used at each step in Listing 2.1	2.34
3.1.	Example of a HTTP request-response conversation	3.4
3.2.	Example of a SOAP message	3.6
3.3.	Abridged example of a WSDL document	3.7
4.1.	Service contract for “Create Invoice” in JML	4.29
5.1.	Definition of the <i>InitialNode</i> class with EuGENia annotations, using the Emfatic textual notation for EMF-based metamodels.	5.6
5.2.	GMPL model used by the algorithm in Section 5.4.5.1	5.23
5.3.	GMPL data for level 0 of the example in Figure 5.4	5.23
5.4.	GMPL data for level 1 of the example in Figure 5.4	5.23
6.1.	Java code wrapping <i>TFunctionalJUnit4</i> with ContiPerf	6.10
6.2.	Java code wrapping one test from <i>OriginalSuite</i> using ContiPerf	6.13
6.3.	Java code using JAX-WS for a “HelloWorld” Web Service	6.14
6.4.	Apache Velocity template extracted from the ServiceAnalyzer catalog for producing the test input message	6.18
6.5.	TestGenerator <i>.spec</i> extracted from the ServiceAnalyzer catalog describing the inputs for the template in Listing 6.4	6.18
6.6.	Template inputs produced by TestGenerator from the <i>.spec</i> in Listing 6.5	6.18
6.7.	Example <i>grinder.properties</i> file with workload configuration parameters	6.19
6.8.	Example Jython script for The Grinder with the contents of the performance test to be run by each simulated client	6.20
7.1.	Message catalogue generated from the WSDL document of the “Estimate production dates” Web Service	7.33
7.2.	Customised TestSpec specification of the input data for testing “Estimate production dates”.	7.33
B.1.	Format of an invocation of the EUnit Ant task	B.6
B.2.	Example of a 2-level data binding	B.8
B.3.	Example of reusing the same operation for several data bindings	B.8
B.4.	Examples of model bindings	B.8
B.5.	Ant buildfile for EUnit with <i><modelTasks></i> and a helper target	B.13
B.6.	EOL script using <i>runTarget</i> to run ETL	B.13
B.7.	Ant buildfile which only runs the EOL script	B.14

B.8. EOL script with inlined models and tasks	B.14
B.9. Testing an ATL model transformation with EUnit	B.16
B.10. Testing an EVL model validation with EUnit	B.16
B.11. Inline model generation in EUnit	B.17
B.12. Several model validation tests for SODM+T using EUnit	B.19
B.13. Java class used to run EUnit from JUnit-compatible tools	B.20

1

Introduction

According to the Merriam-Webster dictionary, manufacture is “something made from raw materials by hand or by machinery”, “the process of making wares by hand or by machinery especially when carried on systematically with division of labor”, “a productive industry using mechanical power and machinery” or “the act or process of producing something”. These definitions refer to several viewpoints through which manufacturing can be examined: the product, the process, the industry and the act itself.

The term “manufacturing” dates back to the Latin “manu factus”, meaning “made by hand”. Manufacturing itself is much older, and can be attributed to the rise of the first agricultural societies, in which some of the population did not need to look for food any longer and could specialise in a particular craft. Until the XVIIIth century, manufacturing was largely decentralised and performed by skilled artisans who passed their skills to their apprentices.

The first major change in manufacturing came with the first Industrial Revolution, which began in the 18th century and continued up to the middle of the 19th century. Technological advances in using water and steam as energy sources propelled the increasing usage of machine tools and more advanced metal working and chemical processes: for instance, the mechanised loom increased the amount of cotton that a single worker could process by a thousand. The famous Jacquard loom used punched cards to control the pattern to be weaved, and is an important precursor of today’s computer programming.

The second Industrial Revolution started in the 1860s and continued until World War I. It is considered to have started with the invention of the Bessemer process, the first inexpensive industrial process for the production of steel. The production of interchangeable parts (in what was known as the “American system”) and the scientific analysis of manufacturing and business practices (currently known as “manufacturing engineering” and “business management”) greatly increased the productivity of the factories. The invention of the telegraph and the telephone simplified communication over large distances, and the tabulating machine by Hollerith was one step further towards modern computers.

The 20th century brought along a large number of innovations, such as nuclear power or the jet engine. One of the most important milestones was the invention of the transistor in 1947 at Bell Labs, which quickly replaced the inefficient and fragile vacuum tubes that were used in electronics until that time, such as those in the first general-purpose computer, the ENIAC (announced in 1946). The rapid miniaturisation of transistors has enabled the creation of increasingly advanced computers and their integration into all aspects of modern life, including manufacturing.

The large rise in demand that occurred during the mid-XXth century motivated manufacturing firms to optimise their operations to reduce unit costs and defects and produce more advanced products. Some of the most repetitive and defect-prone tasks were gradually automated, originally with *ad hoc* mechanisms and finally with programmable machine tools built around a general-purpose computer. This led to the inception of Computer Numerical Control (CNC). Numerical Control (NC) started in the 1950s in the United States with machine tools being controlled by punched cards, and quickly evolved into CNC when programmable computers were integrated. CNC allowed producing complex parts repeatedly with strict tolerances, which was key to the advanced requirements set by the aerospace and automotive industries, among others. Computers are increasingly used in every manufacturing activity, such as designing parts (Computer Aided Design or CAD), evaluating their design (Computer Aided Engineering or CAE) or producing them (Computer Aided Manufacturing or CAM). In addition to producing the parts,

distribution within a firm or between firms can be also automated through conveyor belts and robots.

Beyond manufacturing, computers were increasingly used to manage the companies themselves. While the first Material Requirements Planning (MRP) systems only assisted users in producing detailed production schedules and purchase schedules from higher-level requirements, later Manufacturing Resource Planning (MRP II) systems also considered other aspects such as human resources, capacity planning or demand management, among others. Eventually, these systems evolved into the current Enterprise Resource Planning (ERP) platforms, which cover all information areas of an enterprise.

In recent years, improvements in transportation and communications have fostered collaboration between manufacturing firms, in what is known as distributed manufacturing. Large companies now operate as large networks of geographically separated entities, combining the company's own plants and external suppliers. Small and Medium Enterprises (SMEs) may either take orders from large contractors, or join up together to build products that they would not be able to create by themselves. Computers have been key enablers in this process as well, as they provide standard formats to exchange information across heterogeneous systems and define the shared business processes.

However, this dependency on connected networks of software systems have introduced a considerable level of complexity in the day-to-day operations of the firms, which is becoming increasingly harder to manage conceptually and technically. This has motivated the creation of a large number of abstractions and enterprise architectures that help describe the current networked enterprises.

One of the most common abstractions is *holonic manufacturing*, which views each of the networked entities in a manufacturing enterprise as a *holon*: a term coined by Koestler [6] as a collaboration of lower-level entities that can also take part in higher-level collaborations, constituting what is called a *holarchy* (a generalisation of a hierarchy). Holonic manufacturing systems view each manufacturing agent as a holon, and so there can be enterprise holons, plant holons, process cell holons and so on.

Holons are useful as a high-level concept, but they ultimately need to be implemented in some way as part of the information systems of the manufacturing enterprise. In the literature dedicated to holonic shop floor control, the most common approach has been to implement each holon as an intelligent autonomous agent. While highly powerful, the learning curve and integration challenges of most implementations of intelligent agents and the difficulties in extracting concrete guarantees from their emerging behaviour have hindered their widespread adoption except for highly complex processes that cannot be centralised [7]. In addition, other parts of the enterprise may benefit from holons with a more predictable and repeatable behaviour, such as the high-level management tasks and the communications with other firms and other participants in the supply chain.

For these contexts that require that the manufacturing agents behave in a repeatable and reliable way, it may be better to focus first on simplifying their integration and add only the necessary amount of emergent behaviour later. To that end, those information systems could be represented as Service-Oriented Architectures (SOAs), in which a well-defined catalogue of reusable software programs known as *services* integrate the information within the manufacturing enterprise and its network of collaborators. A single service can use and be used by several other services, and so can be seen as a simpler *holon* that focuses on its organisational capabilities and leaves emergent behaviours as an optional aspect.

Even if services are simpler and easier to implement and integrate than fully self-aware

intelligent agents, creating a SOA is no simple task. It is necessary to obtain a high-level view of the enterprise and examine the interactions between each part to see which programs should be exposed as services to the rest of the organisation. For this reason, a number of SOA methodologies have been proposed in the literature: some of them even have partial automation through the use of increasingly detailed computer models, using what is known as *model-driven engineering*.

Nevertheless, the existing methodologies are mostly focused on the early stages of creating the SOA, and do not assist users in validating the services they have finally developed by testing. This can be a grave omission, as services need to be highly reliable in order for a SOA initiative to be successful.

1.1 Goals and scope

The overall goal of this Thesis is to define a methodology that helps develop the information systems of holonic manufacturing enterprises, focusing on implementing holons in a cost effective way and on helping create the tests for the system. The present Thesis will focus on service-based holons and their capabilities for intra- and inter-enterprise integration, as modelled by the interface between levels 3 and 4 of the ISA-95 standard (Section 2.3.7).

This goal can be subdivided into:

1. Define a methodology that can represent both the information systems of holonic manufacturing enterprises and the tests that should be performed on them. The methodology will be based on the stepwise refinement of a collection of models from the enterprise's business environment to a blueprint of the final system, using *model-driven engineering*.
2. Select and adopt several techniques that assist in generating the test cases over the desired functionality, running them and evaluating the results.
3. Define a set of rules that assist users in deciding how to implement a holon, as these may include varying levels of intelligence.
4. Implement the methodology and validate it through a case study from a distributed manufacturing enterprise.

1.2 Context

The present Thesis is a continuation of the research line on Emergent Manufacturing Systems of the research group TEP-027 "Ingeniería y Tecnología de los Materiales" from the University of Cádiz, led by Dr. Mariano Marcos-Bárcena. Previous work from the research group in collaboration with researchers from the University of Seville produced a Thesis on the development of a holonic design module [1], a book relating the current state of the art in advanced distributed manufacturing systems [3] and a number of contributions on several conceptual approaches for holonic manufacturing [2, 4, 5, 8, 9, 10]. While these conceptual approaches were promising, the group did not have the required software engineering expertise to implement them into real information systems, and the conceptual approaches lacked some of the additional details needed for these later stages.

Therefore, the present Thesis is conducted as a collaboration with the research lines on Service-Oriented Architectures, Model-Driven Engineering and Software Testing of the TIC-025 “UCASE Software Engineering” research group led by Dr. Inmaculada Medina-Bulo. The Thesis intends to provide a concrete method for implementing some of the ideas in the initial conceptual approaches from the TEP-027 research group by applying the software engineering knowledge of the TIC-025 research group. The author received the necessary foundations for this multidisciplinary research by complementing his Bachelor’s Degree on Computer Science with a Master’s Degree on Manufacturing Engineering before starting the present work.

1.3 Hypotheses

The present Thesis has been developed based on the following hypotheses:

- A Service-Oriented Architectures (SOAs) with extensions to provide the required intelligence can meet the requirements imposed by Shen et al. [11] in distributed manufacturing enterprises. Namely, integrate their software and hardware, define an open architecture that can accept new components, communicate effectively inside and outside the company, take human resources into account, react quickly to changes and achieve fault tolerance at multiple levels.
- A defect in a service that has been reused across the SOA of a holonic manufacturing enterprise can generate considerable expenses and hinder the creation of an effective extended enterprise. Developing tests for these services is less expensive than the potential losses in trust and revenue.
- Existing SOA methodologies are a good starting point for defining a new methodology that explicitly takes testing techniques into account, reduces the risk that the system will not behave in the expected manner at some point and assists user in deciding which holons should include additional intelligence.
- Using a model-driven methodology helps drive its usage, reduces its cost and improves the agility of the enterprise to respond to changing technical and business requirements.
- Current overall approaches such as Model-Driven Architecture® (MDA®) from OMG or Software Factories (SF) from Microsoft help realise the overall vision of the enterprise reference architectures that are well-known in the manufacturing world, such as CIMOSA or GERAM. In both cases, the system is formed by combining models at different levels of genericity and granularity and from multiple perspectives in the organisation.

1.4 Document structure

The rest of this text is divided into the following chapters:

- Chapter 2 introduces the concepts underlying this work, starting by expanding on the ideas outlined at the beginning of this chapter. Next, it introduces extended enterprises and the existing abstractions for them, presents the existing enterprise reference architectures and reviews several existing notations for describing manufacturing processes. Later, it describes the current state of the art and applications of multi-agent systems. This is followed by a description of Service-Oriented Architectures (SOAs) and the technologies underlying Web Services. Since SOA is heavily reliant on obtaining the desired level of performance in each service, performance engineering is then introduced. The chapter concludes with a brief discussion of model-driven engineering concepts and tools.
- Chapter 4 compares existing SOA methodologies and some antecedents and selects SODM as the base SOA methodology, presenting it in more depth. It then proposes several extensions on SODM for modelling functional and non-functional requirements.
- Chapter 5 presents SODM with Testing (SODM+T), an extension of Service Oriented Development Method (SODM) with support for performance testing, with improved models and validation. Several algorithms for early performance requirements design are presented: three for time limit inference and one for throughput inference. All algorithms are successfully tested and then evaluated theoretically and empirically.
- Chapter 6 migrates the SODM+T annotations to the standard OMG UML Modelling and Analysis of Real-Time and Embedded Systems (MARTE) profile, and then shows how to generate performance test artefacts for multiple technologies by using model weaving between the annotated models and models extracted from the code. In particular, the approach is demonstrated for Java code and Web Services Description Language (WSDL) documents. Model extraction from Java is performed using an existing tool (MoDisco). Model extraction and random test case input data generation from the WSDL documents are performed using several new tools.
- Chapter 7 applies SODM+T to a manufacturing enterprise from start to finish. The SODM+T business models are extended with details from the ISA-95 standards and then refined until obtaining a service composition model for a particular business service indicating which holon should perform each task. This model is then annotated using the extensions in Chapter 5. Before generating and running the performance test artefacts generated for the Web Services (WS) using the approach in Chapter 6, the case study pays special attention to the way in which the WS is implemented based on the ISA-95 data model. Several issues in the ISA-95 object model are found in the process.
- Chapter 8 summarises the results of this work and presents future lines of work.
- Appendix A provides several additional proofs that are important for the correctness of one of the time limit inference algorithms in Chapter 5.
- Appendix B introduces the Epsilon EUnit unit testing framework for model handling tasks, which was developed during a 3-month stay at the University of York

(United Kingdom). EUnit has been used for the technical validation of the inference algorithms and the modelling tools.

- Finally, Appendices C and D collect the acronyms and the bibliography used in this document, respectively.

References

- [1] F. Aguayo González. *Diseño y Fabricación de Productos en Sistemas Holónicos: Aplicación al Desarrollo de un Módulo Holónico de Diseño*. PhD thesis, University of Cádiz, 2003. 1.3
- [2] F. Aguayo González, J. Lama Ruiz, M. Sánchez Carrilero, R. Bienvenido Bárcena, J. González Madrigal, and M. Marcos Bárcena. Concepción holónica de la ergonomía en sistemas de fabricación automatizados. *Anales de Ingeniería Mecánica*, pages 1087–1095, 2004. 1.3
- [3] F. Aguayo González, M. Marcos Bárcena, M. Sánchez Carrilero, and J. Lama Ruiz. *Sistemas Avanzados de Fabricación Distribuida*. Ra-Ma, Madrid, España, 2007. ISBN 9788478978045. 1.3
- [4] R. Bienvenido Bárcena, M. Álvarez Alcón, J. González Madrigal, M. Marcos Bárcena, and M. Sánchez Carrilero. Holonic manufacturing systems: an emergent proposal for the 21st century. *The International Journal for Manufacturing Science and Production*, 1999. 1.3
- [5] J. González Madrigal, J. Sánchez Sola, M. Marcos Bárcena, and M. Sánchez Carrilero. Aproximaciones a los sistemas de fabricación holónicos. *Informacion de Máquinas-Herramientas y Equipos*, pages 59–65, 1998. 1.3
- [6] A. Koestler. *The Ghost in the Machine*. Penguin Books, June 1990. ISBN 978-0140191929. 1.2
- [7] P. Leitão. Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence*, 22(7):979–991, October 2009. ISSN 0952-1976. 1.2
- [8] M. Marcos Bárcena, M. Álvarez Alcón, M. Sánchez Carrilero, and J. Sánchez Sola. Sistemas de fabricación holónicos: una propuesta para el siglo XXI. *Anales de Ingeniería Mecánica*, 12(3):275–281, 1998. 1.3
- [9] M. Sánchez Carrilero, M. Marcos Bárcena, M. Álvarez Alcón, J. Sánchez Sola, and R. Bienvenido Bárcena. El diseño en los sistemas de fabricación holónicos. In *Actas del X Congreso Internacional de Ingeniería Gráfica*, pages 312–330, 1998. 1.3
- [10] M. Sánchez Carrilero, F. Aguayo González, J. Lama Ruiz, R. Bienvenido Bárcena, and M. Marcos Bárcena. Integración de modelos biónicos, holónicos y fractales para fabricación distribuida. *Anales de Ingeniería Mecánica*, pages 395–403, 2004. 1.3

- [11] W. Shen, Q. Hao, H. J. Yoon, and D. H. Norrie. Applications of agent-based systems in intelligent manufacturing: An updated review. *Advanced Engineering Informatics*, 20(4):415–431, October 2006. ISSN 1474-0346. 1.4

2

Concepts of next-generation manufacturing systems

This chapter introduces the concepts related to manufacturing engineering that underlie the present Thesis and the challenges that remain to be solved.

Section 2.1 presents the current challenges confronted by manufacturing information systems and the available solutions for these challenges. After discussing the historic trend towards increasingly distributed manufacturing systems in Section 2.1.1, Section 2.1.2 will review the relationship between overall concepts of holons, agents and services and Section 2.1.3 will list the issues with the existing agent-based methodologies.

This motivational section is followed by several sections introducing extended enterprises (§2.2) and a survey of the available enterprise integration architectures and specifications (§2.3). Section 2.4 provides a comparison of a selected subset of process modelling notations. Finally, Section 2.5 reviews the state of the art in multi-agent systems and their relationship with holonic manufacturing systems.

2.1 Challenges in manufacturing information systems

Manufacturing enterprises need to deal with an increasingly dynamic market that requires increased variety and quality with reduced margins, and which imposes ever shorter product life cycles. To stay ahead, enterprises must focus on their key strengths and collaborate with others for the rest of the work, joining up into temporary Virtual Enterprises (VEs) to meet the incoming business opportunities. Johansen et al. [43] presented a case study on the importance of these issues for the Saab car maker regarding vehicle building and maintenance. According to Camarinha-Matos and Afsarmanesh [12], VEs have the following advantages:

- *Agility*: the ability to recognise, rapidly react and cope with the unpredictable changes in the environment in order to achieve better responses to opportunities, shorter time-to-market, and higher quality with less overhead.
- *Complementary roles*: several enterprises working together can enter markets that were previously inaccessible in isolation.
- *Achieving dimension*: a partnership can make the whole achieve critical mass and appear “larger” in new markets.
- *Competitiveness*: achieving cost effectiveness by dividing subtasks across specialisations in the VE and quickly gathering the necessary competences.
- *Resource optimisation* and *innovation* by sharing resources and ideas between enterprises, respectively.

These advantages are especially important in Europe, where most of the economy is based on SMEs. According to the 2004 EU Manufuture report [54], 99% of the 2.5 million manufacturing businesses at the time were SMEs. While SMEs tend to exhibit greater agility than larger organisations, their Information Technology (IT) adoption is limited by implementation costs, budget constraints, lack of technological awareness and IT skills, difficulties to determine costs and benefits and questions on the reliability and security of the technology [60].

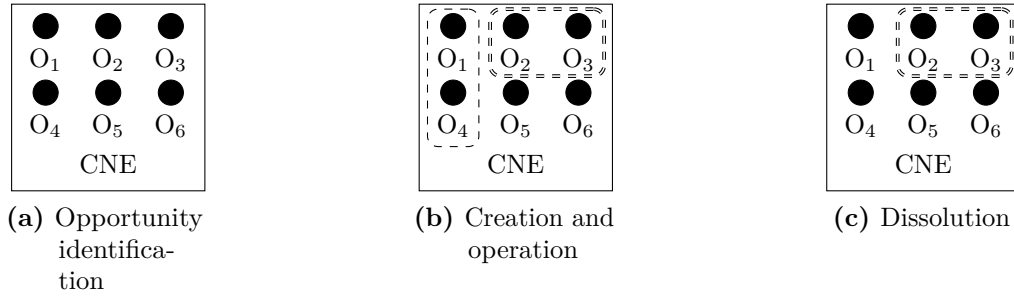


Figure 2.1. Phases of a Collaborative Network [60]: initially, six business opportunities are identified. Two CNs are created to follow some of them. Later on, one of the CN disappears while the other remains in operation.

On their review, Chituc et al. [15] generalised the VEs to the general economic concept of a collaborative business enterprise: a Collaborative Network (CN). The lifecycle of a CN is divided into several phases (shown in Figure 2.1): opportunity identification, creation, operation and dissolution. The same enterprise may participate in multiple CNs, and CNs may change during operation, e.g. when one of the participants disappears during bankruptcy or merges with a third party. Once the objectives that motivated the creation of the CN are met, the CN disappears.

The success of a CN or VE depends on having a common base among its members: common goals and standards, interoperable IT infrastructure and real-time information sharing. Interoperability is a particularly difficult technical requirement, as it requires not only creating and selecting a compatible set of technologies, but also ensuring that the information is well understood across all organisations. These two requirements are known as *application integration* and *information integration* [69].

Application integration focuses on using a compatible set of technologies across all participants, and has received considerable attention by researchers and industry. Initial initiatives tended to be *ad-hoc* for each integration case and therefore quite expensive. Later on, several specifications were drawn and implemented, such as Remote Method Invocation (RMI) or Common Object Request Broker Architecture (CORBA), but these did not see widespread adoption as they were either platform-specific (Java-specific in the case of RMI), highly complex (CORBA), or both. More recent developments have seen the rise of Web Services (WS), which are pieces of software based on the same open standards as the World Wide Web that can be reused from any platform with a conforming implementation. It has been suggested to reorganise existing systems as catalogues of services, in order to take advantage of this reusability and save on development and integration costs: this approach is known as a Service-Oriented Architecture (SOA).

Information integration requires giving the exchanged data a consistent meaning over the CN. The meaning includes the context in which each message was sent, the set of concepts it is built upon (its *ontology*), and how these concepts should be interpreted. Research and development on these aspects has produced a variety of application integration platforms (as implementation platforms), *enterprise architectures* (as common languages for modelling enterprises) and *enterprise ontologies* (representations of the concepts in a business domain). Nevertheless, using these concepts is difficult and can be out of reach for most SMEs, due to the costs involved. A simplified form which can be supported by

automated tools would make them more accessible.

In addition to a greater need for integration and interoperability with their business partners, current manufacturing enterprises need to make their plants more adaptable and reactive to unexpected situations. Shen et al. [74] lists six requirements that every new manufacturing enterprise needs to meet to be competitive:

- R1. Full integration of heterogeneous software and hardware systems within an enterprise, a virtual enterprise, or across a supply chain;
- R2. Open system architecture to accommodate new subsystems (software or hardware) or dismantle existing subsystems “on the fly”;
- R3. Efficient and effective communication and cooperation among departments within an enterprise and among enterprises;
- R4. Embodiment of human factors into manufacturing systems;
- R5. Quick response to external order changes and unexpected disturbances from both internal and external manufacturing environments; and
- R6. Fault tolerance both at the system level and at the subsystem level so as to detect and recover from system failures and minimise their impacts on the working environment.

R1 to R3 deal with information integration, much in the same line as discussed above. R4 is outside of the scope of the present Thesis, which is focused on the IT of the manufacturing enterprises. However, R5 and R6 require that the manufacturing enterprise implements a certain degree of intelligence into their process. Therefore, the automated steps will need to be able to reason about their current situation and react accordingly to meet the production requirements. This problem is orthogonal to enterprise integration, and belongs to the field of production control and scheduling: nevertheless, better information should enable the control systems to make better decisions. It is important, therefore, to ensure that the adopted control techniques can integrate deeply with the business environment.

In summary, the following challenges in manufacturing IT must be considered:

- Obtaining a greater degree of integration and interoperability across enterprises, making VEs more accessible to the many SMEs that make up the largest part of the industry in most economies. SOAs are a recent development that may simplify interoperability between manufacturing information systems, but their adoption needs to be simplified in order to become feasible for most SMEs.
- Developing more advanced production control systems for reacting to unexpected events, while keeping their development and operation complexity within a manageable limit for SMEs. The existing approaches for production control will be discussed in more depth below.
- Integrating the IT dedicated to managing the enterprise with the IT dedicated to controlling the production processes, so both can benefit from the additional information. The business should be notified as soon as possible of disturbances in manufacturing, and the manufacturing process should be able to quickly react to events such as cancelled or changed orders.

2.1.1 Evolution of production control systems

The very first computerised production control systems were *centralised*: a single mainframe controlled every machine in the plant, which had to run in lockstep with it [21]. This allowed for global optimisation across the entire plant and simplified monitoring, as there was only one source for information, but it placed enormous demands on the mainframe. Response times were slow, a single failure could stop the entire plant and the system was hard to extend when new machines were acquired.

In order to provide some redundancy, *hierarchical* control systems divided the intelligence of the original mainframe into levels: most of the intelligence was in a central plant controller, which then delegated on area controllers, then on cell controllers and finally on machine controllers. As orders continued down the hierarchy, the systems became simpler and simpler all the way to the machine level, which could do little beyond following its own orders. This reduced the complexity involved in adding a new machine and allowed having a mix of different computers working at different speeds. However, this did not solve the original issue of having the main controller as a single point of failure: in fact, the system was now threatened as well by failures in the communication links to the main controller. In order to solve this issue, *modified hierarchical* control systems were proposed, where some of the controllers could coordinate themselves independently from the main controller, which only sent orders and managed exceptional situations.

The next step in this evolution were *heterarchical* control systems, in which every pair of elements could talk to each other through a shared medium (e.g. a computer network). In this fully decentralised scheme, every element negotiated with the rest to meet its own goals, using for example the Contract Net Protocol [66]. A heterarchical control system offered several important advantages: individual controllers were easier to implement, the cooperation protocols ensured that the system worked in the presence of errors and adding a new machine did not require changing the rest of the system. However, heterarchical control systems were unable to perform global optimisation, as elements only had local knowledge, and were very sensible to the “market rules” of the cooperation protocols (e.g. relative importance of transport times). The flow time of an order was also hard to predict, as it depended on all the other orders in the system.

Each step of the above evolution (summarised in Figure 2.2) made the control structure flatter in order to benefit fault tolerance and responsiveness (two of the above requirements by Shen et al. [74]), but global optimisation and increased predictability would require adding back some structure into the control system. This has motivated the creation of more advanced manufacturing systems, such as Biological Manufacturing Systems (BMS), Fractal Manufacturing Systems (FrMS) or Holonic Manufacturing Systems (HMS).

A BMS models the enterprise as a set of “tissues” (processes, products or services), formed by cells or *modelons* that perform specific operations and receive and produce genetically coded artefacts. Vaario and Ueda [83] illustrate a BMS in a case study for a small assembly plant. Cells can evolve autonomously according to a fitness function evaluated during simulations, which may vary according to the current needs of the enterprise. Cells can self-regulate and communicate through mechanisms that mimic enzymes and hormones, and can also be divided or merged during operation.

A FrMS builds up the system from self-similar entities at multiple levels of scope [72]: these were inspired by the “fractal factory” concept in Warnecke [84], which was originally applied to companies at the organisational level. The main component of a FrMS is a

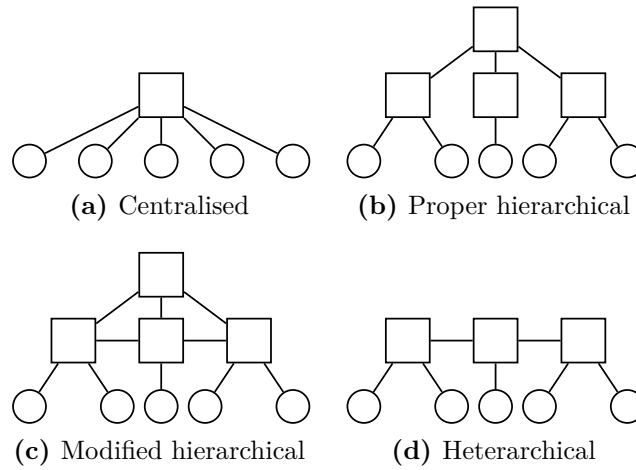


Figure 2.2. Evolution of manufacturing control structures in the 1980s and early 1990s from a centralised approach with a single mainframe, to decentralised approaches based on a computer network [21]. The squares represent controllers and the circles represent machines.

Basic Fractal Unit (BFU), which is composed of five functional modules: an observer, an analyser, a resolver, an organiser and a reporter. Owing to their fractal nature, BFUs behave in the same way regardless of the level of aggregation: a BFU may represent a single machine or an entire plant. Each BFUs provides services following an individual goal while inheriting some plant-level goals from the higher level BFUs. A FrMSs may change their structure during execution in order to improve its performance, creating higher-level fractals or dividing an existing fractal into several ones.

Finally, an HMS consists of a collection of *holons*, a term coined by Koestler [46]: an entity that is both a “whole” (the Greek prefix *hol-*) by itself and may be “part” (*-on*) of several other holons, participating in a *holarchy* (a hierarchy that allows nesting). The concept of a holon was first adopted in manufacturing for one of the case studies conducted by the Intelligent Manufacturing Systems (IMS) project Christensen [16]. Unlike the fractals in a FrMSs or BMSs, holons generally do not divide, merge nor evolve [72]. Holarchies follow a simpler approach that is closer to a functional decomposition of the information system. For instance, the Product-Resource-Order-Staff Architecture (PROSA) has a predefined set of holons (process, resource, order and staff holons) [7]: these holons are only created through explicit interaction with the system (e.g. submitting a production order or adding a machine). However, holons do negotiate and cooperate between themselves: approaches based on the Contract Net Protocol are quite common.

It can be seen that BMSs, FrMSs and HMSs share the same basic idea that the enterprise is composed of a number of independent units that negotiate and cooperate with each other (as in heterarchic manufacturing), and which can be aggregated at multiple levels (as in hierarchical manufacturing). However, they operate quite differently: while cells can evolve, divide and merge and fractals can divide and merge, holarchies tend to stay the same except for adding or removing instances of the available types of holons. Negotiation and scheduling approaches may also differ considerably. In practice, HMSs have received wider adoption in the research community, as they are simpler to implement.

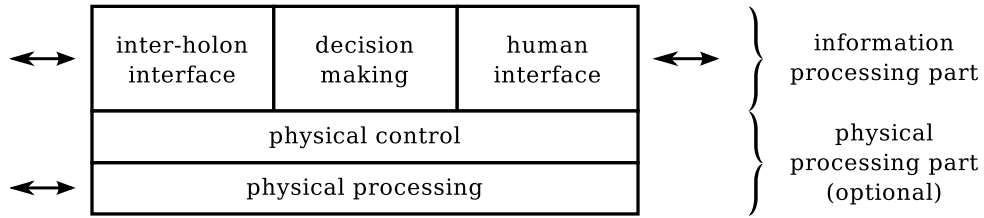


Figure 2.3. General architecture of a holon [9]

2.1.2 Holons, agents and services

The previous sections showed the increasingly distributed nature of manufacturing, whether considering organisational and business aspects, application and information integration or production control. In order to unify these views, it is necessary to provide a concept that unifies the basic units of these perspectives. The concept of a *holon* as originally coined by Koestler is particularly attractive because of its genericity: it can be applied to any domain (whether social or technical), by mapping the holons to the basic building blocks from which the complex systems or societies are built.

This was the approach followed by [16] for production control, expanding on some early work by Japanese researchers. The HMS Consortium specialised the basic concepts of a holon and a holarchy as applied to manufacturing systems: a holon was “an autonomous and cooperative building block of a manufacturing system for transforming, transporting, storing and/or validating information and physical objects”, and a “holarchy” was “a system of holons which can cooperate to achieve a goal or objective”. In particular, Christensen divided the holon into an *information processing* part and an optional *physical processing* part which received the materials and resources, and placed special attention on the integration with human intelligence.

Based on these ideas, Bussmann [9] presented the general architecture of an HMS holon shown in Figure 2.3: the kernel is the decision making component, which guides the communication with the other artificial holons and with humans and provides instructions to the physical control component if it exists. The physical control component manages the physical process required for manufacturing.

Fletcher et al. [25] then suggested placing holons into *cooperation domains*, enabling them to locate, contact, interact and collaborate with each other. Holons could be standalone entities (*atomic*) or be composed from several lower-level holons (*compound*), as the holarchy in Figure 2.4 shows. For instance, the compound holons CH2 and CH5 receive control information from the cooperation domain CD1, which also exchanges data with CH2. In turn, CH2 contains the atomic holons AH1 and AH2 and the compound holon CH1, working together through the nested cooperation domain CD21. Ideally, cooperation domains should evolve dynamically upon demand, but current implementations tend to be limited to hard-wired structures with predefined reactions [56].

Holons are commonly implemented as Multi-Agent System (MAS) since first proposed by Bussmann [9]. An agent is “an autonomous component that represents physical or logical objects in the system, capable to act in order to achieve its goals, and being able to interact with other agents [...]” [52] and a MAS is simply a collection of agents interacting with each other. MAS have received considerable attention from the artificial intelligence research community, with the development of reference architectures, methodologies and

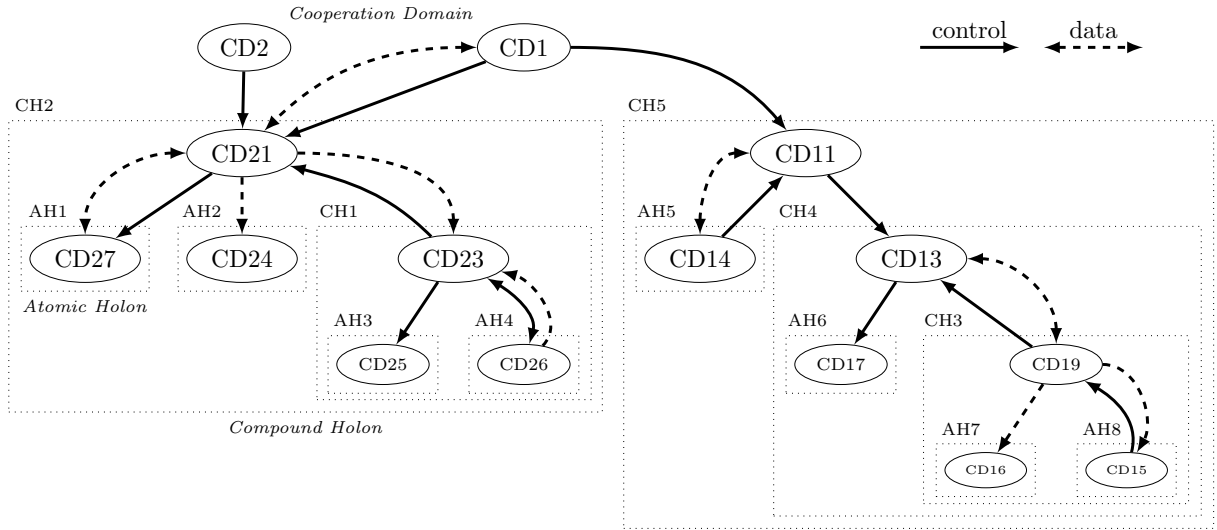


Figure 2.4. An example of a holarchy, adapted from [25]: a hierarchy of holons which can be nested within each other and communicate themselves through cooperation domains by sending and receiving data and control information.

software platforms [26, 62] (§2.5). However, they have not been widely adopted by industry. Monostori et al. [58] and Leitão [52] identified several possible reasons:

- MASs do not reduce the complexity of a problem. In fact, they may increase the initial cost of developing a solution, as these systems use a new set of concepts which will be unfamiliar to the manufacturing enterprises.
- Interoperability is expensive: just because of the increased communication overhead, coarse-grained systems with sophisticated agents are difficult to scale up. Beyond this overhead, it is important to use standard platforms which allow transparent communications between applications. Shared ontologies would also be useful, but due to their cost, they tend to be kept very simple.
- Reconfiguration and integration of physical devices is much more complex in the “real world” than in the simulation-based settings of most research papers.
- Industrial-strength tooling is missing, and wrapping legacy systems into agents is still a manual affair.
- The emergent behaviour of a MAS is also a barrier to practical acceptance of agent-based solutions: industry needs safeguards against unpredictable behaviour and guarantees regarding reliability and performance. Demonstrating that a MAS has the desired system properties is still an open issue.
- Finally, Leitão [52] argues that while MASs and HMSs are frequently said to perform well in presence of disturbances, there is little objective evidence of it in the literature. Proper evaluation frameworks with normalised performance indicators are needed.

Since HMS are usually based on MAS, they share many of these challenges. These findings suggest that while deliberative and proactive agent-based holons may be useful

for highly complex production control systems, they may impose too much complexity in simpler cases or in other areas of the organisation. In fact, it can be argued from the architecture proposed by Bussmann in Figure 2.3 that existing software systems such as a Manufacturing Execution System (MES) that runs the shop floor or the ERP system that manages the business functions are also holons: they can be aggregated at several levels, they communicate with other systems through machine-oriented interfaces and with humans through graphical user interfaces, and it is increasingly common to add reactive and proactive intelligence into them through expert systems, business rule engines or event-driven architectures, among many other techniques. Therefore, not all holons may need to be full-fledged agents: instead, they may have very different levels of intelligence depending on what is needed from them.

From a software engineering perspective, a holon could be thought of as a reusable component with a clear set of interfaces with humans and other components or systems. However, whereas an instance of a regular component would only be part of a single whole (e.g. a software library inside a program), the same holon could be part of multiple “wholes” by participating in several cooperation domains. On top of being autonomous, holons need to have a inter-holon interface based on open standards in order to be interoperable with each other: if necessary, communication primitives beyond the usual request/response interaction (such as the Contract Net Protocol) could be built on top of them. The cooperation domains mentioned by Fletcher can also be simplified depending on the needs of the holons: starting from a regular computer network, more services could be added with the appropriate middleware.

Therefore, this Thesis suggests generalising the implementation of a holon from a full fledged agent to a reusable piece of software that has the following features:

- can be integrated into several other holons while retaining its own separate identity,
- performs a well-defined task which can be reused from other holons,
- provides machine-oriented interfaces through open standards-based technologies,
- uses one or more communication primitives (e.g. request/response or auctions),
- communicates with other holons through one or more cooperation domains,
- may include a human-facing interface (such as a web interface or desktop application),
- 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).

These simplified requirements for implementing a holon can be considered as an extension of the concept of a *service* in SOAs [33]. In fact, Ribeiro et al. [70] found that agents and services had their own strengths and weaknesses and that they complemented each other. Several other authors have already proposed wrapping machines and agent platforms as services [53, 77]. However, there are few approaches that fully combine both approaches: either everything is an agent and only some SOA technologies are used, or agents are ignored in favour of services. An alternative approach would be to create a high-level

model of the manufacturing processes and their supporting business processes, derive the holons from these descriptions and then decide where how “smart” each holon should be.

In closing, this Thesis suggests that the implementation space for a holon is not limited to full-fledged agents, but instead should range from purely reactive service-based holons (which may be simply wrapping legacy software) to fully proactive and intelligent agent-based holons. Development efforts should focus first on obtaining an adequate level of interoperability, and then add intelligence to the holons that require it.

2.1.3 Issues with existing methodologies

There are many methodologies in the literature for developing MAS: two examples are ANEMONA [31] and ASEME [75]. ANEMONA focuses on the analysis of the system: starting from a requirements document with the business practices of the organisation and the production goals, the user decomposes the system into a holarchy of holons, which are then completed with design information for the JADE agent platform¹. However, ANEMONA seems to have little support for automation and no code is publicly available. ASEME starts by modelling the actors and their goals and the use cases of the system, then defining the roles of each agent and specifying their internal behaviour as statecharts, and finally generating the appropriate code for the JADE agent platform. While ASEME has no explicit support for modelling holons, its tools are publicly available and include automated transformations between each step of the methodology. Neither of these technologies consider the possibility that some of the agents may need to be available as services in order to be accessible from external systems.

On the other hand, developing a SOA requires strictly following its basic principles and managing a catalogue of well-defined, reusable and reliable services across the entire organisation [24]. This is not a simple task, and so several methodologies have been proposed to guide the process. Many cover only part of the process, such as the one proposed by Engels et al. [23], but others attempt to cover a large part of the entire process, such as SOMA from Ghosh et al. [30], SODM from de Castro [19] or the component-based methodology from Stojanović [76]. However, these methodologies do not consider assisting the user in defining the test cases, leaving them as a manual task or excluding them from the methodology. This can be a grave omission, since the higher reliance on code reuse that SOA promotes would amplify the potential impact of a software defect. Without well-defined tests, continuous changes on the services would be too dangerous and the agility promised by SOA would not be achieved.

Therefore, it can be concluded that:

- Manufacturing is increasingly moving towards distributed models in all its areas: instead of rigid hierarchical systems with single points of failure, using networks of autonomous entities allows for quicker integration and increased fault tolerance.
- Among the available paradigms, holonic manufacturing has had the widest adoption in the research community. Holonic manufacturing systems have been commonly implemented as multi-agent systems in the literature, but reception by industry has been limited due to their additional complexity and the difficulties in providing guarantees for their emergent behaviour.

¹jade.tilab.com

- Instead, it may be better to implement holons as services that can be reused throughout the organisation, and then give them the intelligence that may be required by each part of the organisation. This intelligence may come in various forms, including but not limited to software agents.
- A methodology is required to support the development of these holonic systems. However, neither the existing agent-oriented methodologies nor the existing service-oriented methodologies provide all the necessary constructs.

Existing holonic or agent-oriented methodologies do not explicitly consider the need to expose some of the agents as services, and the way in which they should integrate with the rest of the enterprise. On the other hand, existing service-oriented methodologies are focused on the decomposition of the system as a catalogue of services and do not provide explicit support for testing the services and assigning a certain level of intelligence to each service-based holon.

2.2 Extended enterprises

Companies are not run in a vacuum: they need to take part in the ecosystem formed by providers, designers, manufacturers, contractors, clients and competitors. In order to be successful, they need to add more value to their products than their competitors. This usually requires taking advantage of available information in order to quickly respond to market demands. In specific terms, Shen et al. [74] list six basic requirements that every new manufacturing enterprise should meet in order to be competitive: integrate their software and hardware, define an open architecture that can accept new components, communicate effectively inside and outside the company, take human resources into account, react quickly to changes and achieve fault tolerance at multiple levels.

In addition, nowadays companies are forced to interact beyond the local geographical area they are used to operate on. Using interconnected information systems is a key step, as well as standardising business practices and solving various logistics problems. After this integration has been achieved, the resulting organisation is known as an extended enterprise [6]. Extended enterprises are based on the establishment and preservation of long-term links of trust to other companies in the value chain. In contrast, agile enterprises rely on quick and continuous adaptation, and virtual enterprises focus on reacting to specific situations with temporary alliances. Johansen et al. [43] have discussed the importance of these long-term links in a case study on the Saab car maker.

Many different approaches have been suggested for describing the structure of an extended enterprise. The most immediate approach is heterarchic manufacturing [21], in which every part of the company can talk to the rest. This provides a high degree of flexibility, but it is hard to predict and optimise. This urged the creation of more structured models, such as bionic, fractal or holonic models [79]. Some authors have even suggested integrating these paradigms [55].

Bionic models are inspired in biological systems [83]. The company consists of a set of “tissues” (representing processes, products or services), formed by “cells” that perform various functions and receive and send genetically codified artefacts (parts and products). Continuing this analogy, a cell controls its own functions by secreting “enzymes” (internal control information) and can influence or be influenced by “hormones” (information from

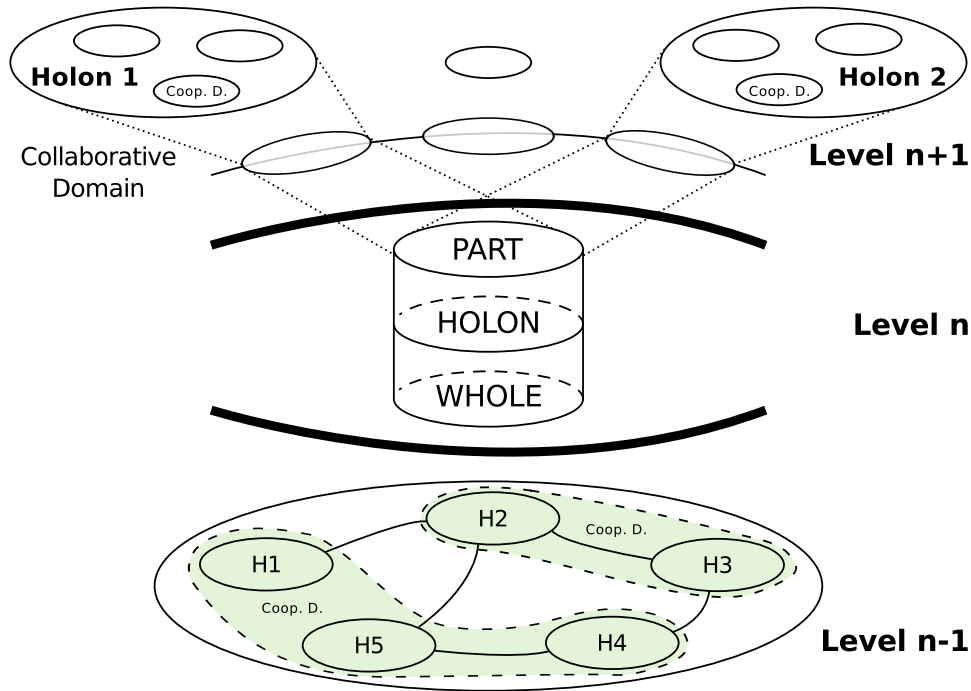


Figure 2.5. Concept of a Holonic Manufacturing System (adapted from [55])

the environment and other business units). Critical situations can be quickly handled using the faster “nervous system” (i.e. a company-wide messaging system).

On the other hand, fractal models start from the mathematical constructs of the same name [84]. These are focused on building the company from self-similar entities at several scope levels. Higher-level fractals are layered on top of lower-level fractals whenever the latter cannot handle all the tasks that must be completed. Conversely, the goals of the organisation flow down from the main fractal that contains the entire company, becoming increasingly concrete level by level. Every fractal can regulate itself up to a certain degree.

Finally, holonic models view organisations as collections of “holons”, a term coined by [46] to mean something that is both a whole and a part. Koestler [45] observed that organisms were “not an aggregation of elementary parts”, but rather a “multi-levelled hierarchy of semi-autonomous sub-wholes” (the so-called holons). Every holon can be viewed as a whole formed by its sub-holons, and also as part of several higher-level holons. Holons delegate work on their sub-holons, work for the higher-level holons, and cooperate with the holons at the same level and their local environment. The set of all holons in the organisation is known as a “holarchy”, which can be viewed as a generalisation of a hierarchy.

Figure 2.5 illustrates the concept of a holon at level n of the holarchy, which is part of several holons at the $n + 1$ -th level and is formed from several holons at the $n - 1$ -th level. At the lowest level, a holon may simply wrap the control software for a physical device, so it can talk to the other holons. Higher-level holons may work at the cell, plant or enterprise level, providing more intelligence while operating with less restrictive time constraints. Holonic concepts have been used to model shop floor control systems, material handling systems, logistics systems and even entire enterprises [3, 52].

As an example of an application of holonic concepts in manufacturing, Brussel et al. [7] have proposed the Product-Resource-Order-Staff Architecture (PROSA) reference

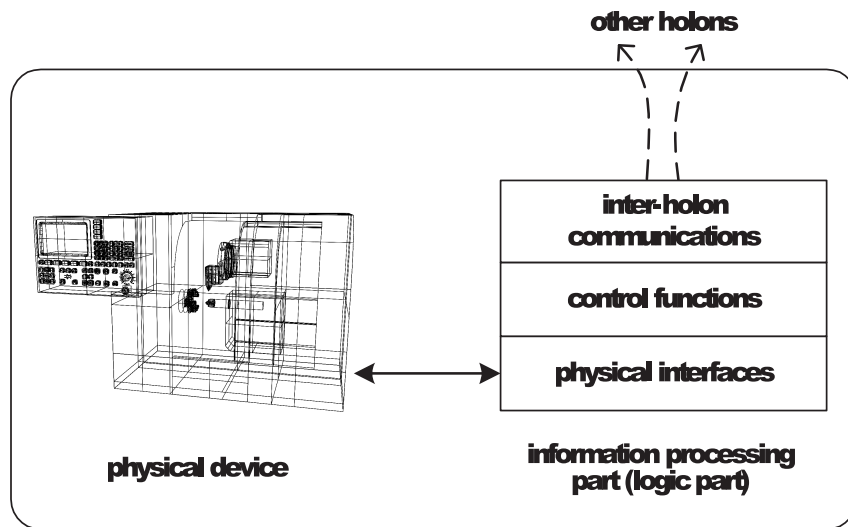


Figure 2.6. A physical holon (based on [52])

architecture for holonic shop floor control systems, identifying four kinds of holons:

- *Product holons* hold the process and product knowledge to assure the correct making of the product with sufficient quality.
- *Resource holons* contain a physical part (a production resource of the manufacturing system), and an information processing part that controls the resource. They may refer to some piece of equipment, a workstation, a cell, a job shop, or an entire factory, among other things.
- *Order holons* represent a task in the manufacturing system. It is responsible for performing the assigned work correctly and on time.
- *Staff holons* can act as external experts that give optional advice to the other holons.

This suggests that even though the bionic, fractal and holonic models are quite different, but they all view the enterprise as a dynamic network of agents with a certain degree of autonomy and ability to coordinate with other agents. One of the most important problems in putting these models into practice is creating the required communication channels across the business units. Information systems tend to be designed and implemented based on short-term requirements specific to each business unit, which may predate current Web-based integration initiatives. This poses many challenges while setting up temporary collaborations with external companies in order to take advantage of their skills and resources.

2.3 Enterprise integration

The previous section showed that the extended enterprise could be conceived as a heterarchy, a biological entity, a fractal entity, a holarchy or a combination of these. Nevertheless, this conceptual view would need to be realised in some concrete way across the people, processes and equipment available throughout the manufacturing enterprise.

As computers became increasingly affordable, there was a large push to use them to integrate all the information produced and required by existing manufacturing firms. This approach was called Computer Integrated Manufacturing (CIM), and specific implementations largely varied on their scope and features. Some of these systems operated at process level, while others handled entire plants. There were also issues due to the different ways in which the systems represented the organisation.

In the next subsections, several reference architectures for enterprise integration will be discussed. Most of them are closely related to manufacturing, either by design (GRAI, PERA, CIMOSA, ISA-95) or by their history (GERAM, ISO 19439/19440). TOGAF was derived not from the manufacturing industry, but from defense IT specialists, explaining its slightly different features. Architectures have been sorted largely by chronological order except for ISA-95, which is described last due to its unique focus on the integration of the business and manufacturing control functions of the enterprise.

2.3.1 GRAI Integrated Methodology (GIM)

One of the first approaches proposed to organise the information in these computer-based systems was the GRAI Integrated Methodology (GIM) by G. Doumeingts et al. [29]. More up to date descriptions are available at [13, 68], and a case study applying GIM is available at [22]. GIM is divided into two large steps:

1. First, conceptual Interrelated Activity, Result and Graph (Graphes et Résultats et Activités Interreliés or GRAI) models for the current enterprise and the desired future enterprise are built from user requirements. These models are divided into functional, decisional, informational and physical models.
2. Using these conceptual models, several sets of specifications are created for the organisation, its information systems and its manufacturing technologies.

GIM divides the enterprise into three subsystems:

Physical System Performs the actual work (i.e. manufacturing). The Physical System is modelled using IDEF0 process descriptions (see Section 2.4.1 on page 2.28).

Decision System Manages products and resources, schedules production and so on. Decision Systems are described using Graphes et Résultats et Activités Interreliés (GRAI) grids and GRAI networks, which will be briefly described below.

Information System Exchanges information and orders between the two previous systems. Information Systems are described using entity/relationship models [14].

Decision Systems are vertically divided into layers by the time horizon of the decisions involved (the “coordination” criterion). The topmost layer may handle strategical business decisions that span several years and are revised every year. In contrast, the bottom layer may only be concerned with the production for the next two months, while monitoring production week by week. The bottom layer is known as the “operating system”, since it interfaces with the Physical System and needs to respond in an event-driven fashion, rather than by time periods.

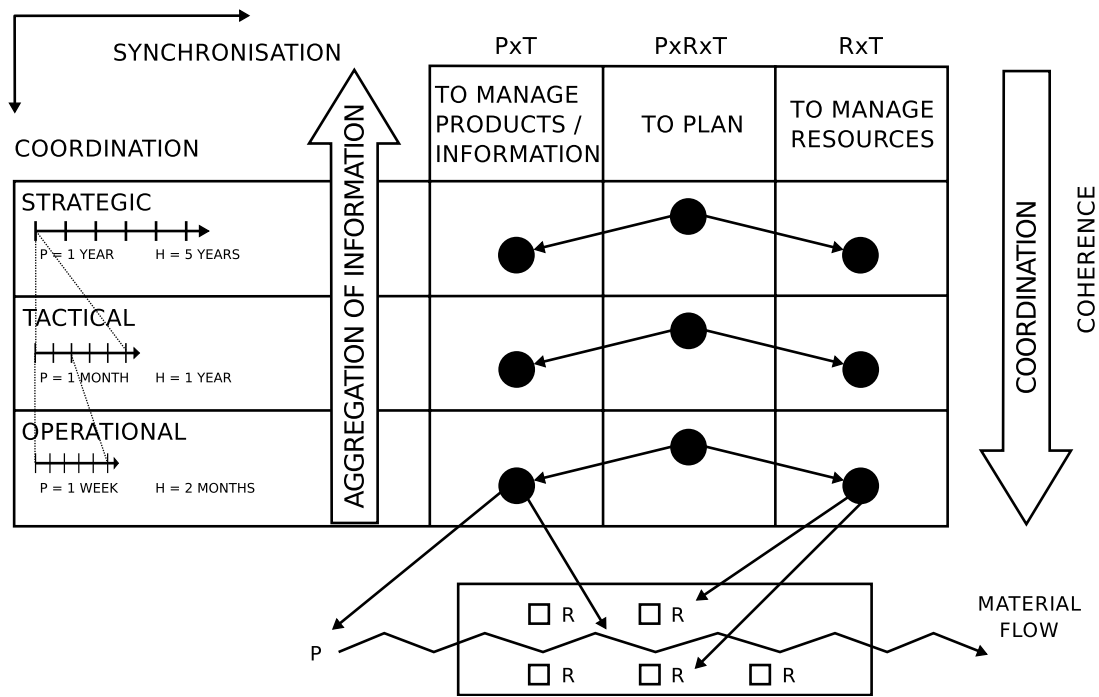


Figure 2.7. GRAI model for a Decision System [68]

Each layer is then divided into several functional areas by the “synchronisation” criterion: product management activities, planning activities and resource management activities. The Information System is designed to support the aggregation of information while ascending through the decision layers, and the coordination of the orders from the upper layers while descending.

Figure 2.7 illustrates the breakdown of the Decision System through the coordination and synchronisation criteria. H is the time horizon that must be considered when taking a decision, and P is the period that should be used for monitoring its results. The bottom edge of the figure shows the Physical System running the material flow through several manufacturing resources, in order to build the desired products.

GRAI can model Decision Systems using GRAI grids. These are quite similar to the table in Figure 2.7: they use one row for each layer in the Decision System, and separate columns for the product management, planning and resource management activities. However, GRAI grids also have columns for external and internal sources of information at each layer. Every cell in the table (except for the information sources) is known as a “decision centre”, a place where decisions are made by taking into account all information flows (represented as single arrows between cells) and decision frames (represented as thick, hollow arrows) that restrict the decisions that can be made.

By itself, GRAI grids can be used descriptively to identify issues within the overall management of an enterprise or a plant. However, they cannot be easily simulated due to their high level of abstraction and usage of natural language descriptions. GRAI nets can be used to describe the decision process in each decision centre in a more structured fashion.

2.3.2 Purdue Enterprise Reference Architecture (PERA)

The Purdue Enterprise Reference Architecture (PERA) was developed in a joint collaboration between the University of Purdue and a consortium of industrial companies [86], together with a methodology (the “Purdue method”) for modelling the enterprise with PERA. Some of the issues PERA set out to solve were excessive complexity in top-down CIM projects, lack of integration in bottom-up CIM projects and lack of personnel involvement, among others. These efforts started in 1986 and produced several other useful results, such as the Purdue CIM Reference Model by Williams [85], a collection of requirements and recommendations for implementing CIM in an enterprise.

PERA models the CIM Business Entity (the representation of the manufacturing firm itself, hereafter known as CBE) using a top-down approach, in which descriptions become increasingly detailed and concrete over time, until the plant is decommissioned. The first two layers (“concept” and “definition”) focus on the information and manufacturing requirements, which finally describe the information and manufacturing functional task networks. After descending to the “specification” layer, it is required to break down tasks into those that will be automated, and those that will be performed by humans. For this reason, the information and manufacturing architectures are bridged by the human and organisational architecture from the “specification” layer down to the last “operations” layer.

Figure 2.8 on page 2.16 summarises the steps involved in the Purdue method. This diagram is also known as the “PERA chime”, due to its bell-like shape. The method covers the entire lifecycle of the firm, from its general conception and overall mission to its replacement.

Similarly to GIM, PERA divides the enterprise into several layers of abstraction. However, its focus is turned more into describing the entire enterprise, rather than focusing on the way information is aggregated and distributed through the layers.

Figure 2.8 on page 2.16 shows that the main unit of decomposition in PERA is a “task module” (shown in Figure 2.9 on page 2.16). Task modules are essentially black boxes that represent a particular transformation process, whether it is a manufacturing step, the execution of a computer program or an action performed by a human. Tasks transform their inputs into a set of outputs which may go either into another task, or to a storage area. The behaviour of a particular task can be controlled through a set of external parameters. If a particular transformation step is deemed to be too complex, task modules may be broken up into simpler task modules. Task modules are interconnected by flows, which may represent data flows for the information architecture, or material and energy flows for the manufacturing architecture.

2.3.3 Computer-Integrated Manufacturing Open System Architecture (CIMOSA)

The Computer-Integrated Manufacturing Open System Architecture (CIMOSA) [47, 48, 49] was developed as part of a project of the European Strategic Programme for Research in Information Technology (ESPRIT), by the European CIM Architecture (AMICE). Similarly to PERA, it models the enterprise through the processes run in its day-to-day operation.

In order to keep models within a manageable size, CIMOSA suggests creating multiple

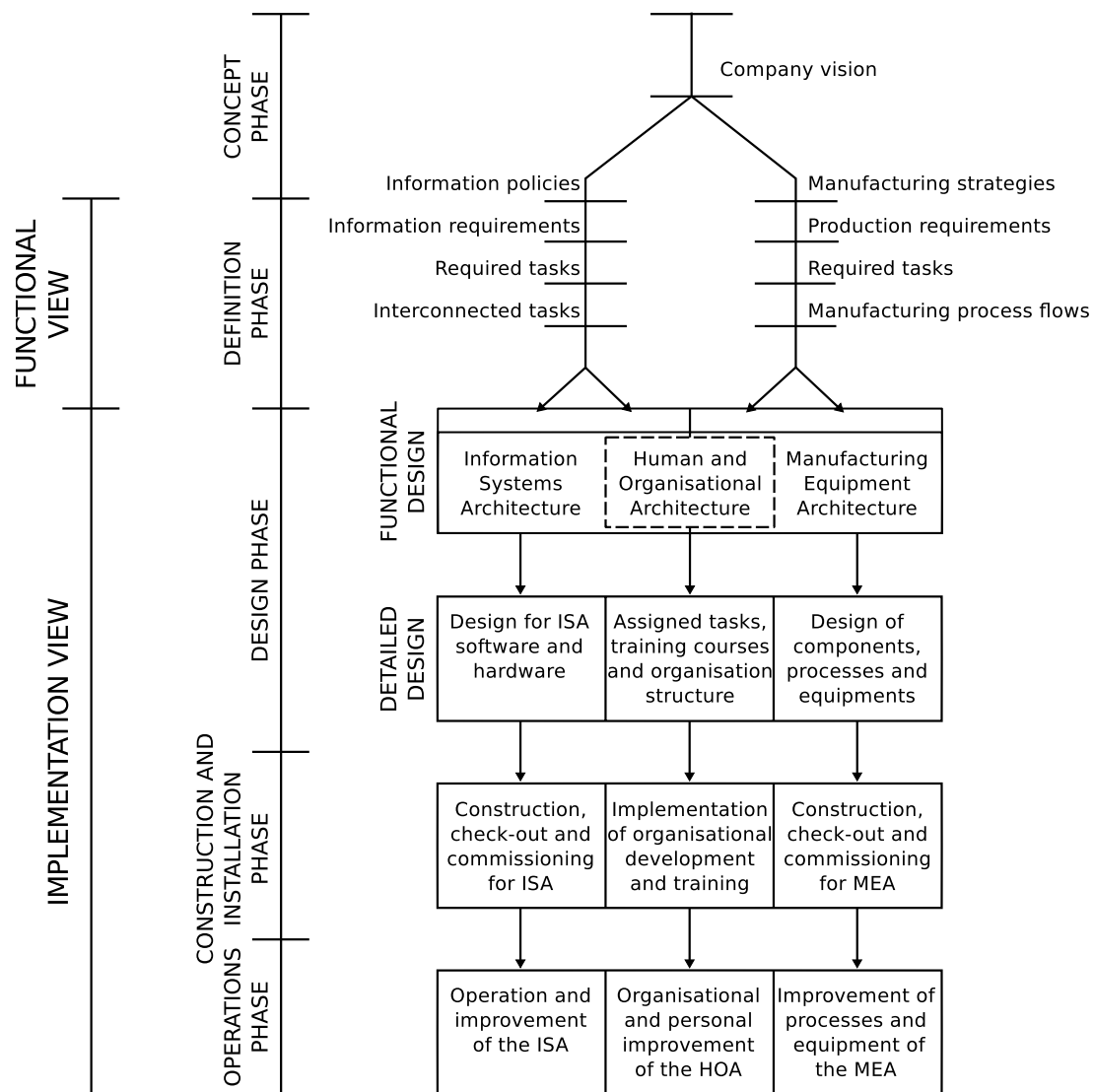


Figure 2.8. Purdue Method [86]

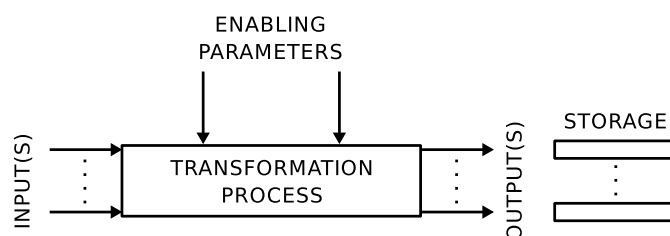


Figure 2.9. PERA task module [86]

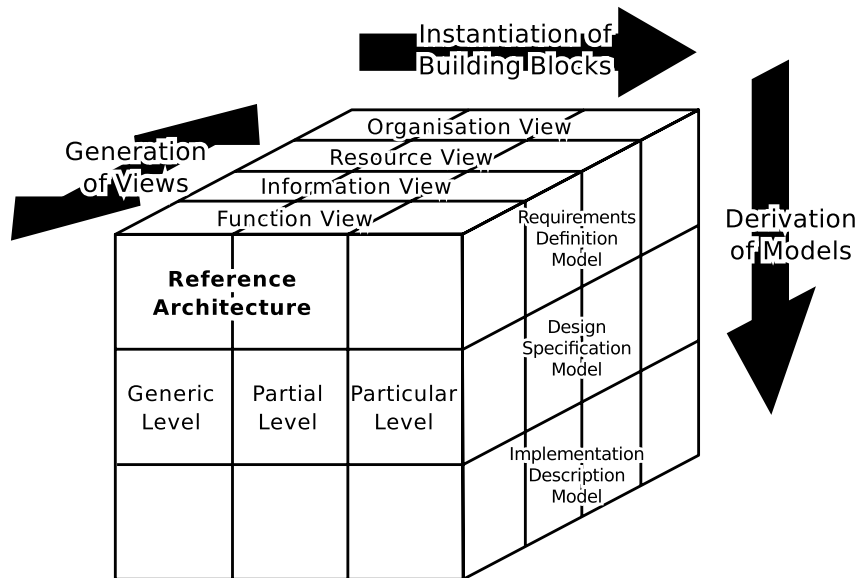


Figure 2.10. CIMOSA views [47]

views over three dimensions, as shown in Figure 2.10. Models can cover four kinds of information (organisation, resources, information and functions), three abstraction levels (requirements, design or implementation) and provide generic, partially specific or fully specific descriptions of a manufacturing firm.

CIMOSA provides modellers with a set of predefined constructs, which serve as a standard vocabulary for practitioners. According to Kosanke [47], Processes, Events and Enterprise Activities describe the functionality and behaviour of the enterprise operation. Inputs and outputs of Enterprise Activities (Enterprise Objects) define the information and resources needed. Organisational aspects are defined as Organisation Elements, which are structured in Organisational Units or Cells. Many of these concepts were standardised as ENV 12204:1996, later replaced by ISO 19440:2007 (see Section 2.3.5).

Figure 2.11 on page 2.18 shows a simple example of the top-down modelling process imposed by CIMOSA. The firm has three Processes, “Administration”, “Quality Assurance” and “Manufacturing” that send and receive Enterprise Objects (products, invoices, payments and orders) between the firm, its customers and its suppliers. The “Manufacturing” process is broken up into several lower-level Enterprise Activities, such as “Machine Part” or “Paint Part”, which are connected together according to a Behavioural Rule Set (BRS) that indicates when an EA is done and when the next EA can begin. Every EA transforms its inputs into a set of outputs and is controlled through external information, much like PERA task modules. In addition, the bottom edge is used to send the required resources into the EA.

2.3.4 Generalised Enterprise Reference Architecture and Methodology (GERAM)

Due to the large number of available reference architectures, the IFAC/IFIP Task Force decided to merge some of the most popular ones (namely, PERA, GIM and CIMOSA) into a single generalised reference architecture. This new architecture would be applicable

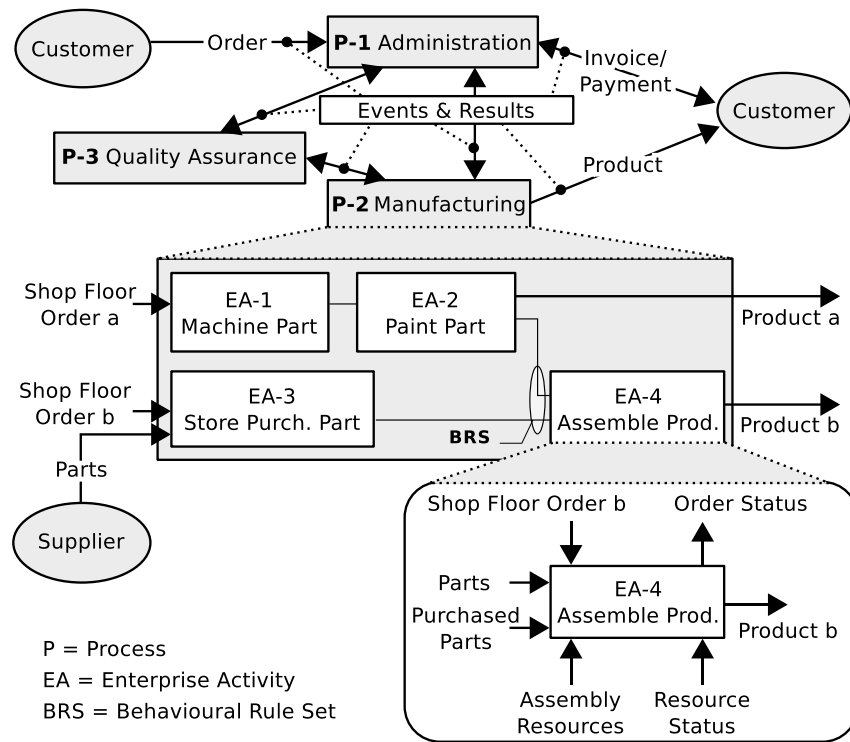


Figure 2.11. CIMOSA process modelling [48]

to any kind of enterprise, and not just to manufacturing enterprises.

From these efforts, the Generalised Enterprise Reference Architecture and Methodology (GERAM) [34] was produced. Version 1.6.3 of GERAM was published as an annex of ISO 15704 [39], which describes the general requirements that must be met by enterprise reference architectures and methodologies.

GERAM takes a more modular approach than PERA, GIM and CIMOSA. As shown in Figure 2.12, the reference architecture itself (GERA) is only one component of GERAM. Other components include the methodology, the modelling languages, reusable partial design or implementation models and the systems themselves. As Chen et al. [13] stated, GERAM is not yet another proposal for a reference architecture, but a framework meant to organise the existing enterprise integration knowledge.

GERAM inherits from CIMOSA the idea of dividing models into multiple views along several axes (as shown in Figure 2.10), but replaces the model derivation axis with a sequence of life-cycle phases much like that in PERA (see Figure 2.8). These phases are not mutually exclusive: several phases may be in effect at the same time. As in CIMOSA, there are function, information, resource and organisation views.

GERAM extends the explicit division in PERA of the enterprise architecture into its information systems architecture, human and organisational architecture and manufacturing architecture (now called the “mission support equipment architecture”). The boundaries between the human architecture and the other two architectures are now less clear-cut: there is an upper-bound of automation called “automatibility”, and a lower-bound called “humanizability”. The enterprise must decide upon a certain “extent of automation” between those bounds in the mission support and information architectures.

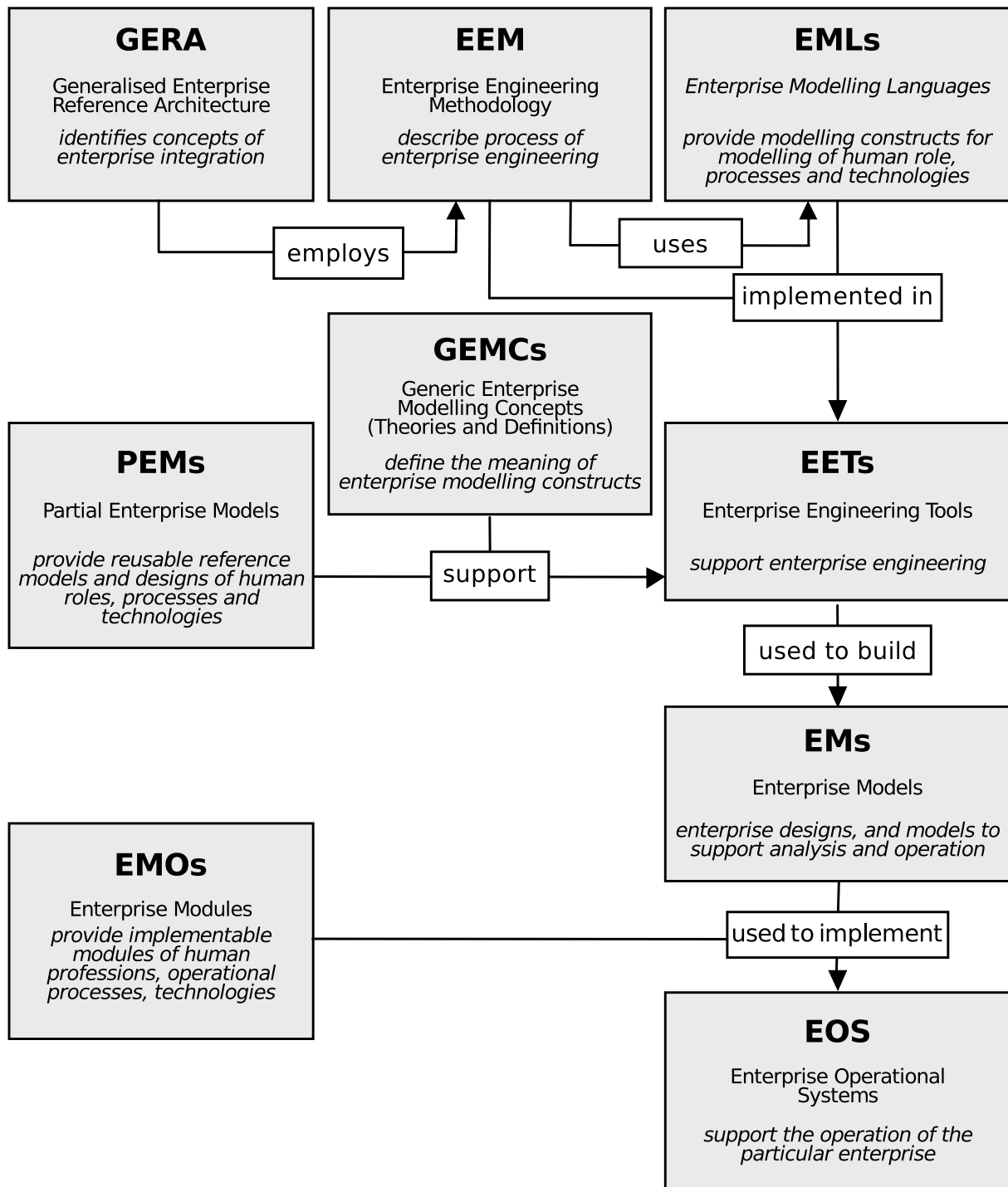


Figure 2.12. GERAM framework components [34]

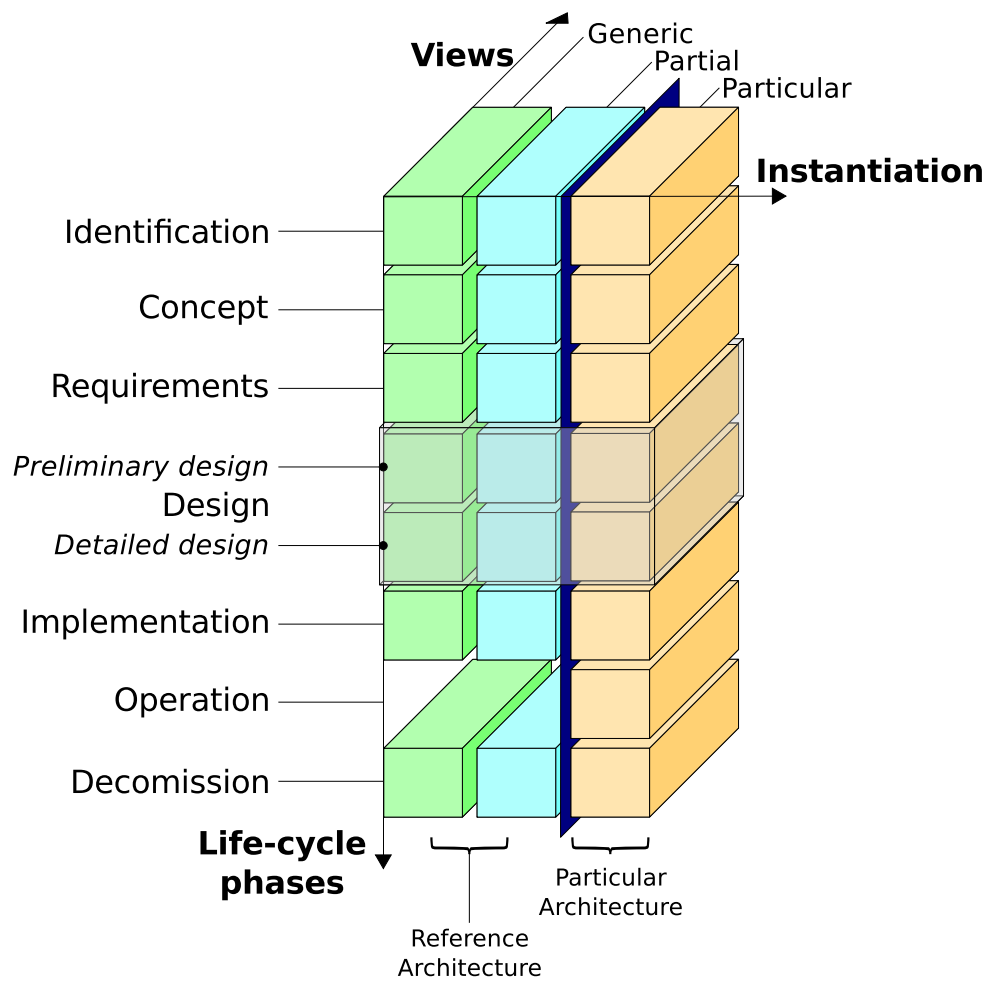


Figure 2.13. GERA Modelling Framework [34]. The organisation, resource, information and function views are not explicitly represented.

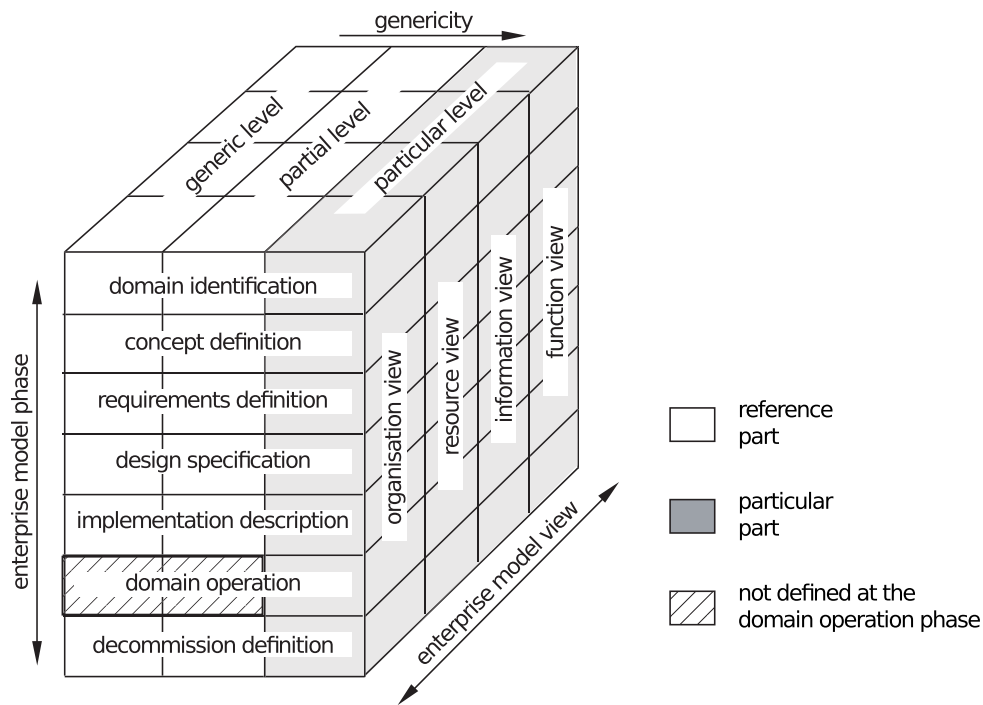


Figure 2.14. EN/ISO 19439 framework for enterprise modelling [41]

2.3.5 EN/ISO 19439 and EN/ISO 19440

ISO 15704:2006 had provided a normative set of requirements for enterprise reference architectures, and presented GERAM as a generic example. It was time to update the existing ENV 40003:1991 and ENV 12204:1996 standard enterprise architecture modelling framework and constructs (based on CIMOSA) with new versions based on ISO 15704. By combining elements from GIM, PERA, CIMOSA and GERAM, a new standard framework and set of constructs were published as EN/ISO 19439:2006 and EN/ISO 19440:2007, respectively [41, 42].

EN/ISO 19439 is a general modelling framework that provides a common vocabulary and streamlines GERAM (compare Figure 2.13 with Figure 2.14). It provides detailed descriptions of the meaning of each modelling dimension and their points. Annex B suggests using the Purdue method in the requirements definition phase, with the same sequence of business entity designation, mission statement, policy description, requirements specification and module, function and task listings.

EN/ISO 19440 provides a normative set of modelling constructs based on the EN/ISO 19439 framework. Most of these constructs are taken from CIMOSA, such as Business Processes (originally Processes), Behaviour Rule Sets and Enterprise Activities, among others. The Decision Centre construct is an adaptation of GRAI grids. EN/ISO 19440 also provides normative textual template-based notations and an informative metamodel based on UML, and suggests which constructs should be used at each modelling phase.

2.3.6 The Open Group Architecture Framework (TOGAF)

Aside from the GERAM and ISO 19439 lineage, there have been many other proposals for enterprise architectures. The Open Group Architecture Framework (TOGAF) is one

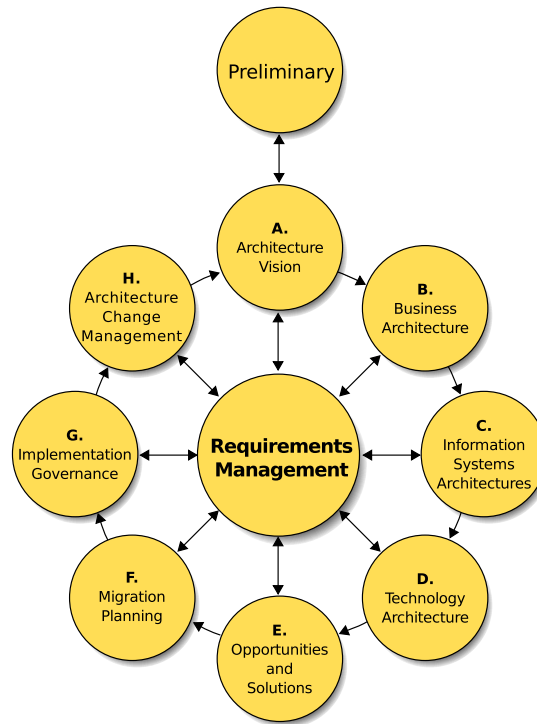


Figure 2.15. TOGAF Architecture Development Cycle [80]

of them [80]. According to the specification, the first version of TOGAF in 1995 was based on the Technical Architecture Framework for Information Management (TAFIM), developed by the US Department of Defense.

Among the previous specifications, TOGAF is closest to the GERAM approach: it provides a base set of concepts and guidelines for modelling, but it does not force specific modelling languages upon users. While TOGAF and GERAM are general-purpose reference architectures for enterprises in any industry, TOGAF lacks the manufacturing bias that GERAM inherited from its ancestors. In turn, TOGAF pays more attention to the information system, identifying the following four architecture domains:

- The *Business Architecture* defines the business strategy, governance, organisation and key business processes.
- The *Data Architecture* describes the structure of the logical and physical data assets and data management resources.
- The *Application Architecture* provides a blueprint for the individual applications to be deployed, their interactions, and their relationships to the core business processes.
- The *Technology Architecture* describes the logical software and hardware capabilities required to support the deployment of business, data and application services.

TOGAF also provides its own Architecture Development Method (ADM), which is defined as an iterative process divided into 8 phases, with an additional set-up phase and several guidelines for managing requirements over all phases. Figure 2.15 illustrates the structure of the ADM.

One of the main advantages of TOGAF over GERAM or ISO 19439/19440 is its much higher level of detail. It provides a considerable number of guidelines and recommendations on how to plan architecture migrations, identify opportunities, govern the implementation and handle specific aspects such as SOAs (see Section 3.1), risk management or security aspects.

TOGAF proposes storing all models in an Architecture Repository, which is a centralised location that organises every artefact, deliverable and reusable building block created within the organisation. Ideally, this would allow enterprises to create architectures for new purposes by reusing elements of previous architectures or industry-specific libraries. Artefacts in the repository are structured according to a metamodel, comply with the stored standards in its Standards Information Base and all changes are logged in its Governance Log. Best practices create new elements in its Reference Library.

The artefacts in the Architecture Repository can be examined through several views. TOGAF defines the Enterprise Continuum as such a view, providing a certain context and a general set of requirements. These shape an Architecture Continuum between generic architectures and specific architectures, much like the genericity axis in ISO 19439. In turn, these architectures support a Solutions Continuum between generic and specific solutions (implementations), which are finally deployed and become part of the context of the Enterprise Continuum. The Architecture Continuum and Solutions Continuum also help classify the artefacts, making the repository more manageable in large organisations.

Instead of the Enterprise Continuum, users may wish to use other ways to view their Architecture Repository, such as the Zachman Framework [89]. As a framework, it does not provide a methodology for creating the models: it is only intended as a way to organise the models, much like ISO 19439. The Zachman framework is defined as a 6 by 6 grid over 2 axes (the question to be answered, and the audience). Table 2.1 shows a simplified description of the framework.

2.3.7 IEC 62264 / ISA-95

While the previous specifications provided reference architectures for modeling the entire enterprise, ANSI/ISA-95 is focused on the integration between the manufacturing control and business IT systems in an organisation. Due to historical reasons, these IT systems have tended to grow separately from each other, and communication is difficult due to differing cultures and terminology. ISA-95 is set to bridge those differences.

ANSI/ISA-95 is a partial reference architecture according to the requirements in ISO 15704, and has been published as the IEC 62264 international standard. Roughly speaking, IEC 62264 provides three kinds of models for the organisation: hierarchical models of the activities within the organisation, data flow models with the information exchanged among these activities and a set of object models that describe the structure of the information and its meaning. Three parts are currently published:

- Part 1 includes most of the basic definitions [35]. IEC 62264-1 inherits the functional hierarchy of the enterprise from the Purdue CIM reference model (see Section 2.3.2), which is shown in Figure 2.16. Level 0 defines the physical processes. Level 1 defines the activities that sense and manipulate the physical processes. Level 2 controls and monitors the physical processes. Level 3 covers the workflow activities for producing the end products. Finally, Level 4 defines the business activities needed to manage a

Audience	Classif.		What	How	Where	Who	When	Why	Classif.	
									Models	
Executive Perspective			Inventory Identification	Process Identification	Distribution Identification	Responsibility Identification	Timing Identification	Motivation Identification	Scope	Contexts
Business Management Perspective			Inventory Definition	Process Definition	Distribution Definition	Responsibility Definition	Timing Definition	Motivation Definition	Business Concepts	
Architect Perspective			Inventory Representation	Process Representation	Distribution Representation	Responsibility Representation	Timing Representation	Motivation Representation	System Logic	
Engineer Perspective			Inventory Specification	Process Specification	Distribution Specification	Responsibility Specification	timing Specification	Motivation Specification	Technology Physics	
Technician Perspective			Inventory Configuration	Process Configuration	Distribution Configuration	Responsibility Configuration	Timing Configuration	Motivation Configuration	Tool Components	
Enterprise Perspective			Inventory Instantiations	Process Instantiations	Distribution Instantiations	Responsibility Instantiations	Timing Instantiations	Motivation Instantiations	Operations Instances	
Audience Names			Inventory Sets	Process Flows	Distribution Networks	Responsibility Assignments	Timing Cycles	Motivation Intentions		

Table 2.1. Zachman Framework for Enterprise Architecture [89]

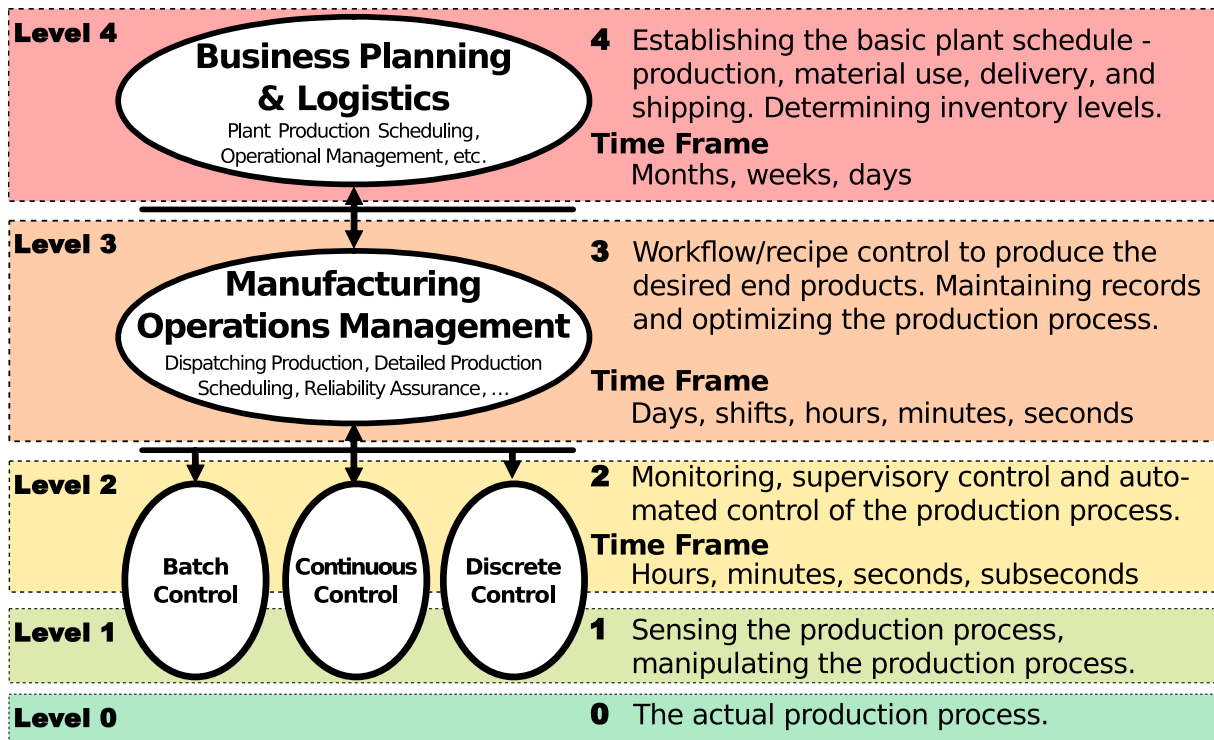


Figure 2.16. ISA-95 activity hierarchy. This is the extended version with additional explanations from Part 3 [38], rather than the original version from Part 1 [35].

manufacturing organisation.

IEC 62264-1 defines two more hierarchical views of the organisation: equipment hierarchies and decision hierarchies. Equipment hierarchies divide the physical assets of the enterprise into sites, areas, work centres and work units, as shown in Figure 2.17. Level 4 activities usually deal with the entire enterprise or a site, while level 3 activities deal with a single site at most. Decision hierarchies classify decision-making activities by what they relate to (products, resources or schedules) and by their scope (from long-term enterprise-wide decisions to short-term decisions about a single action).

After describing these hierarchical views, Part 1 focuses on the interface between levels 3 and 4. Figure 2.18 shows the ISA-95 functional enterprise/control model, which lists all the high-level activities which exchange information between these two levels. The activities outside the thick dashed line belong to level 4, the ones inside belong to level 3, and the ones that straddle the line have some functions in level 3 and some functions in level 4. The rest of the specification presents a set of simple Unified Modelling Language (UML) class diagrams describing the contents of these information flows in plain English.

- Part 2 is conceptually simple but just as long. It lists the attributes of each of the UML classes from Part 1, describing their valid values and meanings [36]. Incidentally, the World Batch Forum (currently “WBF, The Organization for Production Technology”) created the Business to Manufacturing Markup Language (B2MML) [88] from parts

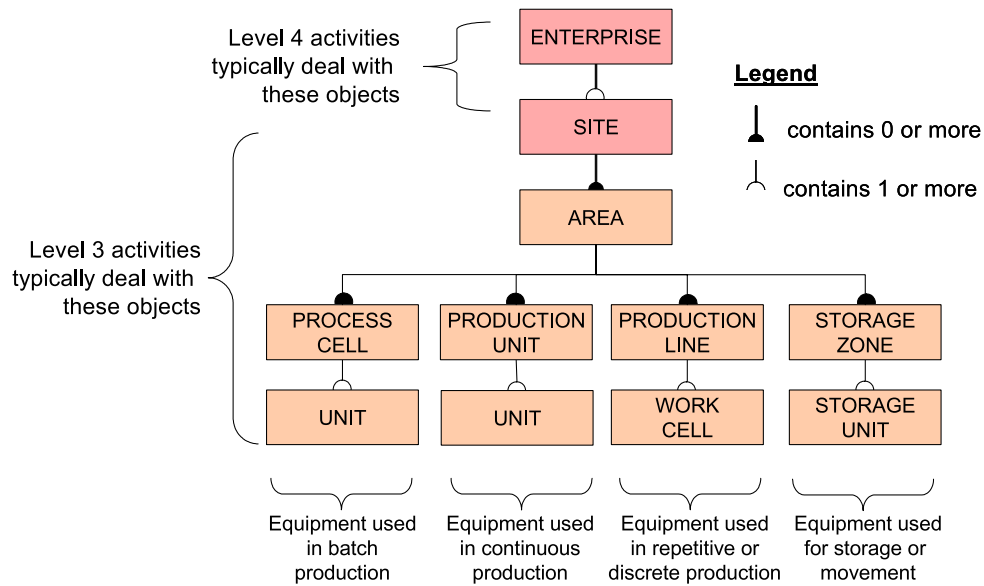


Figure 2.17. ISA-95 equipment hierarchy, initially defined in Part [35] and then extended with storage zones and units in Part 3 [38].

1 and 2 of ISA-95. B2MML implements the ISA-95 data model using XML Schema (see Section 3.1.2.3) for its use in system integration projects.

- Part 3 focuses on the activities within level 3 [38], the information flows between them and the decisions taken during their execution. It can be used as a generic blueprint of the business processes required to manage a manufacturing enterprise. This part also completes the specification in several significant ways: for instance, a decision is now valid during its *horizon*, and is reviewed periodically or when a certain event happens.

All the manufacturing operations management activities are divided into four groups: production, maintenance, quality and inventory management operations. In turn, each kind of management operation must handle four categories of information: what can be done (e.g. production or quality testing capabilities), what to do and when (production schedules, maintenance requests), how to do it (product definition, quality test definition) and how it was done (maintenance response, inventory response).

The standard provides a standard template for each of the four groups of manufacturing operations management activities, as shown in Figure 2.19. This basic template is then extended with the specific details of each activity group and the relevant information flows to the entities in levels 1 and 2.

2.4 Process modelling

The previous section discussed several specifications that aimed to model the entire enterprise by combining models covering multiple perspectives. This section will present several modelling notations used specifically to model the business and/or manufacturing processes run by the enterprise.

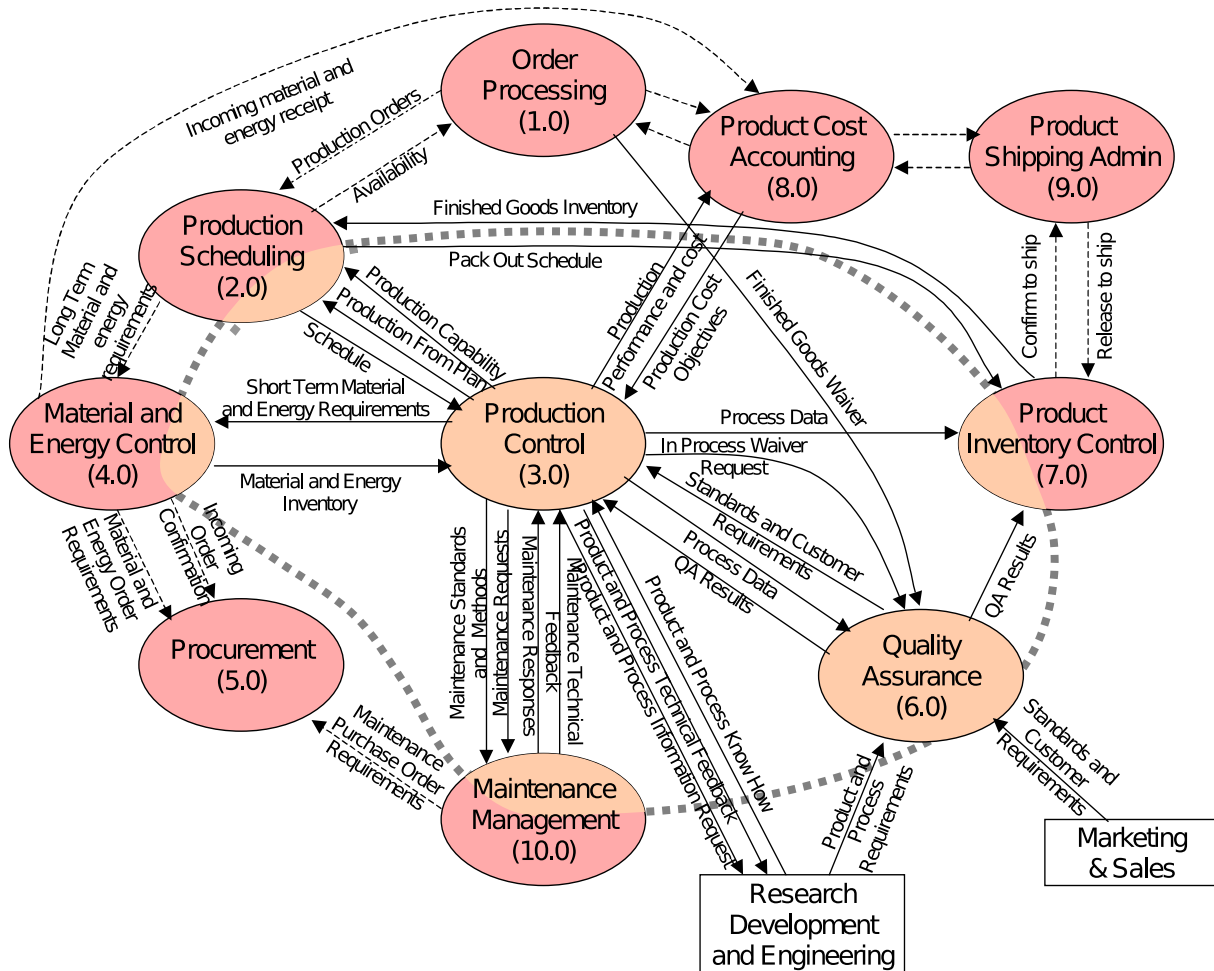


Figure 2.18. ISA-95 functional enterprise/control model [35]. The activities within the thick dashed line belong to ISA-95 level 3, and those outside the line belong to ISA-95 level 4. The activities that straddle the line have some functions in level 3 and the rest in level 4.

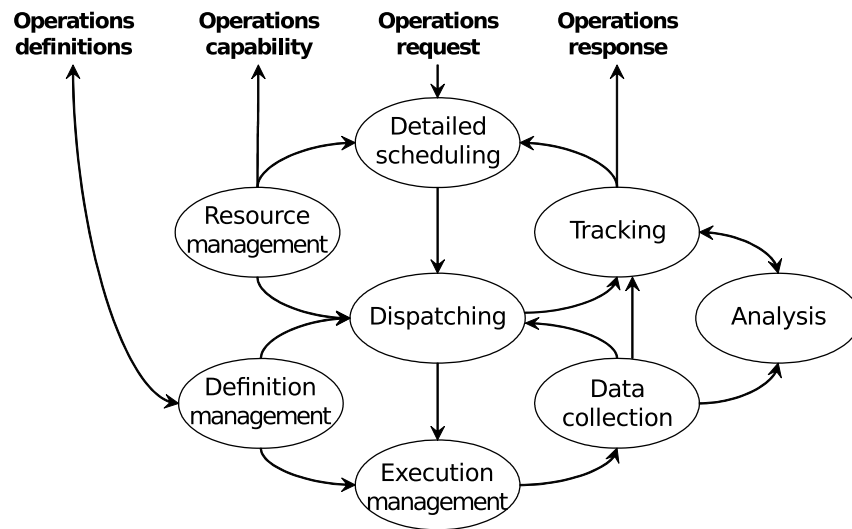


Figure 2.19. ISA-95 generic activity model for a group of manufacturing operations management activities [38]. This model is specialised for each of the production, maintenance, quality and inventory operations management activity groups.

Several notations have been used for this purpose, such as IDEF3, the Process Specification Language (PSL) or the Value Stream Mapping (VSM) notation. More recently, the Business Process Modelling Notation (BPMN) 2.0 standard from the Object Management Group (OMG) has been proposed for modeling business processes, using three kinds of views: collaborations, processes and choreographies. BPMN is intended as a bridge between business process design and process implementation [64]. It has gained considerable momentum in the recent years, with over 73 implementations by various vendors.

This section will describe some of these notations and position BPMN 2.0 among them by using a small case study. The survey by Aguilar-Savén [2] covers a more extensive range of flow-based notations, but it predates BPMN 2.0. Initial work on PSL produced an extensive comparison of the capabilities of the notations available at the time [44].

2.4.1 Integrated DEFinition for Process Description Capture Method (IDEF)

According to the original report, IDEF3 “was created specifically to capture descriptions of sequences of activities” [57]. IDEF3 uses two kinds of models: process schematics and object schematics. Process schematics describe the valid sequences of the Units of Behaviour (UOBs) in the process. Object schematics describe the kinds of objects present in the system, their relationships and their state transitions. Node and link shapes for IDEF3 process and object schematics are shown in Figure 2.20.

Process schematics represent the UOBs as boxes with textual labels and unique identifiers. Precedence links specify valid sequences of UOB activations. There are two types of precedence links: simple and constrained. A simple precedence link from A to B only indicates that whenever A and B both happen, A must happen before B. A may happen and not B, B may happen and not A, or any number of UOBs not included in the process

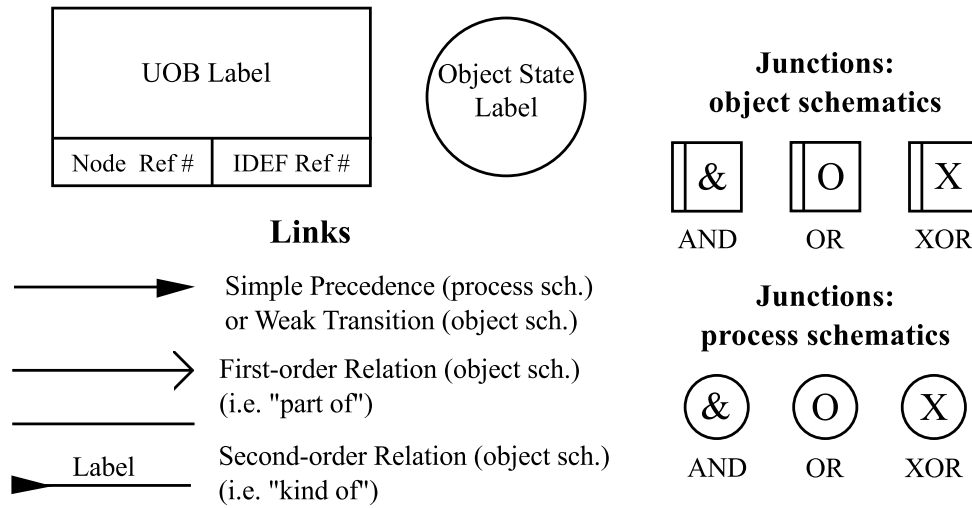


Figure 2.20. Selected subset of the IDEF3 notation

schematic may happen between A and B. Constrained precedence links can further limit the valid possibilities. Finally, junctions can split or join paths. AND junctions activate or join all related paths, OR junctions only some, and XOR junctions exactly one.

Object schematics represent the possible states for each object in the system. Links relate different objects, represent their state transitions or classify them. A state transition from A to B means that object b can only be in state B after object a has been in state A. Object a may be the same as object b or not. Users set conditions on transitions or states by linking them to UOBs from the process schematics.

IDEF3 allows for a hierarchical decomposition of both process and object schematics: starting with a high-level view, modelers can “drill down” into the detailed descriptions. It also allows modellers to indicate where information is “hidden” about parts, object categories and other constructions.

2.4.2 Process Specification Language (PSL)

ISO 18629:2004, widely known as the Process Specification Language (PSL), is a textual notation for describing manufacturing processes [40]. Its goal is to allow different applications to exchange process data. To achieve that interoperability, PSL is organized as an ontology of concepts, related to each other using axioms and definitions. PSL is organized into several layers. A more in-depth description of PSL is available at [5].

The main concepts in PSL constitute the PSL-Core. There are four kinds of entities in PSL processes: activities, objects, activity occurrences and timepoints. Activities can have zero or more occurrences. Timepoints are linearly ordered from a timepoint before all others in the past (*inf-*), to a timepoint after all others in the future (*inf+*). Every activity occurrence and object happens or exists between two timepoints.

The next layer in PSL is the Outer Core, with additional concepts and definitions that are commonly used, such as sub-activities: activities nested inside others. The PSL-Base layer extends the PSL Outer Core with more specialized terms, such as durations (intervals between timepoints) and resource handling, among others. The outermost layer only includes optional extensions.

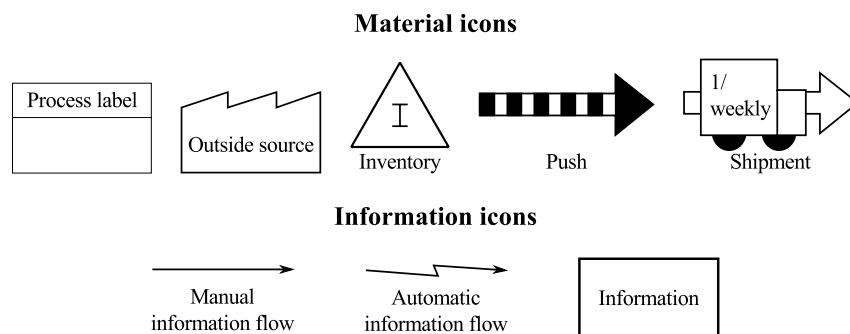


Figure 2.21. Selected subset of the Value Stream Mapping notation

2.4.3 Value Stream Mapping (VSM)

Lean manufacturing strives to reduce costs and increase flexibility by removing waste (muda) from the manufacturing process. The Value Stream Mapping (VSM) notation helps identifying issues and creating improvement plans to reduce waste. A “value stream” contains all the actions required to bring a product to the customer: part and raw material retrieval, intermediate transformation and storage operation, transport and communication, among others. This section will refer to the VSM workbook by Rother and Shook [71] from the Lean Enterprise Institute, focused on the production flow.

Factory icons represent external plants. Incoming and outgoing shipments are represented using a truck icon and a broad arrow. The manufacturing process is divided into process boxes: each box is a sequence of steps in which materials flow continuously. Elements may be connected by information flows (regular, electronic or “go see”) or material flows (push, pull, FIFO or sequenced pull). Material flows usually indicate the kind of inventory handling involved: accumulated inventory, supermarkets or buffer stock. Specific icons are available for load-leveling (“heijunka”) boxes, “kanban”-based systems, operators and possible improvements (“kaizen” bursts). Some of the graphical icons in VSM are listed in Figure 2.21.

2.4.4 Business Process Modelling Notation (BPMN)

In the recent years, interest in modeling business processes for re-engineering, simulation and execution has steadily increased. Nowadays, the most popular notations are all based on “workflows”. Workflows are defined in [87] as “the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules”. Normally, these workflows are described in a specific language, supported by a Workflow Management System (WFMS). The WFMS communicates with the participants (both humans and software entities) to execute the workflows and monitor them.

There are several workflow modelling languages currently in use. Some of them are designed to be directly run and monitored by a WFMS automatically, such as the Web Services Business Process Execution Language (WS-BPEL) 2.0 [63]. However, these notations can be quite hard to grasp for business analysts: simply reading the workflows already required advanced software development skills.

The proliferation of low-level notations for process execution motivated the creation of BPMN as a high-level notation which could be used by both business analysts and

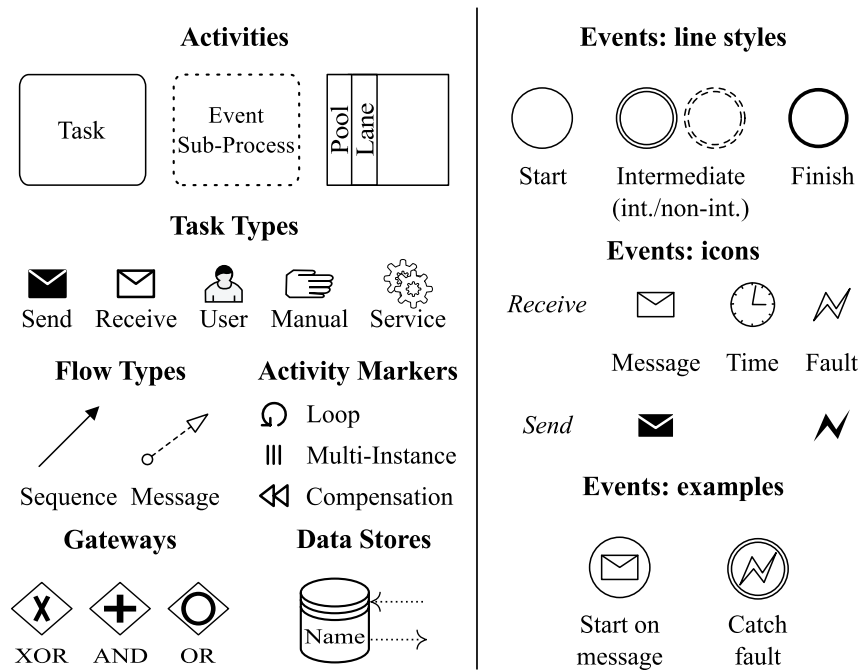


Figure 2.22. Selected subset of the BPMN 2.0 notation

software developers. In fact, the first versions of BPMN were only intended as graphical notations for the WS-BPEL standard. BPMN 2.0 has added its own formal execution semantics based on Petri networks and several file formats to the specification, making it usable both for process design and process enactment [64]. Part of the notation is shown in Figure 2.22.

Activities represent units of work in the process. Activities may have sub-processes describing them in more detail. Some of these sub-processes may be started in response to an event. Activities performed by different stakeholders will be usually placed in different pools, which may be further divided into lanes. The type of a task is noted by decorating it with one or two markers in the upper left corner. BPMN allows three types of markers: loop markers to indicate that the same activity will be run several times in sequence, multi-instance markers to allow multiple executions by different performers (normally in parallel), and compensation markers to indicate that the activity will “compensate” or undo the effects of previous activities. Loop and multi-instance markers cannot be placed at the same time on a single activity.

Events are situations to which the BPMN process reacts. Events are drawn as circles: the line style of the circle indicates if it is a start event that instantiates a process, an intermediate event that the process waits for, or a finish event which concludes execution. Inside the circle, an icon indicates what kind of event is handled.

Activities and events are connected together through flows and gateways. Message flows model the exchanges of information and material between the participants, and sequence flows control the execution of the activities. Sequence flows may optionally converge or diverge through gateways, similar to IDEF3 junctions. Activities may also query or modify data stores: this is represented with a dotted arrow from the activity to the data store or from the data store to the activity, respectively.

Intermediate events may also be placed at the boundary of an activity to respond to

specific situations during their execution. If the event is interrupting (drawn using a double solid line), it will stop the activity when triggered and continue execution through its single outgoing sequence flow. If the event is non-interrupting (drawn using a double dashed line), the sequence flow will be activated without stopping the original activity.

2.4.5 Comparison through a case study

In the previous sections, IDEF3, PSL, VSM and BPMN 2.0 were presented. This section shows how to use each notation to model a hypothetical manufacturing process described in natural language. In the next section, the models will be used to compare the notations.

2.4.5.1 Textual description

The company under study (“Company A”) receives tobacco and cellulose acetate and produces cigarettes. Tobacco preprocessing consists of several steps and slightly varies from product family to product family. 180–200kg boxes of raw tobacco are regularly received from external suppliers. First, the moisture in the raw tobacco is increased and casings are added. Next, tobacco is blended, cut, compressed and packaged. Optionally, the tobacco may be “expanded” before packaging to produce the “light” variants.

Cigarette filters are produced from cellulose acetate tows, separating the fibers before adding a plasticizer and cutting the filter rods into individual filters. After letting the filters harden on trays, they are sent to the cigarette making machine.

Filters and processed tobacco are received by another department, which wraps the tobacco and adds the filters, joining the cigarette with the filter using tipping paper. These cigarettes are then packed into boxes, which are bought by distributors and finally sold by retailers.

It is important to note that Company A recently joined a larger group and needs to synchronize its in-house information system with the SAP R/3 installation in use within the group. This includes inventory levels, manufacturing reports and production forecasts. Shipments from suppliers and to distributors are handled by an external company, part of the same group.

2.4.5.2 IDEF3 model

Figure 2.23 is the IDEF3 object schematic for the manufacturing process. Objects represent intermediate products, from raw materials up to packaged goods. The UOB boxes have a slightly different notation, as they refer to UOBs in the omitted process schematic. They describe the process steps required for each state transition. There are two types of processed tobacco (regular and expanded), and therefore two types of cigarettes (regular and “light”). Most UOBs have a single digit: their contents have not been expanded. However, as an example, the “Make filters” UOB (#2) has been expanded into four nested UOBs, with identifiers from 2.1.1 to 2.1.4. Additionally, the “Filters” object node has a different line style and is decorated with a “C”, indicating there are several types of filters not shown in the diagram.

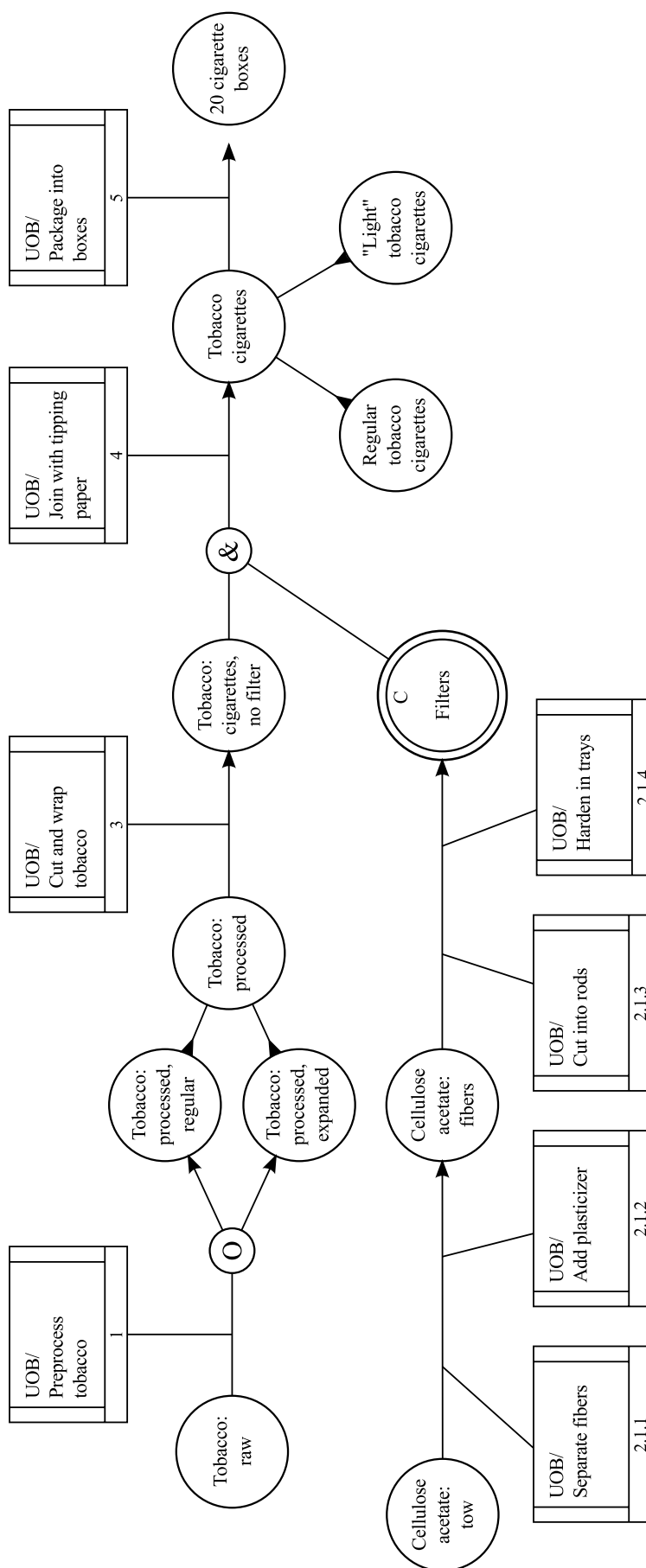


Figure 2.23. IDEF3 model for the notation comparison case study

Listing 2.1 Process Specification Language fragment describing the activity sequences dedicated to preprocessing tobacco

```
(forall (?opt)
  (implies (oof ?opt PreprocessTobacco)
    (exists (?oim ?oac ?orc ?ob ?orb ?ocut ?ocomp)
      (and (oof ?oim IncreaseMoisture) (oof ?oac AddCasings)
        (oof ?orc RefillCasings) (oof ?ob Blend)
        (oof ?orb RefillBlender) (oof ?ocut Cut) (oof ?ocomp Compress)
        (sao ?oim ?opt) (sao ?oac ?opt) (sao ?orc ?opt) (sao ?ob ?opt)
        (sao ?orb ?opt) (sao ?ocut ?opt) (sao ?ocomp ?opt)
        (mpr ?oim ?oac) (mpr ?oac ?orc) (mpr ?oac ?ob)
        (mpr ?ob ?orb) (mpr ?ob ?ocut) (mpr ?ocut ?ocomp))))))
```

Listing 2.2 Process Specification Language fragment describing the machines and materials used at each step in Listing 2.1

```
(reusable Moisturizer IncreaseMoisture)
(possibly_reusable CasingSpreader AddCasings)
(possibly_reusable Blender Blend)
(wearable Cutter Cut)
(reusable Compressor Compress)
(consumable DryTobacco IncreaseMoisture)
(creates IncreaseMoisture MoistTobacco)
(consumable MoistTobacco AddCasings)
(creates AddCasings MoistTobaccoWithCasings)
(consumable MoistTobaccoWithCasings Blend)
(creates Blend BlendedTobacco)
(modifies Cut BlendedTobacco)
(modifies Compress BlendedTobacco)
```

2.4.5.3 PSL description

Due to space constraints, only included the fragment dedicated to preprocessing the tobacco has been included. To simplify the discussion, it is shown by parts. `occurrence_of` has been abbreviated to `oof`, `subactivity_occurrence` to `sao` and `min_precedes` to `mpr`, to save space.

Listing 2.1 describes the activities and subactivities dedicated to preprocessing tobacco and how they are ordered. These constraints ensure the occurrences of the subactivities in “Preprocess Tobacco” go in the order “Increase Moisture”, “Add Casings”, “Blend”, “Cut” and “Compress”. In addition, “Refill Casings” should happen some time after “Add Casings”, and “Refill Blender” some time after Blend.

Listing 2.2 describes the machines and materials used at each step, using PSL resource theory. Resources are described in terms of how an activity using a resource affects other activities which require that resource. Reusable resources can be always used after the activity which uses them completes. Possibly reusable resources require that a setup

activity completes before they can be reused. Wearable resources may not be usable at some point in the future. Consumable resources can never be reused. Finally, an activity may also create or modify a resource.

2.4.5.4 BPMN 2.0 model

Figure 2.24 is a BPMN 2.0 model of the manufacturing process. The model is divided into several lanes: one for each participant in the process. Lanes do not need to represent every action taken by a participant: for instance, this diagram only shows the activities from the parent company and in-house IT directly related to this manufacturing process. The lane for the logistics company is completely empty: all the model shows is that the plant sends shipment requests to it after a batch is done.

The model indicates that the plant receives every day the batches to be produced, and repeats the basic manufacturing process for each of them. Repetition in BPMN 2.0 is modelled by marking the repeated activity (“Produce batch”) with a small circle-shaped arrow. The contents of “Produce batch” are very similar to the IDEF3 process schematic from which Figure 2.23 was produced. The BPMN model adds the capability to model the messages sent to the other participants. An event-based subprocess (marked with a dashed rectangle with rounded corners) indicates that when there is a fault, a message is sent to the in-house IT system notifying that the manufacturing of a certain batch was aborted.

2.4.5.5 VSM model

Figure 2.25 presents a VSM schematic describing the different material and information flows in the plant. Suppliers provide the required tobacco and cellulose acetate tows once a week and these are pushed through the process, which performs 2 weekly shipments of cigarette boxes. There are two information systems communicating with the plant: an in-house system sends daily orders to the tobacco preprocessing area and weekly orders to the filter manufacturing area, and receives regular notifications about the shipments. The SAP/R3 system from the parent company sends weekly manufacturing schedules and receives periodic production and inventory status reports.

2.4.5.6 Comparison results

In the previous section, the same manufacturing process has been described from several viewpoints, using IDEF3, PSL, VSM and BPMN 2.0. This section will compare the expressive capabilities of these notations for several important aspects in manufacturing processes. Table 2.2 summarises these results.

All the notations allow for defining valid sequences for the tasks in the manufacturing process. VSM uses very high-level tasks, dividing the process only where continuous flow is interrupted. IDEF3 and BPMN 2.0 model sequences of activities, which can diverge into different paths or converge into one path using junctions (IDEF3) or gateways (BPMN). BPMN can also describe what to do if something goes wrong (faults), how to undo changes (compensation) and how to respond to signals. Though IDEF3 and BPMN 2.0 allow unspecified activities to be inserted between those in the models, PSL is the most flexible notation for activity sequencing, due to its use of precedence relations.

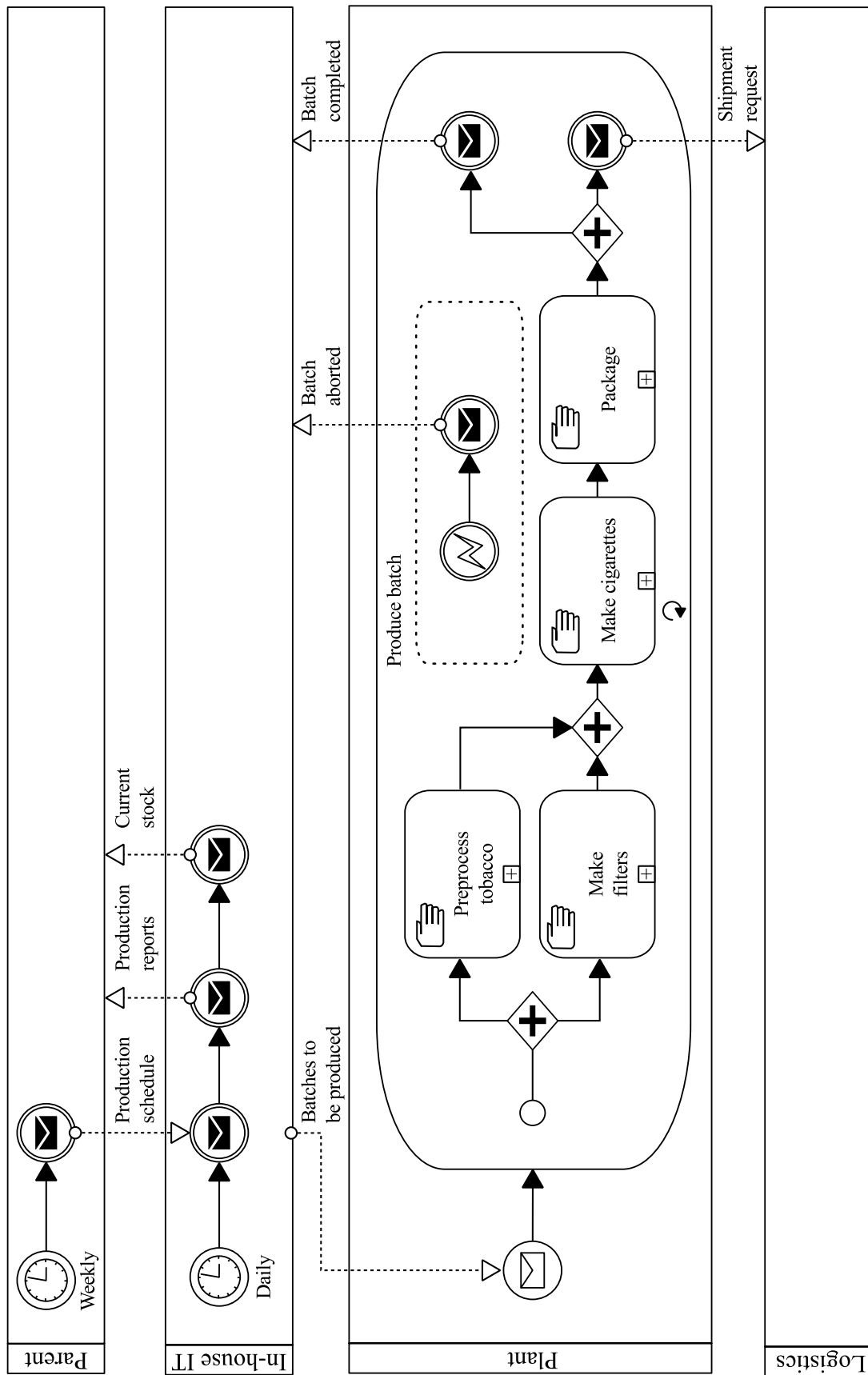


Figure 2.24. BPMN 2.0 model for the notation comparison case study

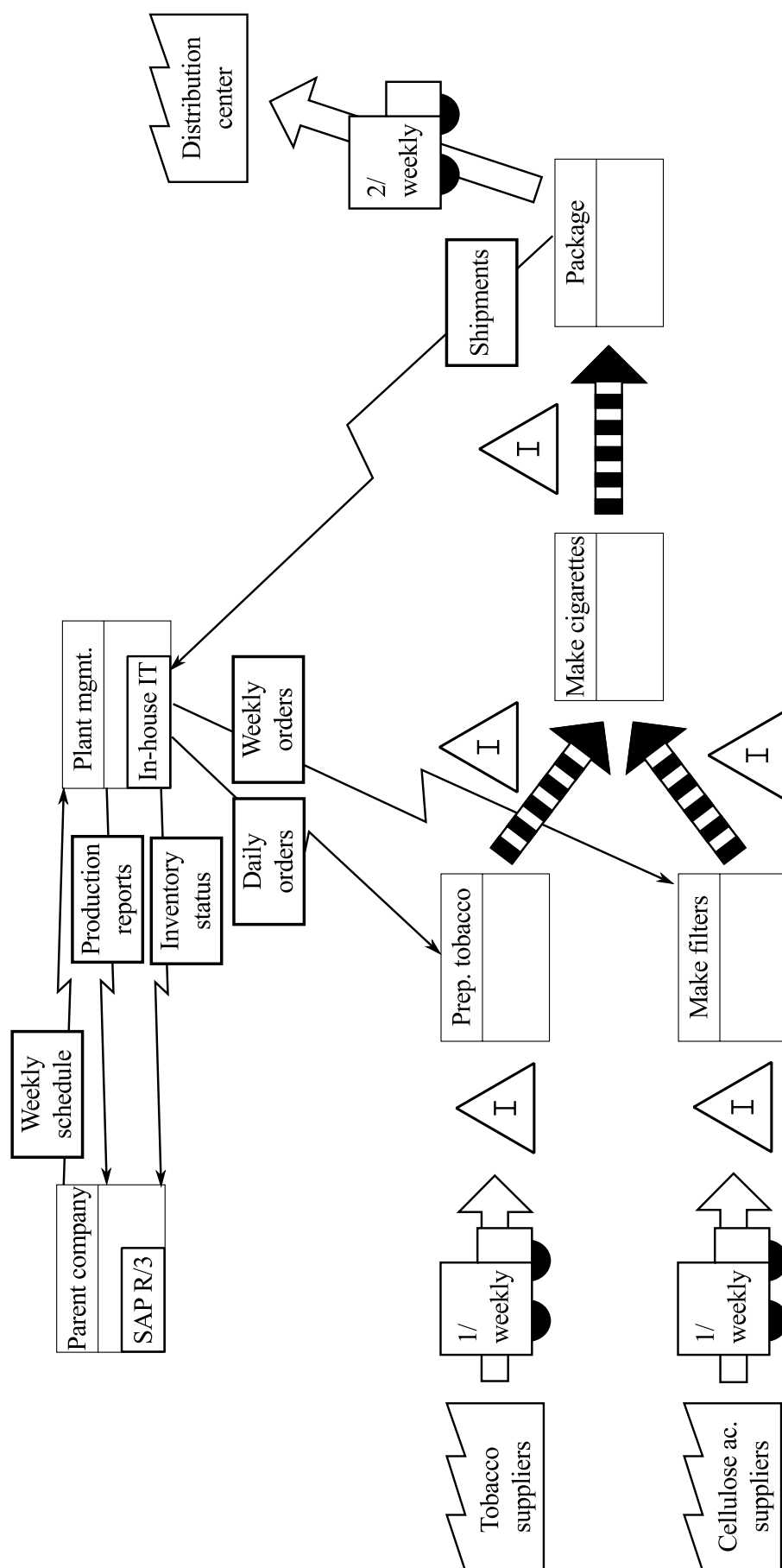


Figure 2.25. VSM model for the notation comparison case study

	IDEF3	PSL	VSM	BPMN 2.0
Activity sequences	Fine-grained (control flows)	Fine-grained (precedence constraints)	Coarse (material flows)	Fine-grained (control flows, events)
Timing constraints	Implicit (text)	Explicit (durations)	Implicit (text)	Explicit (alarms)
Machine-operator assignments	Implicit (objects)	Explicit (resources)	Implicit (data boxes)	Implicit (pools)
Material flows	Implicit (object transitions)	Explicit (resources)	Explicit	Implicit (messages)
Information flows	Needs IDEF0/IDEF1X	Needs extensions	Explicit, no internal structure	Explicit, relies on extensions for internal structure

Table 2.2. Differences between the selected process modelling notations

IDEF3 and VSM do not explicitly model timing constraints: they can only be emulated through textual descriptions. BPMN 2.0 allows for setting alarms at certain times, frequencies or delays, as shown in the models. PSL includes a rich set of theories for the duration of activities.

Machine/operator assignments can be emulated in IDEF3, relating the object node with the machine to the object node. VSM does not model assignments and only includes the parameters of the process which affect material flow, such as changeover or cycle time. BPMN does not explicitly model machine/operator assignments, but they can be emulated using pools and lanes if desired. PSL models machines and operators and resources, and includes several theories for describing constraints on them.

Material flows can be emulated in IDEF3 using state transitions between object nodes, as in Figure 2.23. BPMN cannot accurately model continuous material flows, but can emulate material flow in discrete manufacturing through messages with the part information. Material flows can be explicitly modeled in VSM. PSL models normally describe materials as consumable or renewable resources.

Information flows cannot be described with a single IDEF3 model: supporting IDEF0 and IDEF1X models will be usually required. PSL does not model the information exchanged between the manufacturing steps directly: however, a new extension has been proposed for that [4]. VSM models information flows directly, but does not provide any formal mechanisms to describe their internal structure. BPMN explicitly models the messages exchanged between each of the participants, but relies on vendor-specific extensions to describe the structure of the messages.

From the above observations, it can be concluded that BPMN 2.0 can be seen as a superset of IDEF3 process schematics, adding explicit support for modeling the participants in the process, event handlers and message exchanges. However, BPMN cannot model the

existing objects and their transitions, like IDEF3 object schematics can.

As a constraint-based notation, PSL has more expressive power than BPMN. However, it is also much harder to use than BPMN, and there are fewer tools for it: only Tau and Vampire, two theorem provers. Messaging is still not part of the PSL standard: once it is included, it would be interesting to translate BPMN to PSL and then enhance the PSL description.

VSM is a much simpler notation than BPMN and only provides a very high-level picture of the process, focusing on the material and information flows rather than the exact sequence of operations. For this reason, VSM is observed to complement BPMN: the former is a quick pen-and-paper tool for iterative process improvement, and the latter is for detailed process design and enactment.

As it is, BPMN 2.0 is recommended for two areas: describing the information-intensive activities which support the manufacturing process, and describing repetitive manufacturing processes with few variations. For describing families of interrelated manufacturing processes with high degrees of flexibility, PSL would be the best choice: however, the tools for PSL could be improved.

2.5 Multi-agent systems

Section 2.2 presented several conceptual views of the extended enterprise. All these views modelled the enterprise as a set of individual entities that interacted in some manner. The major difference between these proposals was the actual way in which they interacted.

From the point of view of the distributed artificial intelligence research community, these individual entities would be usually represented as “agents”. According to the Foundation for Intelligent Physical Agents (FIPA) reference architecture [26], an agent is “a computational process that implements the autonomous, communicating functionality of an application”. Leitão [52] mentions several other definitions and provides a more exhaustive definition of an agent as “an autonomous component that represents physical or logical objects in the system, capable to act in order to achieve its goals, and being able to interact with other agents, when it does not possess knowledge and skills to reach alone its objectives”. While the FIPA definition only considers agents as separate entities that can talk to each other, the definition provided by Leitão explicitly considers the ability to cooperate with other agents to meet its own goals and the fact that an agent may operate physical objects in the real world.

2.5.1 Applications

Agent-based systems have many applications in intelligent manufacturing. Shen et al. [74] classified the existing approaches into five categories. Each of these categories below will be examined below.

2.5.1.1 Enterprise integration

Enterprise integration attempts to join all the separate information systems normally used at most enterprises into an uniform whole, in order to optimise its performance. In this situation, agents would normally wrap the existing legacy systems and provide additional

intelligence. Proposals in this area largely vary on their scope and abstraction level, from enterprise-wide reference architectures to dynamic workflow composition engines.

Nahm and Ishikawa [61] proposed a hybrid multi-agent system architecture with two basic types of agents: hybrid behaviour agents (HBAs) that respond to events and monitor their environment, and hybrid interaction agents (HIAs) that act as mediators between HBAs. The architecture was tested by modelling a collaborative product development environment, with several subtypes of HBAs and HIAs that were specific to the problem domain.

Lea et al. [50] presented a prototype of a multi-agent ERP system combining four types of agents: task agents that performed the actual work, data collection agents that accessed existing data stores, coordination agents that joined the agents in each department together, and interface agents that interacted with the human users of the system. Using these agents, Lea et al. showed how an order could be evaluated by a collaboration between the agents of the marketing, accounting, inventory and logistics departments.

As an alternative to explicit workflows such as those written in a complete WS-BPEL or BPMN process description (see Section 2.4.4), Poggi et al. [67] used agents to dynamically compose web services based on the current environment. An existing agent platform was integrated with a business rule engine, an ontology engine and distributed trust handling. A custom engine implemented a subset of the BPEL [63] specification, completing an abstract model by using rule inference and collaborations between agents.

2.5.1.2 Enterprise collaboration

Collaborations between enterprises can take many forms, such as supply chains, virtual enterprises or extended enterprises. Integrating extended enterprises would be handled basically in the same way as a regular one: it would be a matter of increasing the scale and the complexity of the system, as the boundaries between the collaborating organisations would need to be explicitly addressed.

Fox et al. [28] defined a supply chain as “a set of activities which span enterprise functions from the ordering and receipt of raw materials from the manufacturing of products through the distribution and delivery to the customer” and proposed using agents to handle the complex dynamic problems that they presented. Several kinds of functional agents would be required: 1. order acquisition agents that received orders from clients and negotiated terms and changes, 2. logistics agents that coordinated the plants, suppliers and distribution centres, 3. transportation agents that assigned and scheduled inter-plant movement requests, 4. scheduling agents that coordinated requests from the resource agents with the production schedules of the logistics agents and considered trade-offs when solving unfeasible situations, 5. resource agents monitor inventory levels and send purchase requests in order to meet the schedule and 6. dispatching agents perform the order release and real-time floor control functions, delegating on the scheduling agent when deviations from the schedule happen.

More recently, Sadeh et al. [73] proposed the Multi-Agent Supply Chain cOordination Tool (MASCOT). Instead of multiple kinds of agents, Sadeh et al. suggests using a single agent per enterprise and per site. The internal architecture of these agents is considerably more complex, as shown in Figure 2.26. A MASCOT agent contains a set of Knowledge Sources (KS) sharing a common data store (a “blackboard”) and is operated through a graphical user interface. KS range from communication systems with the existing systems

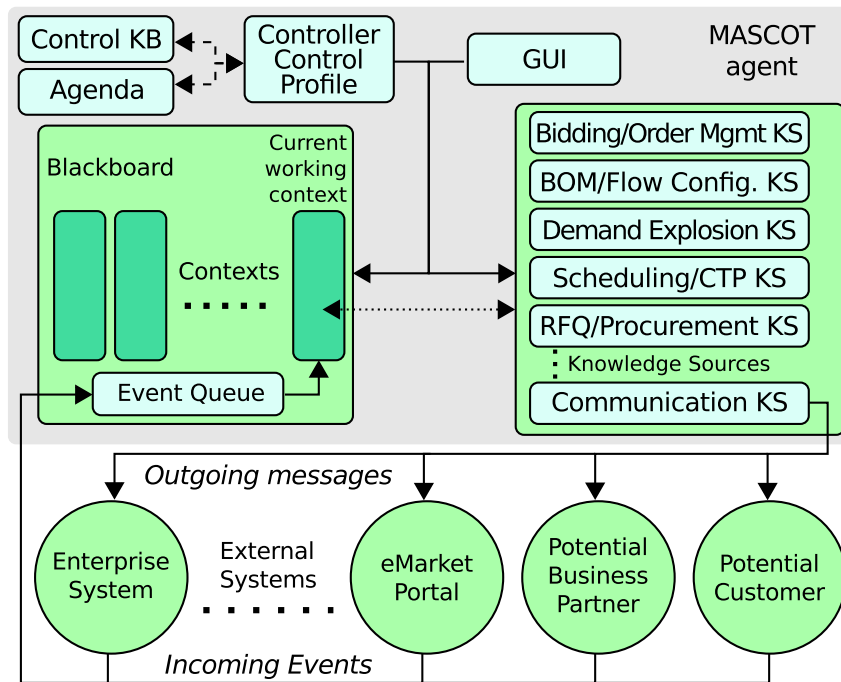


Figure 2.26. Architecture of a MASCOT agent [73]

in the organisation, to BOM databases, order management systems and scheduling systems, among others. Agents can work on multiple problems at the same time, using a different “context” for each of them.

2.5.1.3 Planning and scheduling

Manufacturing process planning and scheduling requires the selection and sequencing of manufacturing processes in order to achieve certain goals while meeting specific constraints. Most real-world scheduling problems are considered to be hard enough that an optimal solution is too expensive to compute, so practitioners must settle for “good enough” solutions using heuristic algorithms. Using these algorithms in a centralised approach may produce better results than using a distributed approach, but only if no disturbances appear, and with the added complexity and risk of having a single point of failure and a bottleneck in the central scheduler. On the other hand, a distributed system may be able to react more quickly to problems and may keep complexity down by relying on the emergent behaviour of all the separate agents [65].

Yet Another Manufacturing System (Yams) is one of the first multi-agent systems for process planning and scheduling [66]. It was based on the contract net model. After posting a job offer, “Manager” agents would collect “Bid” messages from the “Bidder” agents and send an “Award” message to the most suitable one. The “Bidder” would then periodically send status or termination report, and the “Manager” could terminate the contract at any time. In order to reduce communication overhead, agents were organised as a hierarchy mimicking the organisation of the manufacturing company, its plants and its equipment. Agents could only communicate with their children, their parents and their siblings.

Since then, a considerable number of proposals have been published. Most of them are

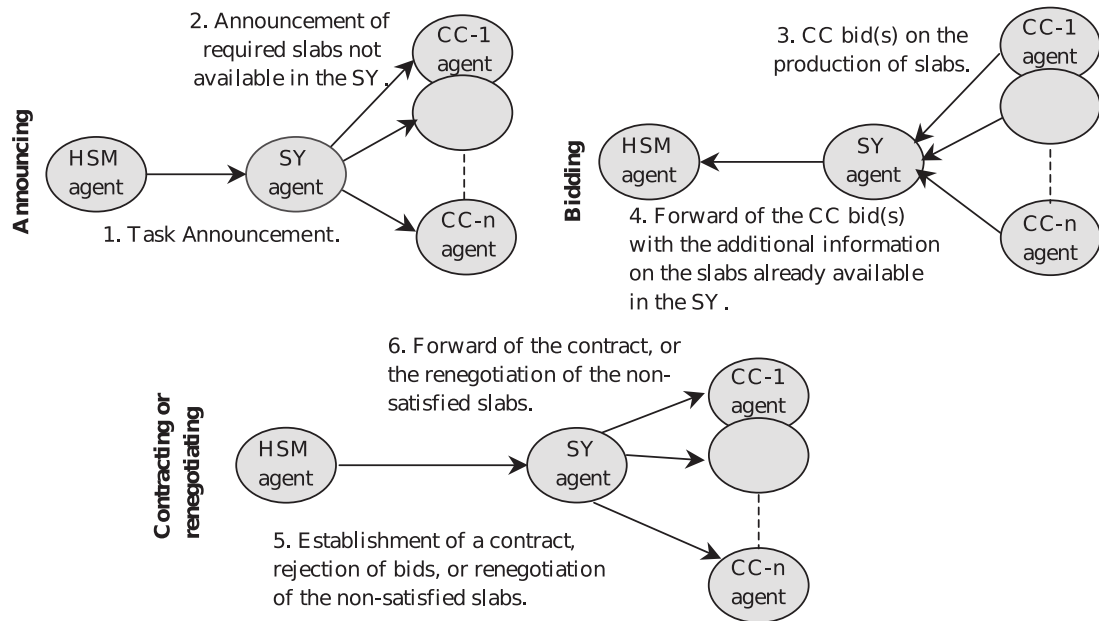


Figure 2.27. Contract net for scheduling steel milling and casting [18]

also based on market-oriented approaches, such as auctions based on delivery times or virtual currencies which combine manufacturing cost with delays [1, 17, 18, 32, 51].

Cowling et al. [18] is particularly interesting, as it combines contract nets with traditional scheduling algorithms and rescheduling strategies for scheduling steel casting and milling. There are four kinds of agents: user agents, slabyard (SY) agents, hot strip milling (HSM) agents and continuous caster (CC) agents. HSM and CC agents have their own internal scheduling algorithms and rescheduling strategies. SY agents act as mediators between the HSM and CC agents. Inter-agent cooperation is performed through a bid mechanism (shown in Figure), in which HSM requests slabs from the slabyard, which may request new slabs if necessary. Schedules are repaired by selecting a strategy that maximises “robustness”, a metric that combines the efficiency of a schedule with the stability of the completion times.

2.5.1.4 Shop floor control

Manufacturing shop floor control relates to strategies and algorithms for operating a manufacturing plant, taking into account both the present and past observed states and the demand from the market. Shop floor control can work at several levels of abstraction.

At a high level, it is mostly concerned with distributing work efficiently through the manufacturing resources in order to meet the schedules and maximise production levels. As it can be considered as a particular case of agent-based scheduling, it will not be discussed further.

On the other hand, at a low level, agents may need to control individual resources in order to implement a specific production process. For instance, Brückner et al. [8] proposed using agents to control some of the painting steps in the Daimler-Benz AG car plant, as the seemingly simple process had problems with feedback loops and buffering due to failed quality checks.

Tseng et al. [81] use agents to wrap incoming jobs and physical resources for a flexible

manufacturing system, which needs to implement mass customisation capabilities. Agents coordinate through an auction mechanism, in which the price charged by the manufacturing agents to the job agents depends on their capability, utilisation and delivery times.

Bussmann and Schild [10] use agents to control another flexible manufacturing system that distributes workpieces over several machines connected by forwards, backwards and supply conveyors. Workpiece agents bid for jobs and machines post offers for them as long as workpieces are properly leaving their storage areas. This prevents bottlenecks in the system. The workpiece and switching table agents collaborate in order to implement dynamic routing.

Agents are also useful for process control beyond manufacturing. Tatara et al. [78] used an adaptive multi-agent system to control a network of geographically distributed chemical reactors, due to the non-linearity of the chemical process involved. Agents can change their local objectives in order to meet global objectives, and communicate through a shared knowledge environment that promotes ideas through reinforcement.

2.5.1.5 Holonic Manufacturing Systems

Holons were previously described in Section 2.2 as a way to conceptualise the enterprise. A holon is an autonomous entity that is formed by several lower-level entities or wraps a physical object, and that in turn is part of several higher-level holons in the holarchy.

The concept of a holon is quite similar to that of an agent, but there are some slight differences. Leitão [52] points out that agents were conceived in the distributed artificial intelligence area (by computer scientists) and that holons were conceived in the computer-integrated manufacturing area. While agents are both a concept and a set of technologies, holons are only conceptual. Holons are built through lower level holons, while agents are normally standalone entities that collaborate and communicate. Holons also tend to accommodate better wrapping physical entities and their real-time constraints.

A large number of proposals in the HMS literature are based on the PROSA architecture (see Section 2.2). For instance, FABMAS is a PROSA-based system that implements holons through a multi-agent system for controlling a semiconductor wafer fabrication facility [59]. Giret Boggino [31] is a PROSA-based methodology using FIPA standards for the system architecture. However, not all of them are, such as the holonic supply chain management system by Ulieru and Cobzaru [82]. This system uses a two-level holarchy, with a multi-enterprise level that integrates customers, order managers, logistics, transport and banking, and an enterprise level that covers the suppliers and the plants. Agents are equipped with rule-based inference engines in a custom language, and use contract nets for negotiation.

Most approaches also rely on the FIPA specifications [27] or the IEC 61499 function block abstraction [37]. Deen [20] presented a holonic architecture that uses IEC 61499 for both low and high level control and FIPA for high level control.

2.5.2 Agent platforms

At the date of this writing (24th November 2013), FIPA lists 10 different agent platforms in their official website². Among these, the Java Agent DEvelopment (JADE) framework

²<http://fipa.org/resources/livesystems.html>

Platform	License	FIPA-based	Website	Features
AgentFactory	LGPL	Yes	www.agentfactory.com	Divided into a runtime environment and a common agent language framework, with support for multiple languages.
Cougaar	BSD-based	No	cougaar.org	Agent behaviour is plugged into the framework's agents. Provides messaging, naming, mobility, blackboards and external UIs.
FIPA-OS	Custom	Yes	sourceforge.net/projects/fipa-os	Java-based. Implements most FIPA experimental specifications. Not updated since 2001.
GORITE	LGPL	No	www.intendico.com/gorite	Java-based. Agents are organised in goal-oriented teams using a BDI framework.
JACK®	Proprietary	No	www.agent-software.com	Java-based. Agents are defined using BDI. Supports mobile devices, teams and simulation.
JADE	LGPL	Yes	jade.tilab.com	Java-based. Agents can be moved between computers. Optional support for semantics, mobile agents, workflows and web services.
Janus	GPLv3	Partially	janus-project.org	Java-based. Uses the Capacity-Role-Interaction-Organisation metamodel. Supports recursive/mobile agents and holons.
MaDKit	GPL	Partially	www.madkit.org	Java-based. Uses the Agent-Group-Role organisational model and implements artificial societies. Supports simulation and provides distributed application authoring tools.
MobileC	Proprietary	Yes	www.mobilec.org	C++-based. Agents are implemented in a portable superset of C90 and run in an interpreter.
Zeus	Custom	Yes	sourceforge.net/projects/zeusagent	Java-based. Includes a rule engine, a planner and monitoring tools. Not updated since 2001.

Table 2.3. A comparison of the currently available agent platforms

and JACK[®] seem to be the most actively developed. FIPA-OS is still available, but it has not been updated since 2009. FIPA-OS and JADE are open source platforms, while JACK[®] appears to be a closed source commercial product. The official website for Zeus is no longer available, but the code can be still downloaded.

Searching “agent software” through Google³ and Wikipedia^{4,5} and the survey by Nikolai and Madey [62] yielded several other results. A large number of these systems were dedicated to agent-based simulation rather than implementing real systems and were discarded. Others had not received any updates in over 3 years, or were no longer available.

Table 2.3 lists the main features of each of the agent platforms. In general, most of these platforms are largely based on Java and are for academic purposes, except for JACK[®] and MobileC, which are commercial products. Judging from the activity in mailing lists and forums, JADE seems to have the most active user community by far. JADE has been extended to handle mobile agents (JADE-LEAP), integrate with Eclipse (eJade⁶) and handle semantic technologies (SemanticAgent⁷), among others.

However, one of the most interesting recent contribution to JADE for the present work is the Workflows and Agents Development Environment (WADE) [11]. WADE provides a set of Java libraries that allow users to implement agents using a workflow paradigm, much like that of the process notations listed in Section 2.4.1 or Section 2.4.4. Instead of defining its own abstract notation, workflows are implemented through regular Java code.

References

- [1] S. Adhau, M. Mittal, and A. Mittal. A multi-agent system for distributed multi-project scheduling: An auction-based negotiation approach. *Engineering Applications of Artificial Intelligence*, December 2011. ISSN 0952-1976. doi: 10.1016/j.engappai.2011.12.003. 2.42
- [2] R. S. Aguilar-Savén. Business process modelling: Review and framework. *International Journal of Production Economics*, 90(2):129–149, July 2004. ISSN 0925-5273. doi: 10.1016/S0925-5273(03)00102-6. 2.28
- [3] R. Babiceanu and F. Chen. Development and applications of holonic manufacturing systems: A survey. *Journal of Intelligent Manufacturing*, 17(1):111–131, 2006. ISSN 0956-5515. doi: 10.1007/s10845-005-5516-y. 2.11
- [4] C. Bock. Interprocess communication in the process specification language. Technical Report NISTIR 7348, National Institute of Standards and Technology, Gaithersburg, MD, USA, October 2006. 2.38
- [5] C. Bock and M. Gruninger. PSL: a semantic domain for flow models. *Software & Systems Modeling*, 4(2):209–231, 2005. ISSN 1619-1366. doi: 10.1007/s10270-004-0066-x. 2.29

³<http://www.google.com>

⁴https://en.wikipedia.org/wiki/Comparison_of_agent-based_modeling_software

⁵https://en.wikipedia.org/wiki/Category:Agent-based_software

⁶<http://selab.fbk.eu/dnguyen/ejade/index.html>

⁷<https://code.google.com/p/semanticagent/>

- [6] J. Browne, I. Hunt, and J. Zhang. The Extended Enterprise (EE). In L. M. Camarinha-Matos, H. Afsarmanes, and V. Merik, editors, *Intelligent Systems for Manufacturing: Multi-Agent Systems and Virtual Organizations*, pages 3–30. Kluwer Academic Publishers, Londres, 1998. 2.10
- [7] H. V. Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters. Reference architecture for holonic manufacturing systems: PROSA. *Computers in Industry*, 37(3):255–274, November 1998. ISSN 0166-3615. 2.5, 2.11
- [8] S. Brückner, J. Wyns, P. Peeters, and M. Kollingbaum. Designing agents for manufacturing control. In *Proceedings of the 2nd AI & Manufacturing Research Planning Workshop*, pages 40–46, 1998. 2.42
- [9] S. Bussmann. An agent-oriented architecture for holonic manufacturing control. In *Proceedings of the First Open Workshop IMS Europe*, Lausanne, Switzerland, 1998. URL <http://stefan-bussmann.de/downloads/ims98.pdf>. 2.6
- [10] S. Bussmann and K. Schild. An agent-based approach to the control of flexible production systems. In *2001 8th IEEE International Conference on Emerging Technologies and Factory Automation, 2001. Proceedings*, volume 2, pages 481–488 vol.2, October 2001. doi: 10.1109/ETFA.2001.997722. 2.43
- [11] G. Caire, D. Gotta, and M. Banzi. WADE: a software platform to develop mission critical applications exploiting agents and workflows. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, pages 29–36, 2008. 2.45
- [12] L. M. Camarinha-Matos and H. Afsarmanesh. Elements of a base VE infrastructure. *Computers in Industry*, 51(2):139–163, June 2003. ISSN 01663615. doi: 10.1016/S0166-3615(03)00033-2. 2.1
- [13] D. Chen, B. Vallespir, and G. Doumeingts. GRAI integrated methodology and its mapping onto generic enterprise reference architecture and methodology. *Computers in Industry*, 33(2–3):387–394, September 1997. ISSN 0166-3615. doi: 10.1016/S0166-3615(97)00043-2. 2.13, 2.18
- [14] P. P.-s. Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1:9–36, 1976. 2.13
- [15] C.-M. Chituc, C. Toscano, and A. Azevedo. Interoperability in collaborative networks: Independent and industry-specific initiatives – the case of the footwear industry. *Computers in Industry*, 59(7):741–757, September 2008. ISSN 0166-3615. 2.2
- [16] J. H. Christensen. Holonic manufacturing systems: Initial architecture and standards directions. In *Proceedings of the First European Conference on Holonic Manufacturing Systems*, Hannover, Germany, December 1994. 2.5, 2.6
- [17] G. Confessore, S. Giordani, and S. Rismondo. A market-based multi-agent system model for decentralized multi-project scheduling. *Annals of Operations Research*, 150(1):115–135, 2007. ISSN 0254-5330. doi: 10.1007/s10479-006-0158-9. 2.42

-
- [18] P. I. Cowling, D. Ouelhadj, and S. Petrovic. Dynamic scheduling of steel casting and milling using multi-agents. *Production Planning & Control*, 15(2):178–188, 2004. 2.42
 - [19] M. V. de Castro. *Aproximación MDA para el desarrollo orientado a servicios de sistemas de información web: del modelo de negocio al modelo de composición de servicios web*. PhD thesis, Universidad Rey Juan Carlos, March 2007. 2.9
 - [20] S. M. Deen. HMS/FB architecture and its implementation. In *Agent Based Manufacturing: Advances in the Holonic Approach*. Springer, July 2003. ISBN 9783540440697. 2.43
 - [21] D. M. Dilts, N. P. Boyd, and H. H. Whorms. The evolution of control architectures for automated manufacturing systems. *Journal of Manufacturing Systems*, 10(1): 79–93, 1991. ISSN 0278-6125. 2.4, 2.5, 2.10
 - [22] G. Doumeingts, Y. Ducq, B. Vallespir, and S. Kleinhans. Production management and enterprise modelling. *Computers in Industry*, 42(2–3):245–263, June 2000. ISSN 0166-3615. doi: 10.1016/S0166-3615(99)00074-3. 2.13
 - [23] G. Engels, A. Hess, B. Humm, O. Juwig, M. Lohmann, J. Richter, M. Voß, and J. Willkomm. A method for engineering a true Service-Oriented Architecture. In J. Cordeiro and J. Filipe, editors, *Proceedings of the 10th International Conference on Enterprise Information Systems*, pages 272–281, Barcelona, España, 2008. ISBN 978-989-8111-38-8. 2.9
 - [24] T. Erl. *SOA: Principles of Service Design*. Prentice Hall, Indiana, EEUU, 2008. ISBN 0132344823. 2.9
 - [25] M. Fletcher, E. Garcia-Herreros, J. Christensen, S. Deen, and R. Mittmann. An open architecture for holonic cooperation and autonomy. In *11th International Workshop on Database and Expert Systems Applications, 2000. Proceedings*, pages 224–230, 2000. doi: 10.1109/DEXA.2000.875031. 2.6, 2.7
 - [26] Foundation for Intelligent Physical Agents. FIPA abstract architecture specification SC00001L, December 2002. URL <http://www.fipa.org/specs/fipa00001/SC00001L.pdf>. Last checked: November 6th, 2013. 2.7, 2.39
 - [27] Foundation for Intelligent Physical Agents. FIPA standard status specifications, 2002. URL <http://www.fipa.org/repository/standardspecs.html>. Last checked: November 6th, 2013. 2.43
 - [28] M. S. Fox, J. F. Chionglo, and M. Barbuceanu. The integrated supply chain management system. Technical report, University of Toronto, Department of Industrial Engineering, 1993. 2.40
 - [29] G. Doumeingts, B. Vallespir, M. Zannittin, and D. Chen. GIM-GRAI integrated methodology, a methodology for designing CIM systems, version 1.0. Technical report, University Bordeaux, Bordeaux, France, May 1992. 2.13
 - [30] S. Ghosh, A. Arsanjani, and A. Allam. SOMA: a method for developing service-oriented solutions. *IBM Systems Journal*, 47(3):377–396, 2008. 2.9
-

- [31] A. Giret Boggino. *ANEMONA: una metodología multiagente para sistemas holónicos de fabricación*. PhD thesis, Universidad Politécnica de Valencia, July 2005. 2.9, 2.43
- [32] D. Goldberg, V. Ciciello, M. B. Dias, R. Simmons, S. Smith, and A. Stentz. Task allocation using a distributed market-based planning mechanism. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 996–997, 2003. 2.42
- [33] IBM Corporation, MESA International, and Capgemini. SOA in Manufacturing Guidebook, May 2008. 2.8
- [34] IFAC/IFIP Task Force. GERAM: generalised enterprise reference architecture and methodology, March 1999. URL <http://www.ict.griffith.edu.au/~bernus/taskforce/geram/versions/geram1-6-3/v1.6.3.html>. 2.18, 2.19, 2.20
- [35] International Electrotechnical Commission. IEC/FDIS 62264-1:2003 – enterprise-control system integration – part 1: Models and terminology, 2003. 2.23, 2.25, 2.26, 2.27
- [36] International Electrotechnical Commission. IEC/FDIS 62264-2:2004 – enterprise-control system integration – part 2: Model object attributes, 2004. 2.25
- [37] International Electrotechnical Commission. Function blocks - part 1: Architecture. Technical Report IEC 61499-1, IEC, 2005. 2.43
- [38] International Electrotechnical Commission. IEC/DIS 62264-3:2007 – enterprise-control system integration – part 3: Activity models of manufacturing operations management, 2007. 2.25, 2.26, 2.28
- [39] International Standards Organization. ISO 15704 – industrial automation systems – requirements for enterprise-reference architectures and methodologies, August 1999. 2.18
- [40] International Standards Organization. ISO 18629-1 – process specification language – part 1: Overview and basic principles, 2004. 2.29
- [41] International Standards Organization. ISO 19439 – enterprise integration – framework for enterprise modelling, 2006. 2.21
- [42] International Standards Organization. ISO 19440 – enterprise integration – constructs for enterprise modelling, 2007. 2.21
- [43] K. Johansen, M. Comstock, and M. Winroth. Coordination in collaborative manufacturing mega-networks: a case study. *Journal of Engineering and Technology Management*, 22(3):226–244, September 2005. ISSN 0923-4748. 2.1, 2.10
- [44] A. Knutilla, C. Schlenoff, S. Ray, S. T. Polyak, A. Tate, S. C. Cheah, and R. C. Anderson. Process specification language: An analysis of existing representations. Technical Report NISTIR 6133, National Institute of Standards and Technology, Gaithersburg, MD, USA, 1998. 2.28

-
- [45] A. Koestler. Some general properties of self-regulating open hierarchic order (SOHO). In A. Koestler and J. R. Smythies, editors, *Beyond Reductionism: New Perspectives In The Life Sciences*. Houghton Mifflin Co, 1971. ISBN 0807015350. 2.11
 - [46] A. Koestler. *The Ghost in the Machine*. Penguin Books, June 1990. ISBN 978-0140191929. 2.5, 2.11
 - [47] K. Kosanke. CIMOSA – overview and status. *Computers in Industry*, 27(2):101–109, October 1995. ISSN 0166-3615. doi: 10.1016/0166-3615(95)00016-9. 2.15, 2.17
 - [48] K. Kosanke and M. Zelm. CIMOSA modelling processes. *Computers in Industry*, 40(2–3):141–153, November 1999. ISSN 0166-3615. doi: 10.1016/S0166-3615(99)00020-2. 2.15, 2.18
 - [49] K. Kosanke, F. Vernadat, and M. Zelm. CIMOSA: enterprise engineering and integration. *Computers in Industry*, 40(2–3):83–97, November 1999. ISSN 0166-3615. doi: 10.1016/S0166-3615(99)00016-0. 2.15
 - [50] B.-R. Lea, M. C. Gupta, and W.-B. Yu. A prototype multi-agent ERP system: an integrated architecture and a conceptual framework. *Technovation*, 25(4):433–441, 2005. ISSN 0166-4972. doi: 10.1016/S0166-4972(03)00153-6. 2.40
 - [51] Y.-H. Lee, S. R. T. Kumara, and K. Chatterjee. Multiagent based dynamic resource scheduling for distributed multiple projects using a market mechanism. *Journal of Intelligent Manufacturing*, 14(5):471–484, 2003. ISSN 0956-5515. doi: 10.1023/A:1025753309346. 2.42
 - [52] P. Leitão. Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence*, 22(7):979–991, October 2009. ISSN 0952-1976. 2.6, 2.7, 2.11, 2.12, 2.39, 2.43
 - [53] F. Macia-Perez, J. V. Berna-Martinez, D. Marcos-Jonquera, I. Lorenzo-Fonseca, and A. Ferrandiz-Colmeiro. A new paradigm: cloud agile manufacturing. *International Journal of Advanced Science and Technology*, 45:47–54, August 2012. ISSN 2005-4238. 2.8
 - [54] Manufuture High Level Group. Manufuture: a vision for 2020. Technical report, European Commission, Brussels, Belgium, November 2004. ISBN 92-894-8322-9. 2.1
 - [55] M. Marcos, F. Aguayo, M. Sánchez Carrilero, L. Sevilla, and J. R. Lama. Toward the next generation of manufacturing systems. Frabiho: a synthesis model for distributed manufacturing. In *Proceedings of the First I*proms Virtual Conference*, pages 35–40. Elsevier, 2005. 2.10, 2.11
 - [56] V. Marik, M. Fletcher, and M. Pechoucek. Holons & agents: Recent developments and mutual impacts. In V. Marik, O. Stepankova, H. Krautwurmova, and M. Luck, editors, *Multi-Agent Systems and Applications II*, volume 2322 of *Lecture Notes in Computer Science*, pages 89–106. Springer Berlin / Heidelberg, 2002. ISBN 978-3-540-43377-4. 2.6
-

- [57] R. J. Mayer, C. P. Menzel, M. K. Painter, P. S. de Witte, T. Blinn, and B. Perakath. IDEF3 process description capture method report. Interim Technical Report AL-TR-1995-XXXX, Knowledge Based Systems Inc., Texas, USA, September 1995. 2.28
- [58] L. Monostori, J. Váncza, and S. Kumara. Agent-based systems for manufacturing. *CIRP Annals - Manufacturing Technology*, 55(2):697–720, 2006. ISSN 0007-8506. doi: 10.1016/j.cirp.2006.10.004. 2.7
- [59] L. Mönch, M. Stehli, and J. Zimmermann. FABMAS: an agent-based system for production control of semiconductor manufacturing processes. In V. Marík, D. McFarlane, and P. Valckenaers, editors, *Holonic and Multi-Agent Systems for Manufacturing*, volume 2744 of *Lecture Notes in Computer Science*, pages 258–267. Springer Berlin / Heidelberg, 2003. ISBN 978-3-540-40751-5. 2.43
- [60] F. Nachira. Towards a network of digital business ecosystems fostering the local development. Discussion paper, European Commission, Brussels, Belgium, September 2002. URL <http://www.digital-ecosystems.org/doc/discussionpaper.pdf>. Last checked: November 6th, 2013. 2.1, 2.2
- [61] Y.-E. Nahm and H. Ishikawa. A hybrid multi-agent system architecture for enterprise integration using computer networks. *Robotics and Computer-Integrated Manufacturing*, 21(3):217–234, June 2005. ISSN 0736-5845. doi: 10.1016/j.rcim.2004.07.016. 2.40
- [62] C. Nikolai and G. Madey. Tools of the trade: A survey of various agent based modeling platforms. *Journal of Artificial Societies and Social Simulation*, 12(2):2, 2009. 2.7, 2.45
- [63] OASIS. Web Service Business Process Execution Language (WS-BPEL) 2.0, April 2007. URL <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>. Last checked: November 6th, 2013. 2.30, 2.40
- [64] Object Management Group. Business process model and notation 2.0, January 2011. URL <http://www.omg.org/spec/BPMN/2.0/>. Last checked: November 6th, 2013. 2.28, 2.31
- [65] D. Ouelhadj and S. Petrovic. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, 12(4):417–431, 2009. ISSN 1094-6136. doi: 10.1007/s10951-008-0090-8. 2.41
- [66] H. V. D. Parunak. Manufacturing experience with the contract net. *Distributed Artificial Intelligence*, 1:285–310, 1987. 2.4, 2.41
- [67] A. Poggi, M. Tomaiuolo, and P. Turci. An agent-based service oriented architecture. In *Proc. 8th AI* IA/TABOO Joint Workshop From Objects to Agents: Agents and Industry: Technological Applications of Software Agents, Genova*, pages 157–165, 2007. 2.40
- [68] R. Poler, F. Lario, and G. Doumeingts. Dynamic modelling of decision systems (DMDS). *Computers in Industry*, 49(2):175–193, October 2002. ISSN 0166-3615. doi:

- 10.1016/S0166-3615(02)00083-0. URL <http://www.sciencedirect.com/science/article/pii/S0166361502000830>. 2.13, 2.14
- [69] J. T. Pollock. The big issue: Interoperability vs integration. *eAI Journal*, October 2001. URL http://me.jtpollock.us/pubs/2001.08-BigIssue_eAIJournal.pdf. 2.2
- [70] L. Ribeiro, J. Barata, and P. Mendes. MAS and SOA: complementary automation paradigms. In A. Azevedo, editor, *Innovation in Manufacturing Networks*, volume 266 of *IFIP International Federation for Information Processing*, pages 259–268. Springer Boston, 2008. ISBN 978-0-387-09491-5. 2.8
- [71] M. Rother and J. Shook. *Learning to See: Value Stream Mapping to Add Value and Eliminate MUDA*. Lean Enterprise Institute, June 1999. ISBN 978-0966784305. 2.30
- [72] K. Ryu and M. Jung. Agent-based fractal architecture and modelling for developing distributed manufacturing systems. *International Journal of Production Research*, 41(17):4233–4255, 2003. ISSN 0020-7543. doi: 10.1080/0020754031000149275. 2.4, 2.5
- [73] N. M. Sadeh, D. W. Hildum, and D. Kjenstad. Agent-based e-supply chain decision support. *Journal of Organizational Computing and Electronic Commerce*, 13(3-4): 225–241, 2003. 2.40, 2.41
- [74] W. Shen, Q. Hao, H. J. Yoon, and D. H. Norrie. Applications of agent-based systems in intelligent manufacturing: An updated review. *Advanced Engineering Informatics*, 20(4):415–431, October 2006. ISSN 1474-0346. 2.3, 2.4, 2.10, 2.39
- [75] N. Spanoudakis and P. Moraitis. Using ASEME methodology for model-driven agent systems development. In D. Weyns and M.-P. Gleizes, editors, *Agent Oriented Software Engineering XI*, volume 6788 of *Lecture Notes in Computer Science (LNCS)*, pages 106–127. Springer-Verlag Berlin Heidelberg, 2011. 2.9
- [76] Z. Stojanović. *A Method for Component-Based and Service-Oriented Software Systems Engineering*. PhD thesis, Delft University of Technology, 2005. 2.9
- [77] D. Tapia, S. Rodríguez, J. Bajo, and J. Corchado. FUSION@: a SOA-based multi-agent architecture. In J. Corchado, S. Rodríguez, J. Llinas, and J. Molina, editors, *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*, volume 50, pages 99–107. Springer Berlin/Heidelberg, 2009. 2.8
- [78] E. Tatara, M. North, C. Hood, F. Teymour, and A. Cinar. Agent-based control of spatially distributed chemical reactor networks. In S. Brueckner, G. Di Marzo Seruendo, D. Hales, and F. Zambonelli, editors, *Engineering Self-Organising Systems*, volume 3910 of *Lecture Notes in Computer Science*, pages 222–231. Springer Berlin / Heidelberg, 2006. ISBN 978-3-540-33342-5. 2.43
- [79] A. Tharumarajah, A. Wells, and L. Nemes. Comparison of emerging manufacturing concepts. In *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics*, pages 325–331, California, EEUU, 1998. 2.10

- [80] The Open Group. *The Open Group Architecture Framework (TOGAF) Version 9.1*. The Open Group, 2011. ISBN 978-90-8753-679-4. 2.22
- [81] M. M. Tseng, M. Lei, C. Su, and M. E. Merchant. A collaborative control system for mass customization manufacturing. *CIRP Annals - Manufacturing Technology*, 46(1): 373–376, 1997. ISSN 0007-8506. doi: 10.1016/S0007-8506(07)60846-4. 2.42
- [82] M. Ulieru and M. Cobzaru. Building holonic supply chain management systems: an e-logistics application for the telephone manufacturing industry. *IEEE Transactions on Industrial Informatics*, 1(1):18–30, 2005. ISSN 1551-3203. doi: 10.1109/TII.2005.843827. 2.43
- [83] J. Vaario and K. Ueda. Biological concept of self-organization for dynamic shop-floor configuration. In N. Okino, T. Hiroyuki, and F. Susumu, editors, *Selected, revised proceedings of the IFIP TC5/WG5.7 International Conference on Advances in Production Management Systems*, volume 114 of *IFIP Conference Proceedings*, pages 55–66, Kyoto, Japan, November 1996. Chapman & Hall. ISBN 0-412-82350-0. 2.4, 2.10
- [84] H. Warnecke. *The Fractal Company: A Revolution in Corporate Culture*. Springer-Verlag, August 1997. ISBN 038756537X. 2.4, 2.11
- [85] T. J. Williams, editor. *A Reference Model for Computer Integrated Manufacturing (CIM)*. Instrument Society of America, North Carolina, USA, second edition, 1989. ISBN 1-55617-225-7. 2.15
- [86] T. J. Williams. The Purdue enterprise reference architecture. *Computers in Industry*, 24(2–3):141–158, September 1994. ISSN 0166-3615. doi: 10.1016/0166-3615(94)90017-5. 2.15, 2.16
- [87] Workflow Management Coalition. WPMC-TC-1011: terminology and glossary 3.0, February 1999. URL http://www.workflowpatterns.com/documentation/documents/TC-1011_term_glossary_v3.pdf. Last checked: November 6th, 2013. 2.30
- [88] World Batch Forum. Business to manufacturing markup language (B2MML), 2008. URL http://www.isa.org/Content/NavigationMenu/General_Information/Partners_and_Affiliates/WBF/Working_Groups2/XML_Working_Group/B2MML/B2MML.htm. Last checked: November 6th, 2013. 2.25
- [89] J. A. Zachmann. The Zachman Framework™: the Official Concise Definition, 2008. URL <http://www.zachmaninternational.com/index.php/the-zachman-framework>. Last checked: November 6th, 2013. 2.23, 2.24

3

Concepts of software and service engineering for distributed manufacturing

This chapter presents the software engineering concepts behind the present Thesis and how they can help solve the challenges identified in the previous chapter.

SOAs are introduced in Section 3.1 as a scalable approach to organize and integrate the information systems of a manufacturing enterprise using standards-based technologies with greater industrial support. In particular, from Section 3.1.2 onwards some of the most commonly used technologies for developing SOAs are presented.

Since using SOAs raises several issues on how to meet performance requirements, Section 3.2 introduces the overall field of performance engineering.

The high complexity of developing a SOA has been tackled in the literature by developing model-driven methodologies that help structure and partially automate the process involved. The methodologies themselves are later discussed in Chapter 4: readers are suggested to read Section 3.3 first, as it introduces the concepts behind model-driven software engineering in general.

3.1 Service-oriented architectures

Section 2.4.4 described how workflows could help an organisation formalise their practices, assisting the improvement and control of their business processes. However, the information systems need to provide the basic building blocks to make them work.

This requires a change in the way the systems are conceived: instead of closed silos of information which are solely manipulated through a user interface, they must now provide a catalogue of services which can be flexibly reused over all workflows in the organisation. These new systems are known as Service-Oriented Architectures (SOAs).

3.1.1 Definitions and goals

There is not a clear consensus on what a SOA is. The Organisation for the Advancement of Structured Information Systems (OASIS) defines SOA as “a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains” [30]. Essentially, SOA is a different way of organising separate software systems from different enterprises into a single whole and taking advantage of that integration.

The World Wide Web Consortium (W3C) has a much more detailed definition: “a SOA is a form of distributed systems architecture that is typically characterized by the following properties: logical view [...], message orientation [...], description orientation [...], granularity [...], network orientation [...], platform neutral [...]” [55]. This is a rather more technical definition, which focuses on the distributed nature of services and the clear separation between what the services does and how it is implemented. It also requires that services have formal descriptions that can be processed automatically, which is not always the case in practice.

Erl [17] notes that “SOA establishes an architectural model that aims to enhance the efficiency, agility and productivity by positioning services as the primary means through which solution logic is represented [...]”. In turn, services “exist as physically independent software programs with distinct design characteristics that support the attainment of the strategic goals associated with service-oriented computing”. These goals are highly focused on increased reuse and standardisation, in order to save costs:

- Increased Intrinsic Interoperability: through common design principles and the usage of open standards, software programs can directly work with each other without requiring expensive custom integration logic.
- Increased Federation: a collection of standardised services that are easy to compose creates an information environment in which applications and resources are united, and yet each of these maintains their autonomy and self-governance.
- Increased Vendor Diversification Options: using vendor-neutral technologies throughout the enterprise prevents organisations to be locked into specific solutions. Organisations are free to adopt new solutions and resources, or to replace previously existing ones.
- Increased Business and Technology Domain Alignment: services can be derived from descriptions of the business itself, increasing the abstraction level of the system and ensuring that the system represents the actual business requirements of the enterprise. In addition, having business functions as explicit software programs allow to recombine these into new processes when necessary.
- Increased Return on Investment (ROI): while the first version of a proper service can cost more to develop than adding the same feature to a traditional standalone application, it will produce greater returns in the long run, as it is used from other parts of the organisation.
- Increased Organisational Agility: as more and more of the logic in the organisation is not tied to particular applications but to services in a well-governed catalogue, creating new solutions will be faster through the reuse of these standardised building blocks.
- Reduced IT Burden: increasing reuse reduces redundancy and waste of business logic, which in turn reduces the size and the operational cost of the IT department.

The definitions proposed by OASIS and Erl are intentionally vague: they consider that any combination of technologies can be used to build a SOA. The key aspect is to follow its general principles and create a high-quality catalogue of services. In practice, however, most SOA development uses Web Services (WS) as defined by the W3C [56] and meeting the technical requirements listed in the W3C SOA definition. By using an open and standardised technology stack, users can interoperate between differing software platforms without any technical or financial barriers.

3.1.2 Web Services

The previous section examined three different definitions of SOA. Two of these definitions were mainly conceptual and regarded SOA as a different paradigm for organising enterprise IT and aligning IT with the business strategy. The definition proposed by the World Wide Web Consortium (W3C) was mostly technical, however.

This conceptual and technical split is also present for services. Services are largely conceptual entities and do not impose specific technologies. However, the need for

standardisation within the industry has pushed several organisations to provide collections of vendor-neutral specifications for their implementation.

In particular, the W3C defines a WS as [56]:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

OASIS has a similar definition of a WS [41]:

The term Web Service describes a specialized type of software, which is designed to support a standardized way for provision and consumption of services over the Web, through the compliance with open standards such as eXtensible Markup Language (XML), SOAP, Web Services Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI).

Both of these definitions mention the same basic set of technologies: WSDL for describing the format of the Simple Object Access Protocol (SOAP) messages, which are usually sent over HyperText Transfer Protocol (HTTP). In addition, OASIS mentions the Universal Description, Discovery and Integration (UDDI) standard for service discovery. The rest of this section will provide short descriptions for each of these technologies.

3.1.2.1 Hypertext Transfer Protocol (HTTP)

The HyperText Transfer Protocol (HTTP) is the underlying technology behind the World Wide Web, being used to serve all web traffic today. However, it has many other uses beyond websites, thanks to its design.

HTTP is an extensible and generic stateless server-client protocol that operates independently from the kind of information transmitted, thanks to specific provisions for typing and content negotiation. It also provides simple facilities for authentication, which are commonly combined with other standards for encryption and secure identities. Interacting through HTTP usually works as follows:

1. The client sends a request to a server, specifying a Uniform Resource Identifier (URI) a, a HTTP method, the HTTP version in use and a set of headers.

A URI is a string that uniquely identifies the remote resource. A Uniform Resource Locator (URL) is a kind of URI that indicates where the resource is. The two most common HTTP methods are GET (an idempotent request for a remote resource) and POST (a modification of a remote resource).

Clients can specify additional requirements on the desired content through headers. For instance, the **Accept-Language** header can be used to specify the preferred language for the requested content (e.g. Spanish and then English if not available).

2. The server replies with a status code (which may be a standard code or a custom one), a set of headers detailing additional information about the delivered content, and the content of the response itself.

Listing 3.1 Example of a HTTP request-response conversation

GET / HTTP/1.0

HTTP/1.0 302 Found

Location: <http://www.google.es/>

Cache-Control: private

Content-Type: text/html; charset=UTF-8

Date: Sat, 01 Sep 2012 16:03:33 GMT

Content-Length: 218

X-XSS-Protection: 1; mode=block

X-Frame-Options: SAMEORIGIN

<HTML><HEAD><meta ...>

<TITLE>302 Moved</TITLE></HEAD><BODY>

<H1>302 Moved</H1>

The document has moved

<A HREF="<http://www.google.es/>">here.

</BODY></HTML>

Status codes are divided by their first number into informational (1xx), successful (2xx), redirections (3xx), client errors (4xx) and server errors (5xx).

Using headers, the server can indicate additional information about the provided content, such as its type, language, date, cache expiration date and so on.

Listing 3.1 shows a simple usage of HTTP for retrieving <http://www.google.com>. The client connects to the server at www.google.com and sends the first line, indicating that it wants to retrieve the root resource (with URI /) using HTTP 1.0. The client terminates its request by sending two CRLF (return carriage and linefeed) sequences. The rest of the listing is the response from the server, which redirects the user to <http://www.google.es>. According to the **Content-Type** response header, the response includes an HyperText Markup Language (HTML) document in UTF-8 encoding with a link to the target URL, in case the browser does not follow the redirect automatically.

3.1.2.2 Extensible Markup Language (XML)

The most straightforward way to exchange documents through a network is to simply send the text as is using a specific encoding, without any formatting. In order to add machine-readable annotations, one approach would be to mark specific parts with a special syntax. For instance, a specific word could be marked to be displayed in bold on the screen by surrounding it with special tags, and a block of text could be marked as a section of the document. This is the basic approach followed by a *markup language*.

One of the first standards on markup languages was the Standard Generalised Markup Language (SGML) [18], published as ISO 8879:1986 [23]. SGML provided a way to define markup languages from a set of generic primitives, so developers could reuse the same

tools to parse documents written in these languages. One of these languages was the HyperText Markup Language (HTML).

HTML was largely successful due to the World Wide Web, but SGML was not widely adopted as a general-purpose document exchange format due to its complexity. Writing a SGML parser was too difficult. For this reason, W3C developed a simplified version of SGML called the eXtensible Markup Language (XML) [57].

XML documents are much easier to parse thanks to their tree-like structure. Broadly speaking, XML documents consist of regular text, opening tags and closing tags. An *element* is delimited by an opening tag and a closing tag, and it may contain nested elements or blocks of text. Elements may also contain a set of key-value pairs known as *attributes* inside the opening tag.

A XML document is formed by a single root element that contains zero or more elements. A document is well-formed if the opening and closing tags of all non-root elements are part of the same parent element, i.e. the elements are properly nested inside each other.

3.1.2.3 XML Schema

The W3C published XML as a standard for defining markup languages using a common set of concepts or tools. However, documents in a specific XML-based markup language not only need to be well-formed: they also need to be *valid*. A valid XML document conforms to the *schema* that defines the XML-based markup language itself.

XML provides a subset of the SGML Document Type Definition (DTD) schema language, but it was quite limited and could not handle some XML features, such as namespaces. This prompted the W3C to define a more advanced schema language known as XML Schema [54]. XML Schema has been widely adopted in industry: in fact, SOAP, WSDL, UDDI and many other XML-based technologies define their document formats with it.

3.1.2.4 Simple Object Access Protocol (SOAP)

The Simple Object Access Protocol (SOAP) [58] is an XML-based message format for exchanging structured and typed information between peers in a decentralised and distributed environment. SOAP also provides directives to encode certain kinds of information, several message exchange patterns and instructions on how to send SOAP messages through HTTP and regular email.

SOAP messages are formed by a single **Envelope** element, which contains an optional **Header** and a mandatory **Body**. In case of an error, the **Body** will contain a single **Fault** element with the **Code** and **Reason** of the problem. Listing 3.2 shows an example of a SOAP message that confirms that a request for a loan was granted.

3.1.2.5 Web Services Description Language (WSDL)

The W3C Web Services Description Language (WSDL) specification allows for creating machine-readable descriptions of network services. WSDL documents define operations and message at an abstract level, leveraging the XML Schema specification, and are then bound to specific network protocols and message formats into an endpoint. The concrete endpoints are finally published as Web Services at a certain URL. WSDL 1.1 provides mappings for SOAP, HTTP and Multipurpose Internet Mail Extensions (MIME) (email).

Listing 3.2 Example of a SOAP message

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <loan:requestResponse xmlns:loan="...">
      <accept xsi:type="xsd:boolean" xmlns="">true</accept>
    </loan:requestResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 3.3 shows an abridged example of a WSDL description of the WS that produced the response in Listing 3.2. The `message` elements (lines 3–10) describe the input and output messages and each of their parts. These messages are the input and output of the “approve” operation of the “loanApprovalPT” `portType` (lines 11–15), which is in turn bound to the SOAP HTTP binding using a `binding` element (lines 17–33). Finally, the concrete endpoint represented by the “ApprovalBinding” is published as the “ApprovalService” Web Services (lines 34–38), which is listening at the `http://localhost:8000/ApprovalService` URL.

3.1.2.6 Universal Description Discovery and Integration (UDDI)

The Universal Description, Discovery and Integration (UDDI) specification published by the Web Services Interoperability Organisation (WS-I) described both an XML-based global Web Services registry and the facilities required to query it [40]. UDDI providers would replicate each other’s information, much like the Domain Name System (DNS) for the World Wide Web, and WS developers would search the UDDI registry and find appropriate WS using the provided meta data. Unfortunately, that view was not realised in the long term, after some of its major backers removed their support [44].

3.1.2.7 Web Services Interoperability Basic Profile (WS-I BP)

Web Services are intended to promote interoperability through the use of open standards, such as SOAP or WSDL. However, in practice, these specifications are quite complex and not all vendors support the same subsets of features. In order to keep WS platforms interoperable with the rest, WS-I published its Basic Profile (WS-I BP) [52].

Web Services Interoperability Basic Profile (WS-I BP) imposes a set of restrictions on the usage and implementation of several of the above standards, such as SOAP, XML Schema, WSDL and UDDI. In some cases, WS-I BP forbids the usage of certain awkward combinations of features. WS-I BP may also provide additional guidance beyond what is specified in one of the standards.

Listing 3.3 Abridged example of a WSDL document

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <definitions name="ApprovalService" targetNamespace="..." ...>
3   <message name="creditInformationMessage">
4     <part name="firstName" type="xsd:string"/>
5     <part name="name" type="xsd:string"/>
6     <part name="amount" type="xsd:integer"/>
7   </message>
8   <message name="approvalMessage">
9     <part name="accept" type="xsd:boolean"/>
10  </message>
11  <portType name="loanApprovalPT">
12    <operation name="approve">
13      <input name="input1" message="tns:creditInformationMessage"/>
14      <output name="output1" message="tns:approvalMessage"/>
15    </operation>
16  </portType>
17  <binding name="ApprovalBinding" type="ns0:loanApprovalPT">
18    <soap:binding
19      transport="http://schemas.xmlsoap.org/soap/http"
20      style="rpc"/>
21    <operation name="approve">
22      <soap:operation/>
23      <input name="input1">
24        <soap:body use="literal"/>
25      </input>
26      <output name="output1">
27        <soap:body use="literal"/>
28      </output>
29      <fault name="fault1">
30        <soap:fault name="fault1" use="literal"/>
31      </fault>
32    </operation>
33  </binding>
34  <service name="ApprovalService">
35    <port name="ApprovalServicePort" binding="tns:ApprovalBinding">
36      <soap:address location="http://localhost:8000/ApprovalService"/>
37    </port>
38  </service>
39 </definitions>
```

3.2 Performance engineering

Obtaining the desired level of performance has been a regular concern since the development of the first computer systems, as shown by the early survey in [29]. There are basically two approaches: evaluating a model of a prospective system (known as *performance engineering*), or measuring the performance of an implemented system (*performance testing*).

These approaches are complementary: using analytic models reduces the risk of implementing an inefficient software architecture, which is expensive to rework [48]. When the system is implemented, measuring its performance is more accurate, and can detect not only design issues, but also bad coding practices and unexpected workloads or platform issues [3]. Some authors have proposed overloading “performance engineering” to point to both model- and measuring-based approaches [53].

This section will review some of the works that are more closely related to ours. The existing notations will be visited first, focusing on the currently available standards. The currently available performance analysis algorithms will be then discussed. Readers unfamiliar with UML are suggested to visit the introduction presented in Section 4.3.1.

3.2.1 Notations

Performance engineering usually involves building a simplified representation (a *model*) with information on each part of the system, from which the expected global performance is derived. There is a large number of works dealing with model-based testing, i.e., “the automatable derivation of concrete test cases from abstract formal models, and their execution” [50]. Most of them (as evidenced by Utting itself) are dedicated to functional testing: this section will focus instead on those dedicated to performance aspects.

The most common formalisms in performance engineering are layered queuing networks [42], stochastic Petri networks [28] and process algebra specifications [49]. These formalisms are backed by in-depth research and the last two have solid mathematical foundations in Markov chain theory. However, they introduce an additional layer of complexity which might discourage some users from applying these techniques.

Widespread adoption of UML as a *de facto* standard notation has prompted researchers to derive their analytic models from UML models, first with *ad hoc* annotations and later consolidating on standard extensions to UML, such as the Schedulability, Performability and Time (SPT) profile [33] or the Quality of Service and Fault Tolerance Characteristics and Mechanisms (QoS/FT) profile [34]. SPT extended UML with a set of stereotypes describing scenarios that various analysis techniques could take as inputs. QoS/FT had a broader scope than SPT and a more flexible approach: users formally defined their own quality of service vocabularies and used them to annotate their models.

When UML 2.0 was published, OMG saw the need to update the SPT profile and harmonise it with other new concepts. This resulted in the Modelling and Analysis of Real-Time and Embedded Systems (MARTE) profile [35], published in 2009. Like the QoS/FT profile, the MARTE profile defines a general framework for describing quality of service aspects. The MARTE profile uses this framework to define a set of pre-made UML stereotypes, as those in the SPT profile. A more in-depth description of MARTE is available in Section 6.1 (page 6.1).

3.2.2 Algorithms

Silver et al. [47] annotated each task in a workflow with probability distributions for their running times, and collected data from 100 simulation runs to estimate the probability distribution of the running time of the entire workflow and validate it using a Kolmogorov-Smirnov goodness-of-fit test. Simulating the workflow allows for flexibly modeling the stochastic behaviour of each service, but it has a high computational overhead due to the number of simulation runs that are required.

The SWR algorithm proposed in [7] computes the expected quality-of-service (QoS) of the workflow by iteratively reducing its graph model to a single task. SWR only works with deterministic values, but its QoS model can describe the cost, reliability and minimum, average and maximum times for each service. It is interesting to note that Cardoso et al. combine several data sources to estimate the QoS of each task: designer experience, previous times in the current instance, previous times in all instances of this workflow, and previous times in all workflows. The designer is responsible for specifying the relative importance of each data source.

Moving beyond workflows and into entire software systems, Bernardi et al. have defined the Dependability and Analysis Modeling sub-profile for MARTE [4]. It has been combined with the standard Generic Quantitative Analysis Modelling (GQAM) and Performance Analysis Modelling (PAM) sub-profiles of MARTE to evaluate the risk that a soft real-time system does not meet its time limits [5].

Alhaj and Petriu generated intermediate performance models from a set of UML diagrams annotated with the MARTE profile, describing a service-oriented architecture [1]: activity diagrams model the workflows, component diagrams represent the architecture and sequence diagrams detail the behaviour of each action in the workflows.

Ardagna and Pernici used a Mixed Integer Linear Programming (MILP) solver to select which services should be used in a Web Services composition written in WS-BPEL [2]. Each task had a set of candidate services, and the engine could decide at each moment which of them should be invoked by solving a MILP problem with an objective function expressed as a weighted sum of several quality of service metrics. Loops were handled by manually specifying the probability distribution of their iterations and expanding or *unpeeling* the paths according to a certain probability threshold.

3.3 Model-driven software engineering

Software programs are very complex entities: a single system may have tens of thousands of lines of code and may need to meet functional and non-functional requirements of all kinds. Describing the system will also require detailing its data structures, algorithms, deployment instructions, configuration and so on. When confronted with complex systems that need to be examined from many viewpoints, it is common in engineering disciplines to create *models* of the system.

In the context of software engineering, models are heavily used in the early development stages of a system, especially for requirements engineering and high-level design. However, these models tend to be usually only used as documentation: mapping these models to an actual system is left as a manual exercise. As a result, the return on the investment in building those models is greatly reduced, and some developers simply do not feel the need

to build these models or update them as the code changes. Some developers simply use those high-level models as throw-away documentation to sketch an idea [31].

Model Driven Software Engineering (MDSE) attempts to subvert this tendency by using models as inputs and outputs for computer-based tools that assist developers [6]. Schmidt [45] identifies several key differences between MDSE and the failed Computer Assisted Software Engineering (CASE) initiatives in the 80s and 90s:

- CASE tools needed to generate much more code than current MDSE tools, as the existing platforms lacked support for many of the features required for the models. Current tools can take advantage of the large variety of frameworks and libraries for user interfaces, networking, Web Services, and so on. Programming languages have advanced as well and are now much more concise than before.
- At the time, CASE tools did not support concurrent engineering and were hard to integrate with the traditional tools used to write regular code. Current development environments are much more extensible and can naturally integrate traditional code-centric tools with newer model-centric ones.
- CASE tools tried to represent every part of the system, in an attempt to effectively replace lower-level programming languages. In the end, the models produced were as complex as regular code. In contrast, MDSE promotes targeting notations to specific domains, making their implementation and usage much simpler.
- MDSE places a large emphasis on the creation of reusable model handling technologies. There are frameworks for transforming, validating, comparing and merging models, among many other tasks. This reduces the work involved in implementing new usage scenarios for existing models and checking the validity of the models, and allows users to suitably customise their development processes. Traditional CASE tools lacked these facilities.

The rest of this section introduces MDSE more in depth and presents some of the available technologies. Section 3.3.1 provides formal definitions for some of the concepts involved. Section 3.3.2 describes several approaches proposed to organise model-driven methodologies. Finally, Section 3.3.3 lists some of the available model handling technologies.

3.3.1 Definitions

There is little consensus on what a model is, but for this document the abstract definition by Rothenberg [43] will be adopted: “to model [...] is to represent a particular referent cost-effectively for a particular purpose”. A model may have many purposes: it may be used to specify something to be produced (e.g. as part of a contract), to predict what would happen in some scenarios, or to communicate some facts in a simpler way (e.g. as part of a lesson). However, it should be easier to use the model for that purpose than the original referent.

Seidewitz [46] identifies two important concepts related to using models in practice:

- An *interpretation* of a model is a mapping of the model’s elements to elements of the referent such that the truth value of statements in the model can be determined

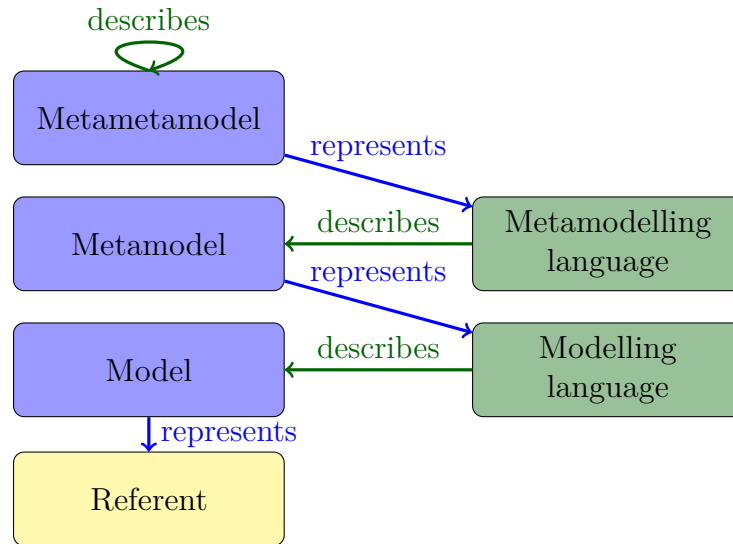


Figure 3.1. Language and model levels in MDE/MDSE

from the referent, to some level of accuracy. It is the interpretation of the model that gives it meaning.

For instance, if a UML class diagram models a system written in the Java programming language, each UML class could be interpreted as corresponding to a single Java class with the same name and equivalent methods.

- A *theory* is a way to deduce new statements about a referent from the statements already in some model of the referent. Every deducible statement from the model must be consistent with each other. Models that conform to the theory can be used to derive new information about the referent. If the theory is correct, these pieces of information should be correct as well.

Going back to the UML class diagram, the class model can suggest how the classes will be used in practice, and help verify if other models using those classes are consistent or not.

Every model is expressed in a *modelling language*. A description of a formal language is divided into three parts [22]: the set of all valid expressions (its *syntax*), the space of all the meanings that the expressions can take (its *semantic domain*) and a *semantic mapping* from the syntax to the semantic domain. In general, it is easier to deal with abstract entities rather than the actual text or graphics: for this reason, it is quite common to have a *parser* read the *concrete syntax* and produce a simplified in-memory representation known as the *abstract syntax* of the expression. This also allows for having multiple concrete syntaxes that map to the same abstract syntax.

In summary, the abstract syntax of a modelling language is the set of concepts that can be used in the models it describes. This is also known as the *metamodel* of the modelling language. The models of this language are said to *conform* to its metamodel, as they use its concepts. In turn, a metamodel may also conform to a *metametamodel* of a *metamodelling language*. Normally, *metametamodels* tend to be defined in terms of their own language, in order to stop the abstraction process. Figure 3.1 summarises these relationships.

Due to the way these layers of abstraction are based upon each other, some authors have proposed the “Mn-model” terminology, where $n \geq 0$ is the abstraction level. Some examples using UML [38] are listed below:

- A “M0-model” would be the actual referent. For instance, it could be the physical car right in front of the observer.
- The corresponding “M1-model” would be the UML *Car* class that represents the abstraction of the concept of a car. The original object would be a kind of *Car*: it could have a certain color, a certain maker and model and a license plate, among other properties.
- At the “M2-model” level, the *Car* class is a specific instance of a UML *Class*.
- Finally, at the “M3-model” level, a UML *Class* would be a Meta-Object Facility (MOF) *Class* [36]. Since MOF is defined in terms of itself, it is impossible to raise further the abstraction level.

3.3.2 Existing approaches

Models can be linked, chained and combined in any way. However, some particular approaches have received large amounts of attention from the research community and industry. This section will introduce two of them: the Model-Driven Architecture[®] (MDA[®]) initiative from the Object Management Group (OMG), and the Software Factories from Microsoft.

3.3.2.1 Model-Driven Architecture[®]

The Model-Driven Architecture[®] proposal from OMG suggests that the description of a system should be divided into three viewpoints [32]:

- The *Computation Independent Viewpoint* focuses on the environment of the system and its requirements. The details of the structure and processing of the system are hidden or not yet determined. This viewpoint uses Computation Independent Models (CIMs).
- The *Platform Independent Viewpoint* focuses on the operation of a system while hiding the details necessary for a particular platform. A platform independent view shows that part of the complete specification that does not change from one platform to another. As the name suggests, this viewpoint uses Platform Independent Models (PIMs) in a general-purpose or domain-specific modelling language.
- The *Platform Specific Viewpoint* augments the Platform Independent Viewpoint with details on how the system will be mapped on top of a *platform*, which is a set of standardised reusable subsystems, frameworks and libraries that provide domain-agnostic functionality. This viewpoint uses Platform Specific Models (PSMs).

MDA[®] attempts to separate the environment of a system from its abstract requirements and its technical requirements. Ideally, this should make migrating the system to a new technology should be much faster. As models become increasingly detailed, it is easier to

	Business	Information	Application	Technology
Conceptual	Use cases, business goals	Business entities and relationships	Business processes	Service distribution, QoS strategy
Logical	Workflows, roles	Message schemas	Service interactions and definitions	Logical server types
Physical	Process specifications	Database schemas and data access strategy	Detailed design	Physical servers

Table 3.1. 2D grid projection of a simple software factory schema [20]: the columns define concerns and the rows define levels of abstraction.

define manual or automated transformations that generate some or all of the code and/or documentation of the final system from the PIMs and the PSMs.

Interestingly, this CIM \rightarrow PIM \rightarrow PSM split is quite reminiscent of the genericity axis of the ISO 19439 model views (Section 2.3.5).

3.3.2.2 Software Factories

Software factories are a model-driven approach that borrows many concepts from manufacturing theory. The definition by Greenfield et al. [21] of a software factory is:

A software product line that provides a production facility for the product family by configuring extensible tools using a software template based on a software schema.

A *software factory schema* is a directed graph whose nodes are viewpoints and whose edges are computable relationships between viewpoints called mappings. This is a much more general representation than the layered architecture of MDA[®]. For the sake of simplicity, schemas may be projected into a two-dimensional grid, much like that of the Zachmann Framework (see Table 2.1). Table 3.1 shows a simple schema for a product line of service-oriented systems. The schema will usually need to be extended with a description of the process with which each of these viewpoints and their artefacts should be developed. Schemas may include fixed and configurable parts for the purposes of mass customisation, much like the configurable parts of a manufacturing process in Flexible Manufacturing Systems.

The schema only describes the overall process, but it does not implement it. The relevant modelling languages, patterns, frameworks and tools will need to be developed, packaged and distributed to developers. The collection of all these assets is known as a *software factory template*. Tools should be able to accept a software factory template and reconfigure themselves automatically in order to efficiently produce systems in the relevant product line.

3.3.3 Available technologies

Section 3.3.1 explained how models were normally defined using metamodels, and metamodels were in turn defined using metametamodels. These levels of abstraction are not only useful for ontological purposes: they are also important for code reuse.

There is a large number of common tasks that need to be performed on models, such as validation, transformation, code generation, comparison and so on. Writing these tasks from scratch for every metamodel would be too expensive. Instead, current model handling technologies require that the metamodel conforms to the specific metametamodel, and leverage the concepts of the metametamodel so the user only needs to provide an abstract description of how to perform the task.

Two of the most actively used metametamodels are ECore [10] and Kernel Meta Meta Model (KM3) [19]. ECore is heavily based on Essential MOF, a core subset of MOF, and implements most of the specification. However, some tools are not tightly bound to a specific metametamodel, such as the Epsilon framework [27]. The Epsilon framework implements a family of task-specific model handling languages that are not implemented directly in terms of a specific metametamodel, but rather on top of a pluggable abstraction layer (the Epsilon Model Connectivity layer) that accepts multiple modelling technologies¹. The Epsilon Object Language (EOL) is the base language for all the Epsilon languages, and it is an imperative language inspired on the OMG Object Constraint Language (OCL) [39] (described below).

The rest of this section lists a selection of currently available model handling technologies, divided by task. Due to its much wider adoption, the discussion will focus on the ECore-based ecosystem.

3.3.3.1 Metamodel definition

In addition to implementing ECore itself, the Eclipse Modeling Framework (EMF) provides a graphical editor to define ECore-based metamodels. Users can define packages, classes and fields in a hierarchical editor which mirrors the contents of the actual `.ecore` file. EMF can generate libraries for loading and saving models and generic editors from the definition of an ECore metamodel.

Several alternative notations have appeared that make metamodel descriptions more concise and easier to read and write. In addition to using models based on a subset of UML class diagrams or deriving metamodels from XML Schema files, several textual domain-specific languages have been developed. Emfatic [15] can represent all the ECore constructs, such as inheritance, containment, references and annotations, among others. Xcore is a newer Eclipse project that is set to replace Emfatic with a cleaner syntax and more advanced editors [9].

3.3.3.2 Model-to-model (M2M) transformations

From an abstract perspective, a *model transformation* can be defined as any program that takes models as inputs. However, in practice this term is usually reserved to using a model to create another model or modify an existing one.

¹As of 2012-09-02, there are Epsilon Model Connectivity (EMC) drivers for ECore, BibTeX, Command-Separated Values, Human-Usable Textual Notation, plain XML, NetBeans MetaData Repository and Z models.

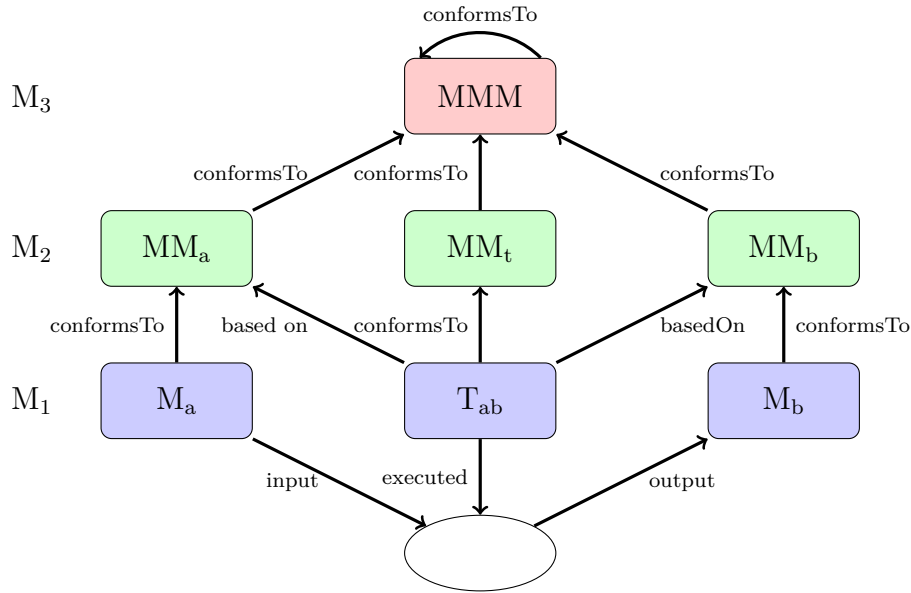


Figure 3.2. QVT and ATL transformation pattern [24]

In 2002, OMG published a Request for Proposals for a model transformation language that aligned with its standards (such as UML or MOF). One of these proposals was the ATLAS Transformation Language (ATL) [24], which later diverged into an independent open source project. The merged proposals resulted in the MOF Query/View/Transformation (QVT) specification [37]. ATL and several implementations of QVT are part of the Eclipse Model to Model Transformation (MMT) project [14].

QVT and ATL share the same transformation pattern, shown in Figure 3.2. They take a model (i.e. a M_1 -model) M_a as input and execute a transformation model T_{ab} to obtain model M_b . M_a conforms to the MM_a metamodel (M_2 model), T_{ab} conforms to the MM_t metamodel and M_b conforms to the MM_c metamodel. In turn, all the metamodels conform to a shared metametamodel (M_3 model), which only conforms to itself.

From an abstract point of view, model transformation consists of creating a mapping from the input model to the target model. This mapping can be described in a declarative manner by using patterns on the source and target elements. Alternatively, an imperative or procedural description would specify the steps required to obtain the elements of the output model from the input model. In practice, it is quite common to mix these approaches: declarative mappings are more concise and less prone to bugs, but more advanced mappings may require some implicit descriptions. QVT and ATL implement both approaches, being hybrid transformation languages.

The Epsilon framework includes three Model to Model (M2M) languages: the Epsilon Transformation Language (ETL), the Epsilon Wizard Language (EWL) and Flock [27]. ETL is a hybrid transformation language much like QVT or ATL that provides several additional features, such as support for interactive transformations or multiple input and output models. EWL is an imperative language that is specialised for in-place transformations that represent repetitive day-to-day modelling tasks. In-place transformations are a special case of the approach in Figure 3.2, in which M_a and M_b are the same model. Flock is a model migration language that specialises ETL in order to migrate models from a previous revision of their metamodel to the latest one.

3.3.3.3 Model weaving

As described above, model transformation tools implement an automated mapping from a source model to a target model. However, it may be unfeasible to provide such an automated mapping. For instance, it may be required to relate two models that were produced separately for different purposes, based on their meanings. In addition, these relations may provide more information than just a mapping: links could have a set of attributes detailing what sort of relationship exists between the two model entities.

In this case, the T_{ab} model in Figure 3.2 would be created manually instead of through a program in a model transformation language. Being a model, T_{ab} could be then used as an input of an automated transformation together with M_a and M_b .

The practice of manually creating a model that links other models together (a *weaving model*) is known as *model weaving*. One of the first proposals for model weaving was the ATLAS Model Weaver (AMW) [8]. AMW provides a generic weaving metamodel, which would take the role of MM_b in Figure 3.2.

Alternatively, users may create custom weaving metamodels and use the standard EMF tooling. This is the approach used in Epsilon ModelLink [25], a special-purpose model editor designed for creating links between two models and a specially designed weaving metamodel.

3.3.3.4 Model-to-text (M2T) transformations

One common limitation in QVT and the other M2M languages is that they cannot manipulate textual representations of the models. In order to produce text from models, a different set of technologies is required.

One of the first ECore-based Model to Text (M2T) languages was Java Emitter Templates (JET), originally part of the EMF project and now part of the Eclipse M2T project [12]. JET is inspired on Java Server Pages (JSP): templates are a mix of plain text, “scriptlets” (fragments of Java code), “expressions” (embedded strings within the output) and “directives” which define settings.

Xpand is another M2T language that provides type safety for templates and aspect oriented programming constructions, among other features, such as an improved editor with code completion. Acceleo is an actively developed implementation of the OMG MOF Model to Text Transformation (MOFM2T) 1.0 standard. Both Acceleo and Xpand are also part of the Eclipse MMT project.

The Epsilon framework includes a M2T language, called the Epsilon Generation Language (EGL). It is a template-based language, much like JET, but instead of Java fragments, it uses EOL fragments. This allows EGL to extend the available model operations using EOL, which is more concise than Java for manipulating models. In addition, EGL supports traceability reports, protected areas, language-specific formatters and modularisation.

3.3.3.5 Model validation

Models may need to meet additional restrictions beyond conforming to their respective metamodels. For instance, a UML class diagram with a *Dog* class in it may need to ensure that its *legs* field is greater or less than 4. This fact could be easily represented using a simple annotation in natural language, but more complex cases would be ambiguous and would be impossible to check using automated tools.

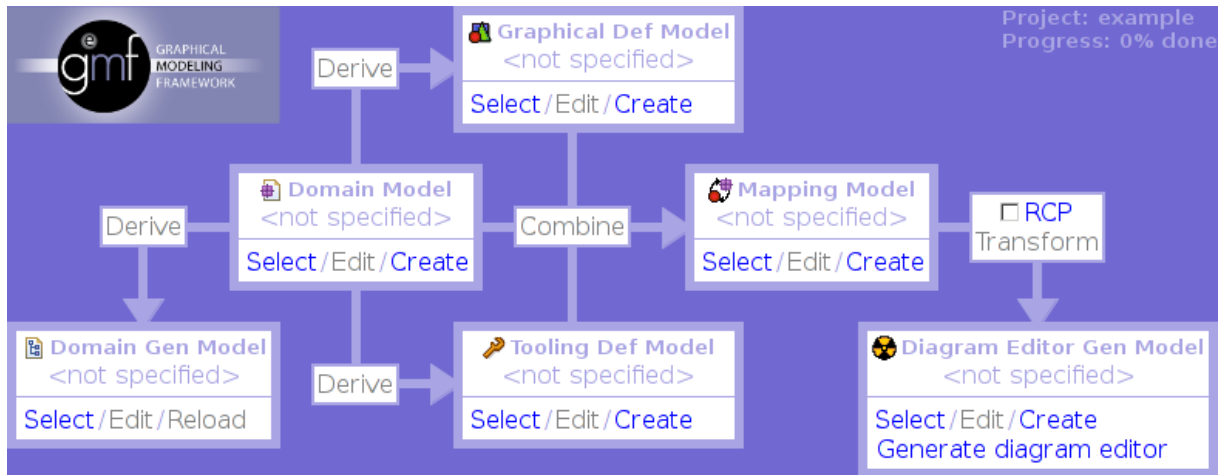


Figure 3.3. Screenshot of the GMF Dashboard view

For this reason, OMG developed the Object Constraint Language (OCL) specification, based on previous business modelling efforts and the Syntropy method [39]. The first versions of OCL could only be used with UML models, but since version 2.0, OCL can be used with any MOF-based model. OCL can be used both as a query language or as a constraint language (as in the example above). As a specification language, OCL is free of side effects: it cannot change anything in the model during evaluation, it does not implement loops and it cannot invoke operations with side effects. In addition, expressions may not always be directly executable.

The Eclipse Model Development Tools (MDT) project provides a partial implementation of OCL for ECore-based models [13]. OCL is integrated into model editors through the EMF Validation Framework [10]. Other constraint languages and libraries have been developed for this framework. In particular, the Epsilon framework provides the Epsilon Validation Language (EVL), a constraint language based on EOL that provides support for interactive feedback, warnings, inter-constraint dependencies, automated inconsistency repairing and inter-model constraints, among others.

3.3.3.6 Graphical modelling notations

As mentioned above, EMF can generate a hierarchical editor automatically from an ECore-based metamodel. However, these tree-based editors can be cumbersome to work with when modelling complex structures. A graphical editor which explicitly represented relations between models and the attributes of these entities would be better suited. On the other hand, creating such an editor from scratch would take too long for most practitioners.

The Eclipse Graphical Modelling Framework (GMF) was developed in order to reduce the cost of creating these graphical editors [11]. GMF implements a model-driven process to generate most of the code required, so developers only need to manually modify the parts that are unique to their domain. GMF uses four kinds of models in addition to the ECore metamodel: one for shapes, another for editing tools, a third one for relating the shapes and tools with the metamodel, and a fourth one for customising the code to be generated. Figure 3.3 shows a screenshot of the GMF Dashboard, which guides developers following the GMF workflow.

Though GMF saves a considerable amount of effort, creating these models is still not a

trivial task, and requires some knowledge about the underlying libraries and frameworks. Maintaining the GMF models and keeping them in sync with changes in the metamodel itself can also be difficult.

EuGENia is a tool based on the Epsilon framework that reduces the amount of work required to produce, customise and maintain the models required by GMF [26]. EuGENia takes a specially annotated ECore metamodel and the EOL-based *polishing transformations* and drives the GMF code generation process. EuGENia provides primitives for often requested primitives, such as compartments, vector-based figures, geometric shapes and so on.

Another alternative is to use higher-level libraries to build the graphical editors, instead of using a model-driven process. This is the approach taken by Graphiti [16], which hides the complexities of the underlying graphical technologies behind a simplified interface. Interestingly, this simple interface led to the creation of Spray [51], a family of textual domain-specific languages for generating graphical model editors.

References

- [1] M. Alhaj and D. C. Petriu. Approach for generating performance models from UML models of SOA systems. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '10, pages 268–282, New York, USA, 2010. ACM. doi: 10.1145/1923947.1923975. 3.9
- [2] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering*, 33(6):369–384, 2007. doi: 10.1109/TSE.2007.1011. 3.9
- [3] A. Avritzer and E. J. Weyuker. Deriving workloads for performance testing. *Software: Practice and Experience*, 26(6):613–633, 1996. ISSN 1097-024X. 3.8
- [4] S. Bernardi, J. Merseguer, and D. C. Petriu. A dependability profile within MARTE. *Software & Systems Modeling*, 2009. ISSN 1619-1366. doi: 10.1007/s10270-009-0128-1. 3.9
- [5] S. Bernardi, J. Campos, and J. Merseguer. Timing-Failure risk assessment of UML design using time petri net bound techniques. *IEEE Transactions on Industrial Informatics*, 2010. ISSN 1551-3203. doi: 10.1109/TII.2010.2098415. 3.9
- [6] J. Bézivin. On the unification power of models. *Software and Systems Modeling*, 4(2): 171–188, May 2005. ISSN 1619-1366. doi: 10.1007/s10270-005-0079-0. 3.10
- [7] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web*, April 2004. doi: 10.1016/j.websem.2004.03.001. 3.9
- [8] M. D. Del Fabro, J. Bézivin, and P. Valduriez. Weaving models with the eclipse AMW plugin. In *Proceedings of the 2006 Eclipse Modeling Symposium, Eclipse Summit Europe*, Esslingen, Germany, October 2006. 3.16

-
- [9] Eclipse Foundation. Eclipse wiki – Xcore, 2012. URL <http://wiki.eclipse.org/Xcore>. 3.14
 - [10] Eclipse Foundation. Eclipse Modeling Framework, 2013. URL <http://eclipse.org/modeling/emf/>. Last checked: November 6th, 2013. 3.14, 3.17
 - [11] Eclipse Foundation. Graphical Modeling Project, 2013. URL <http://www.eclipse.org/modeling/gmp/>. Last checked: November 6th, 2013. 3.17
 - [12] Eclipse Foundation. Main page of the Model to Text project (M2T), 2013. URL <http://www.eclipse.org/modeling/m2t/>. Last checked: November 6th, 2013. 3.16
 - [13] Eclipse Foundation. Model Development Tools (MDT) project homepage, 2013. URL <http://www.eclipse.org/modeling/mdt/?project=ocl>. Last checked: November 6th, 2013. 3.17
 - [14] Eclipse Foundation. Model to Model Transformation (MMT) project homepage, 2013. URL <http://www.eclipse.org/mmt/>. Last checked: November 6th, 2013. 3.15
 - [15] Eclipse Foundation. Emfatic project homepage, 2013. URL <http://www.eclipse.org/modeling/emft/emfatic/>. Last checked: November 6th, 2013. 3.14
 - [16] Eclipse Foundation. Graphiti project homepage, 2013. URL <http://www.eclipse.org/graphiti/>. Last checked: November 6th, 2013. 3.18
 - [17] T. Erl. *SOA: Principles of Service Design*. Prentice Hall, Indiana, EEUU, 2008. ISBN 0132344823. 3.1, 3.2
 - [18] C. F. Goldfarb. The roots of SGML – a personal recollection, 1996. URL <http://www.sgmlsource.com/history/roots.htm>. Last checked: November 6th, 2013. 3.4
 - [19] R. Gorrieri, H. Wehrheim, F. Jouault, and J. Bézivin. KM3: a DSL for metamodel specification. In *Formal Methods for Open Object-Based Distributed Systems*, volume 4037 of *Lecture Notes in Computer Science*, pages 171–185. Springer Berlin Heidelberg, 2006. 3.14
 - [20] J. Greenfield. Software factories: Assembling applications with patterns, models, frameworks, and tools, November 2004. URL <http://msdn.microsoft.com/en-us/library/ms954811.aspx>. Last checked: November 6th, 2013. 3.13
 - [21] J. Greenfield, K. Short, S. Cook, S. Kent, and J. Crupi. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, first edition, August 2004. ISBN 9780471202844. 3.13
 - [22] D. Harel and B. Rumpe. Meaningful modeling: what’s the semantics of “semantics”? *Computer*, 37(10):64–72, 2004. ISSN 0018-9162. doi: 10.1109/MC.2004.172. 3.11
 - [23] International Standards Organization. ISO 8879:1986 – information processing – text and office systems – standard generalized markup language (SGML), 1986. 3.4

- [24] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: a model transformation tool. *Science of Computer Programming*, 72(1-2):31–39, June 2008. ISSN 0167-6423. doi: 10.1016/j.scico.2007.08.002. 3.15
- [25] D. S. Kolovos. Epsilon ModelLink, 2012. URL <http://eclipse.org/gmt/epsilon/doc/modelink/>. Last checked: November 6th, 2013. 3.16
- [26] D. S. Kolovos, L. M. Rose, S. B. Abid, R. F. Paige, F. A. C. Polack, and G. Botterweck. Taming EMF and GMF using model transformation. In D. C. Petriu, N. Rouquette, and O. Haugen, editors, *Model Driven Engineering Languages and Systems*, volume 6394 of *LNCS*, pages 211–225. Springer-Verlag, Berlin, Germany, 2010. ISBN 978-3-642-16144-5. 3.18
- [27] D. S. Kolovos, L. M. Rose, R. F. Paige, and A. García-Domínguez. The Epsilon book, 2013. URL <http://dev.eclipse.org/svnroot/modeling/org.eclipse.epsilon/trunk/doc/org.eclipse.epsilon.book/EpsilonBook.pdf>. Last checked: November 6th, 2013. 3.14, 3.15
- [28] J. P. López-Grao, J. Merseguer, and J. Campos. From UML activity diagrams to Stochastic Petri nets: application to software performance engineering. *SIGSOFT Softw. Eng. Notes*, 2004. doi: 10.1145/974043.974048. 3.8
- [29] H. Lucas. Performance evaluation and monitoring. *ACM Computing Surveys*, September 1971. doi: 10.1145/356589.356590. 3.8
- [30] C. M. MacKenzie, K. Laskey, F. McCabe, P. Brown, and R. Metz. Reference Model for Service Oriented Architecture 1.0, October 2006. URL <http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf>. Last checked: November 6th, 2013. 3.1
- [31] Martin Fowler. UmlAsSketch, August 2012. URL <http://martinfowler.com/bliki/UmlAsSketch.html>. Last checked: November 6th, 2013. 3.10
- [32] Object Management Group. MDA Guide version 1.0.1, June 2003. URL <http://www.omg.org/cgi-bin/doc?omg/03-06-01>. Last checked: November 6th, 2013. 3.12
- [33] Object Management Group. UML Profile for Schedulability, Performance, and Time (SPTP) 1.1, January 2005. URL <http://www.omg.org/spec/SPTP/1.1/>. Last checked: November 6th, 2013. 3.8
- [34] Object Management Group. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QFTP) 1.1, April 2008. URL <http://www.omg.org/spec/QFTP/1.1/>. Last checked: November 6th, 2013. 3.8
- [35] Object Management Group. UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) 1.1, June 2011. URL <http://www.omg.org/spec/MARTE/1.1/>. Last checked: November 6th, 2013. 3.8
- [36] Object Management Group. Meta-Object Facility (MOF) 2.4.1, August 2011. URL <http://www.omg.org/spec/MOF/2.4.1/>. Last checked: November 6th, 2013. 3.12

-
- [37] Object Management Group. Query/View/Transformation (QVT) 1.1, January 2011. URL <http://www.omg.org/spec/QVT/1.1/>. Last checked: November 6th, 2013. 3.15
 - [38] Object Management Group. Unified Modeling Language (UML) 2.4.1, August 2011. URL <http://www.omg.org/spec/UML/2.4.1/>. Last checked: November 6th, 2013. 3.12
 - [39] Object Management Group. Object Constraint Language Specification (OCL) 2.3.1, January 2012. URL <http://www.omg.org/spec/OCL/2.3.1/>. Last checked: November 6th, 2013. 3.14, 3.17
 - [40] Organization for the Advancement of Structured Information Standards. Universal Description Discovery and Integration Standard 3.0, October 2004. URL http://uddi.org/pubs/uddi_v3.htm. Last checked: November 6th, 2013. 3.6
 - [41] Organization for the Advancement of Structured Information Standards. Web service implementation methodology, July 2005. URL https://www.oasis-open.org/committees/documents.php?wg_abbrev=fwsi. Last checked: November 6th, 2013. 3.3
 - [42] D. C. Petriu and H. Shen. Applying the UML Performance Profile: Graph Grammar-based Derivation of LQN Models from UML Specifications. In *Proceedings of the 12th Int. Conference on Computer Performance Evaluation: Modelling Techniques and Tools (TOOLS 2002)*, volume 2324 of *Lecture Notes in Computer Science*, pages 159–177, London, UK, 2002. Springer Berlin. 3.8
 - [43] J. Rothenberg. The nature of modeling. *Artificial Intelligence, Simulation, and Modeling*, pages 75–92, 1989. 3.10
 - [44] SAP News. Microsoft, IBM, SAP to discontinue UDDI web services registry effort, January 2006. URL <http://soa.sys-con.com/node/164624>. Last checked: November 6th, 2013. 3.6
 - [45] D. C. Schmidt. Model-Driven Engineering. *IEEE Computer*, 39(2):25–31, 2006. ISSN 0018-9162. 3.10
 - [46] E. Seidewitz. What models mean. *Software, IEEE*, 20(5):26–32, 2003. ISSN 0740-7459. doi: 10.1109/MS.2003.1231147. 3.10
 - [47] G. A. Silver, A. Maduko, J. Rabia, J. Miller, and A. Sheth. Modeling and simulation of quality of service for composite web services. In *Proceedings of 7th World Multiconference on Systemics, Cybernetics and Informatics*, pages 420–425. Int. Institute of Informatics and Systems, November 2003. 3.9
 - [48] C. U. Smith and L. G. Williams. Software performance engineering. In L. Lavagno, G. Martin, and B. Selic, editors, *UML for Real: Design of Embedded Real-Time Systems*, pages 343–366, The Netherlands, May 2003. Kluwer. 3.8
 - [49] M. Tribastone and S. Gilmore. Automatic extraction of PEPA performance models from UML activity diagrams annotated with the MARTE profile. In *Proceedings of the 7th Int. Workshop on Software and Performance*, pages 67–78, Princeton, NJ, USA, 2008. ACM. doi: 10.1145/1383559.1383569. 3.8

- [50] M. Utting, A. Pretschner, and B. Legeard. A taxonomy of model-based testing, April 2006. URL <http://researchcommons.waikato.ac.nz/handle/10289/81>. Last checked: November 6th, 2013. 3.8
- [51] J. Warmer, K. Thoms, M. Boger, F. Filipelli, M. Bauer, and J. Reichert. Spray project homepage, 2012. URL <https://code.google.com/a/eclipselabs.org/p/spray/>. Last checked: November 6th, 2013. 3.18
- [52] Web Services Interoperability Organization. Basic profile - version 1.1 (Final), August 2004. URL <http://www.ws-i.org/Profiles/BasicProfile-1.1.html>. Last checked: November 6th, 2013. 3.6
- [53] M. Woodside, G. Franks, and D. Petriu. The future of software performance engineering. In *Proceedings of Future of Software Engineering 2007*, pages 171–187, Los Alamitos, CA, USA, 2007. IEEE Computer Society. doi: 10.1109/FOSE.2007.32. 3.8
- [54] World Wide Web Consortium. XML Schema Part 0: Primer (Second Edition). Technical report, November 2004. URL <http://www.w3.org/TR/xmlschema-0/>. Last checked: November 6th, 2013. 3.5
- [55] World Wide Web Consortium. Web services architecture, February 2004. URL <http://www.w3.org/TR/ws-arch/>. Last checked: November 6th, 2013. 3.1
- [56] World Wide Web Consortium. Web services glossary, February 2004. URL <http://www.w3.org/TR/ws-gloss/>. Last checked: November 6th, 2013. 3.2, 3.3
- [57] World Wide Web Consortium. Extensible Markup Language (XML) 1.1 (Second Edition), August 2006. URL <http://www.w3.org/TR/xml11/>. Last checked: November 6th, 2013. 3.5
- [58] World Wide Web Consortium. SOAP version 1.2 part 0: Primer, April 2007. URL <http://www.w3.org/TR/soap12-part0/>. Last checked: November 6th, 2013. 3.5

4

Existing service-oriented methodologies

The previous chapter introduced the key concepts underlying the present work. These concepts extended from high-level business aspects (such as extended enterprises or enterprise integration architectures), to system architectures such as Multi-Agent Systems (MASs) and Service-Oriented Architectures (SOAs), or particular software development disciplines such as model-driven development or performance engineering.

One of the goals of the present Thesis was defining a methodology which could represent both the information systems of a distributed manufacturing enterprise and its tests. Since a large fraction of the information systems will be part of a SOA, this chapter will review a selection of the existing model-driven methodologies which attempt to reduce their development cost. Since none of the reviewed methodologies explicitly included testing models, one of them will be selected as a starting point, presented in more depth and extended to describe functional and nonfunctional requirements of the services to be implemented.

4.1 State of the art

Implementing a SOA is a hard task, as it can affect the entire organisation. Without a proper methodology, it can be prohibitively expensive even for a small manufacturing firm. For this reason, various SOA methodologies have been proposed to reduce their cost. Since these methodologies are very different from each other, they must be compared regarding their completeness, how much manual work they require and how costly they are.

The present discussion will focus on model-driven methodologies, which derive increasingly detailed descriptions (i.e. models) of the SOA to be built through a series of manual and automated transformations. There are other well-known methodologies such as the one proposed by Erl [11] or Papazoglou and Heuvel [28], but these are limited to textual descriptions of the steps involved, and do not explicitly involve any modelling.

Additionally, the review will be limited to methodologies that cover both identifying which services are needed, modelling the services to be implemented and (perhaps partially) implement them. There are model-driven methodologies that are entirely dedicated to specific steps in the process, such as the GAMBUSE methodology by Nguyen et al. [23], which uses gap analysis to detect differences between the existing and the desired business processes and tries to reuse as many of the existing assets as possible when listing which services are required by the organisation.

The first methodology to be reviewed is by Stojanović [30] and is based on a concept which predates SOAs: component-based development. The appropriate parallelisms between these concepts will be drawn and the steps involved will be listed. This will be followed by a discussion of the service-centric and organisation-wide Service Oriented Modeling and Architecture (SOMA) [14], SODM [7] and Business Process Service Oriented Methodology (BPSOM) [9] methodologies, and the methodology proposed by Hoyer et al. [17], which is focused on implementing service-based integration solutions. Table 4.1 summarises the most important features of these methodologies.

4.1.1 Prior work on component-based systems

Service-oriented architectures were not imagined from thin air: rather, they represent the next evolutionary step of several existing ideas in software engineering. Many parallelisms

Table 4.1. Feature comparison of the SOA methodologies under review

Name	Scope	Based on	Notations	Automated?	Cost
Stojanović	Analysis (company-wide) and design	Components, later reoriented to services	Up to the user, UML preferred (simple models)	None	Medium
SOMA	Entire lifecycle (company-wide)	UML classes with stereotypes are converted into services	UML profiles (advanced models)	Code generation and doc. generation	High
SODM	Analysis (company-wide), design and implementation (partial)	UML activities with stereotypes are converted into services	Value diagrams, BPMN business processes, UML profiles (simple models)	Descending M2M transformations and WS interface generation	Medium
BPSOM	Analysis (company-wide), design and implementation (nominal)	UML models provide high-level description of interfaces and communication protocols	BPMN business processes, SoaML UML profile, UML sequence diagrams	Descending M2M transformations (undocumented)	Medium/High
Hoyer	Analysis (for a specific integration solution), design and implementation	Domain and workflow models produce a service model, from which WSDL, XML Schema and BPEL descriptions are generated	UML activity, sequence and component diagrams	Service model generation and WSDL/XML Schema/BPEL generation	Medium

can be established between the idea of a SOA and its immediate ancestor: Component-Based Development (CBD).

There are many definitions of “component”: for the purposes of this work, it can be defined as a “black box” that is clearly separated from the rest which provides a certain functionality under some contract and which can be assembled together with other components.

In many ways, this is quite similar to the concept of a service. The main differences between services in SOA and components in CBD are:

- The way in which the part is related to the whole. Components tend to be hidden within the application and cannot be told apart from it. On the other hand, services retain their individuality even when composed into higher-level services: the same service can be used from several places.
- The abstraction level at which they operate. Components tend to be design or implementation artefacts, such as a particular user interface widget or a particular library for performing a certain task. In contrast, services provide business-level logic which is independent of the programming language used to implement the services themselves.

Taking advantage of these similarities, Stojanović [30] extends a methodology originally for CBD to SOA. There are better known component-based methodologies such as Kobra [2] or Catalysis [10], but they do not explicitly handle the case of SOA. Stojanović describes the concepts to be used in each model, allowing the user to select one among several notations for them.

The overall process followed by this methodology can be summarised as follows:

1. The business goals, processes, entities, resources, events and rules of the existing environment are captured in a set of Business Domain Models (BDMs).
2. The use cases to be implemented by the system are identified and collected in a Business Component Model (BCM). A business component brings together use cases which handle the same domain elements, change together, use the same systems and/or are part of a single transaction.
3. The applications to be used for implementing each of the business components are identified and described in the Application Component Models (ACMs).
4. Finally, these ACMs are instantiated by selecting the actual implementation artefacts they will be created from: external or in-house libraries, business logic, and so on. This step produces a set of Implementation Component Models (ICMs).

These steps are similar to the OMG MDA[®] approach (§ 3.3.2.1) which first models the business, then a platform-independent view of the system with its abstract requirements and then a platform-specific view of the system which is closer to the implementation. This methodology does not impose any particular notation, though UML is used for its case study. However, all model transformations are entirely manual, and no formal mappings are defined between them.

4.1.2 IBM SOMA

The IBM SOMA methodology defines an integrated approach over the SOA development process. It spans the entire process, from its conception to its monitoring and maintenance [14]. It consists of an iterative process divided into multiple stages.

First, SOMA requires creating a business domain model and a set of templates for the available integration solutions. The next step is finding all the services which should be part of the architecture by combining multiple information sources: enterprise goals, a conceptual model of the environment, and existing legacy systems.

Next, these services will be reorganised into a coherent whole. Since a large number of potential services may have been identified, SOMA requires selecting the subset that achieves the best balance between benefit and cost and postponing the rest. Finally, the selected services will be implemented, validated, deployed and tracked.

The main contribution from SOMA is the combination of several requirements elicitation techniques to extract as many services as possible:

- Goal-Service Modeling (GSM) extracts services by studying the business goals of the organisation and finding out in which ways the system can support these goals and the related metrics.
- Domain decomposition takes the available concepts in the business domain and extracts services from the things that the system could do with them. For instance, for an “order”, a system may create it, check its status, modify it, create an invoice for it or cancel it, among other actions.
- Analysing existing assets can also produce services by studying already available systems and finding which parts of their functionality could be reused elsewhere in the organisation. Mission-critical systems may not be replaceable in the short term, but their logic may be still useful beyond its normal bounds.

The services obtained using these three methods are then rearranged in a hierarchy, which is trimmed down with a custom litmus test that concludes whether it is worth incurring the additional cost entailed by providing the service.

The methodology has been implemented as a series of extensions to the Rational Unified Process (RUP) and uses UML with custom extensions for modelling advanced aspects.

Though it has been validated through several projects and is based upon a known process, SOMA is a complex methodology that requires creating a large set of intermediate documents using a wide array of tools. For this reason, SOMA may not be well-suited to small or medium manufacturing firms, which may not have enough resources to perform this kind of exhaustive modelling. A more lightweight approach could be more effective in these cases.

4.1.3 SODM

The Service Oriented Development Method (SODM) is a model-driven methodology for developing service-based systems [7, 8, 31]. The SODM development process follows the OMG MDA[®] proposal: while the business view encompasses the CIM level, the information systems view covers the PIM and PSM levels. Figure 4.1 shows the models used by SODM and how they relate to each other.

The business view models the environment through a collection of *value exchange models*, as proposed by Gordijn and Akkermans [15]. Existing business processes are described using BPMN activity diagrams (§ 2.4.4).

After the business view is defined, the next step is developing the PIM level of the information system view. To do this, a list of business services is extracted from the value exchange models, from which a collection of *use case models* is derived. These use case models are then subdivided and related with each other in a set of *extended use case models*, using the BPMN models as a reference.

With the extended use case models, the user has described what the system should do. The next step is specifying how the system should do it. In SODM, this is achieved by creating *service process models* that arrange the identified use cases in order to carry out a business service. These models are extended once more to produce the *service composition models*, which split the service activities between the entities that participate in the process (the *business collaborators*).

The PSM level is created from the PIM service composition models. The *extended service composition models* specify the service activities which should be deployed as reusable Web Services, and from these a set of *Web Service interface models* will be produced. These two kinds of models are linked to each other: any change to one of them will result in changes to the other. From these two kinds of models, a set of solution-specific *application business logic models* will be produced, and some of the code of the desired system will be generated.

This methodology is more lightweight than SOMA, and some of the mappings between the models are automated, unlike the methodology by Stojanović (§ 4.1.1). It uses simpler notations and strives to use the most appropriate notation at each step. At the CIM level, BPMN is used, due to its higher abstraction level and its popularity among business analysts. In contrast, UML is used at the PIM and PSM levels to make this part of SODM more accessible to software developers [8].

This methodology is a good starting base for creating a methodology for creating service-oriented systems in the context of distributed manufacturing systems. Current business practices can be captured in the available models, and then used to add more and more details until the models can generate a skeleton of the entire system.

Admittedly, it does not cover all the stages in the development process of a SOA. For instance, it does not attempt to support the testing of the software within the SOA. However, these shortcomings can be addressed by extending SODM with additional testing models and transformations.

4.1.4 BPSOM

The Business Process Service Oriented Methodology (BPSOM) was proposed by Delgado et al. [9] as an extension of the existing practices of a software development firm (as shown in Figure 4.2). It defines a model-driven process for developing services from the business processes in the organisation.

Like SODM, it is based on the OMG MDA approach (§ 3.3.2.1). Business processes constitute the CIM level and are modelled using BPMN. The business process models are transformed using QVT (Query/View/Transformation [24], § 3.3.3.2) to a collection of SoaML (Service oriented architecture Modelling Language [27]) models. SoaML provides a metamodel and a UML profile for the specification and design of services within a service-

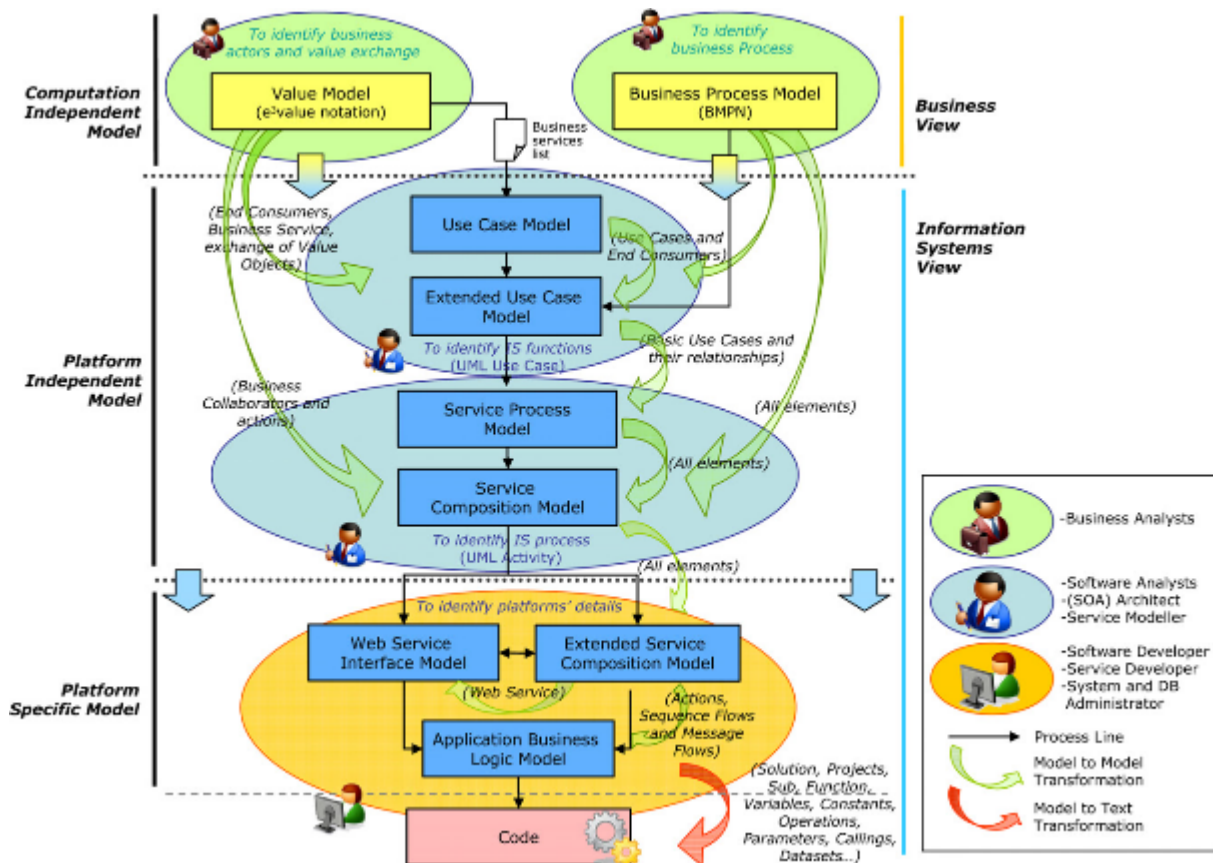
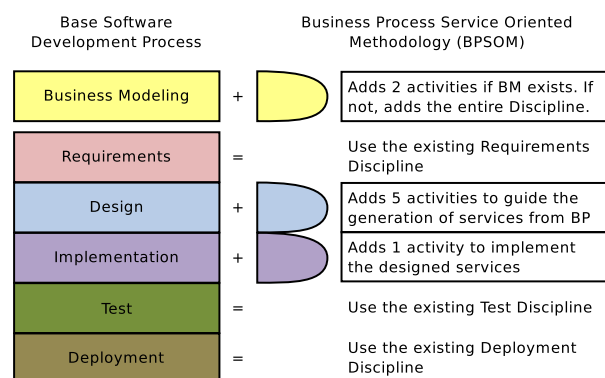


Figure 4.1. Outline of the models used by SODM [8]



Support Disciplines are used as defined in the base process.

Figure 4.2. Integration of BPSOM into the existing software development process [9]

oriented architecture. SoaML provides several ways in which services can be described, classifying them into simple interfaces for Web Services in the RPC (Remote Procedure Call) style, service interfaces (bidirectional WS with different interfaces for consumer and provider) and service contracts (which entail the execution of particular choreography).

BPSOM is divided into the following activities:

- *BM1: Assess the target organisation*, introducing the project team to the organisation for which the development will be carried out.
- *BM2: Identify business processes* and model them with BPMN.
- *D1: Identify and categorize services* as required to implement the business processes. The SoaML models will describe the Service Architecture (SA) listing the participants, the service contracts and the roles the participants play in them.
- *D2: Specify services* by filling out the rest of the details in the SoaML models, such as interfaces, operations or input and output parameters.
- *D3: Investigate existing services* which are listed in a central catalogue, in case they can be reused.
- *D4: Assign components to services*. These components will eventually implement the functionality of the service.
- *D5: Define services interaction* through UML sequence diagrams that describe the service orchestrations and choreographies throughout the system.
- *I1: Implement services*, in which the components that comprise each of the services are implemented.

BPSOM is quite similar to SODM, which is to be expected, as they are both based on the OMG MDA approach. Both methodologies start from business process models in BPMN and use them to create PIM-level models and then the code. While SODM uses custom models at the PIM and PSM levels instead of SoaML models, in the end both methodologies generate code from a detailed description of the selected services.

BPSOM does not provide any facilities for verification and validation: it only extends the development process with the activities that are strictly required for creating the services, but not for testing or deploying them. Several tools for driving the execution of the methodology have been published, but the QVT transformations themselves or their descriptions are not publicly available.

4.1.5 Hoyer

Hoyer et al. defined a top-down model-driven methodology for creating service-oriented applications in [17]. It has a smaller scope than SODM and BPSOM, being focused on the development of a particular integration solution rather than a company-wide SOA.

The first step of the methodology is eliciting the general requirements for the integration solution, using whichever techniques are available (though the authors recommend creating UI prototypes). The requirements themselves do not need to be in any particular notation.

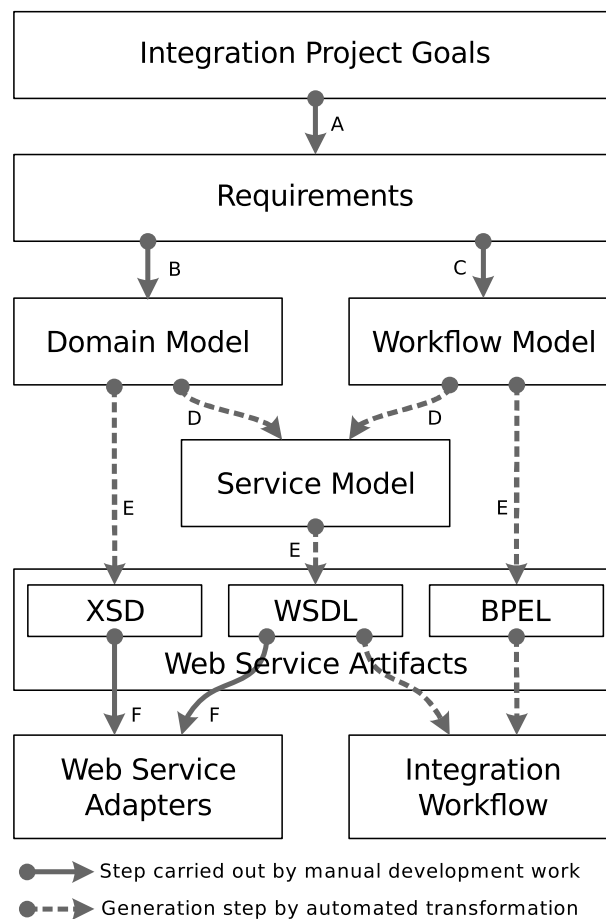


Figure 4.3. Overview of the model-driven development process by Hoyer et al. [17]

Next, these requirements will be used to manually derive a set of domain models (UML class diagrams) and workflow models (UML activity). These will be automatically transformed using QVT into a service model, consisting of a set of UML interfaces and components. From these three UML models, the methodology generates the required XML Schema datatypes, WSDL interfaces and WS-BPEL service compositions for the particular integration solution. The final step requires the user to implement the desired Web Services from the WSDL and XML Schema documents: this can be partly automated depending on the target programming language and frameworks.

The overall approach is shown in Figure 4.3. The methodology uses a small number of models in comparison to SODM and BPSOM, and it is quite developer-centric, judging from the exclusive usage of UML during the modelling stages. A closer notation to that used by business analysts (such as BPMN) would be better suited to capture workflows. However, BPMN cannot be easily translated to WS-BPEL, as suggested by Weidlich et al. [32], so this might have been intentional.

4.2 Selection of a base methodology

In the previous section, several of the existing model-driven methodologies for developing SOAs were reviewed, producing the results shown in Table 4.1. All these methodologies used top-down approaches, starting from high-level business processes and using various techniques to select which business functionality should be made available throughout the organisation as reusable services. However, they largely varied in several important aspects, such as their scope, their degrees of automation, their cost or the complexity of the models being used.

In order to decide which methodology should be used as a starting point, several requirements were set:

- R1. The methodology must be extensible to the entire manufacturing company, as the SOA will need to integrate the manufacturing plants of the company and its partners in the value chain.
- R2. The methodology must provide at least partial automation for producing the analysis and design models.
- R3. The methodology must be publicly available. Ideally, a reference implementation should be available as well, but it is not required.
- R4. The methodology should use well-known notations and simple models, keeping its cost within the resource constraints present in a small-medium enterprise.

Requirement R2 excluded the methodology by Stojanović. Requirements R1, R3 and R4 excluded SOMA. Hoyer was excluded due to R1. This left only BPSOM and SODM. Among these, SODM was selected as it provided detailed descriptions of the transformations involved in the analysis and design stages, used a more accessible subset of the UML notations and used value diagrams to model the business model motivating the supporting business processes and information systems. It is also important to note that graphical SoaML modelling tools are currently limited to costly proprietary options, such as MagicDraw, IBM Rational Software Architect or Enterprise Architect.

4.3 Detailed description of SODM

Having selected SODM for the above reasons, this section will describe the models used by SODM (shown in Figure 4.1) and the concepts involved in more depth. Due to the interdisciplinary nature of the present thesis, it is assumed that some readers may be unfamiliar with software engineering notations such as UML. After a short introduction to the subset of UML used by SODM, each of the models used by SODM will be visited, starting from the highest-level models up to the models closest to the implemented system.

4.3.1 UML subset used by SODM

SODM uses several kinds of models defined by the Unified Modelling Language (UML) published by the Object Management Group (OMG) [25]. These are class diagrams, use case diagrams and activity diagrams. As UML is a very complex specification, this section will only present the subset of UML that SODM uses in each kind of diagram.

UML can be extended using “profiles”. Profiles add new “stereotypes” that can be applied to certain kinds of model elements and specify additional semantic information. The extensions provided by SODM will be detailed below, followed by some new proposed extensions for testing functional and non-functional requirements.

4.3.1.1 Class diagrams

Class diagrams represent each of the concepts that will be handled by the system, according to the object-oriented programming paradigm. A “class” is an abstract concept that represents all “objects” of a certain kind. For instance, while “wheeled vehicle used for transporting passengers, which also carries its own engine or motor” is the abstract concept of an *Car*, “that car” is an object of the underlying class.

All objects of a class have certain pieces of information associated with them (known as “attributes”) and provide various behaviours (“methods”).

A is said to be a subclass of *B* when all the objects of *A* are also objects of *B*, i.e. the objects of *A* are a specific kind of objects of *B*. Continuing the previous example, *SUV* would be a subclass of *Car*, as every *SUV* is also a *Car*. This relation can also be expressed in other ways: *B* is a superclass of *A*, *A* inherits from *B*, *B* is a parent of *A*, or *A* is a descendant of *B*. Subclassing is useful as the objects in these subclasses will inherit all the attributes and methods of their superclasses: for instance, since every *Car* will have an **engine** attribute, *SUV* will implicitly have it as well.

There are several ways to specify inheritance relationships in UML, but these diagrams will only consider the most common kinds: complete and disjoint. “Complete” means that there are no more subclasses beyond those explicitly drawn in the diagram, and “disjoint” means that a certain object may only belong to one subclass. This means that if only the *SUV* and *Bus* subclasses are defined, it is assumed that no *Sedan* objects exist.

Objects in a class may be related to objects in another class. This is represented using an “association” between the classes, which may have a descriptive name and in which each of the participating classes will take a certain role. These roles may also have descriptive names. An association can also impose limits on how many objects of each class can be related to each other at the same time, by specifying a “cardinality” on each role. For instance, it can be assumed that an *Car* is owned by one *Person* and that one *Person* may

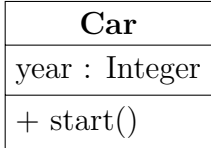
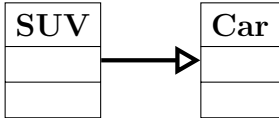
own zero or more *Cars*. The resulting association “owns” will have two roles: “owner” for *Person* (with cardinality “1”) and “possession” for *Car* (with cardinality “*”).

Objects may contain other objects through “composition” or “aggregation” associations:

- In a composition, the container has the exclusive ownership of the contained object and if the container is destroyed, the contained object will be destroyed as well. A composition from *Car* to *Engine* would indicate that an *Engine* is part of exactly one *Car*, and that the *Engine* would be discarded together with the *Car*.
- Aggregations do not require exclusive ownership and do not limit the lifetime of the contained object. For instance, a *Family* may be the aggregation of many *Person* objects. However, the same *Person* may belong to multiple *Family* objects or may also belong to a *SportsClub*, for instance. If the *SportsClub* disbands, this will not affect the *Person* objects in it.

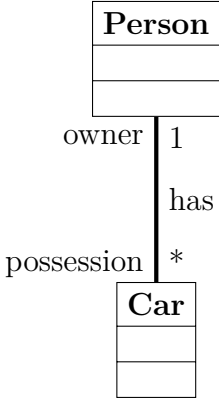
The notation used in class diagrams is presented on Table 4.2.

Table 4.2. UML class diagram concepts used by SODM

Name	Notation	Meaning
<i>Class</i>		<p>Describes a set of objects that share the same specifications of features, constraints and semantics. All objects in a class have various fields with pieces of information about them (attributes) and certain behaviours (methods).</p> <p>The example on the left shows a class that represents all <i>Cars</i>. The year attribute indicates its registration year and the start method starts its engine.</p>
<i>Generalisation</i>		<p>Relation from a source (child) class <i>A</i> to a target (parent) class <i>B</i>, indicating that every object of <i>A</i> is also an object of <i>B</i> and should therefore inherit its attributes and methods. On the left, <i>SUVs</i> are described as a particular kind of <i>Cars</i>.</p>

Continues on next page

Continued from previous page

Name	Notation	Meaning
<i>Association</i>	 <pre>classDiagram class Person { } class Car { } Person "1" -- "*" Car : has note for Person "owner" note for Car "possession"</pre>	<p>Relation between a certain number of objects of one or more classes. An association may link two or more “ends”. The number of objects at each end is limited by the “cardinality” of that end. For instance, if a certain end has cardinality 2, that means that after selecting specific objects for all other ends, only two objects of the class at that end may be related to them. Each end plays a certain “role” in the association. If an association end is navigable, instances at that end will be easy to list from instance at the other ends. The simplest associations are binary and bidirectional (both ends are navigable), but they need not be. If only one of the ends is navigable, it is marked with an arrow tip. Association and role names may be omitted if desired. Cardinalities are assumed to be 1 by default. The “has” association on the left models the fact that a <i>Person</i> may have zero or more <i>Cars</i>, and every <i>Car</i> will be owned by exactly one <i>Person</i>.</p>

Continues on next page

Continued from previous page

Name	Notation	Meaning
Composition	<pre> classDiagram class Car class Wheel class LicensePlate Car "1" *-- "4" Wheel Car "1" *-- "1" LicensePlate </pre>	<p>Binary association in which one of the ends is “owned” by the other. This means that if the object at the “owner” ends is destroyed, the “owned” objects will cease to be as well. The cardinality of the “owned” will indicate how many objects of its class are owned by each instance of the “owner” class.</p> <p>The example on the left shows that a <i>Car</i> has a <i>LicensePlate</i> and four <i>Wheels</i>. Since the end for <i>LicensePlate</i> in its association with <i>Car</i> does not have an arrow tip, the model also shows that one can find the <i>Car</i> instance from its <i>LicensePlate</i>.</p>
Aggregation	<pre> classDiagram class Team class Person Team "1" o-- "1..*" Person </pre>	<p>Binary association which models a whole-part relationship, much like a composition. Unlike a composition, the part can belong to multiple wholes and does not need to be destroyed together with any of them.</p> <p>The example on the left shows that a <i>Team</i> is formed by one or more <i>Person</i> objects, and that a <i>Person</i> may be part of several teams. Even if a <i>Team</i> is disbanded, the <i>Person</i> objects in it will remain.</p>

4.3.1.2 Use case diagrams

Use case diagrams model the scenarios in which the stakeholders in the business environment in which the system will operate (the “actors”) will use the system to obtain a tangible result (their so-called “use cases” for the system).

These diagrams link each of the actors with the use cases in which they have an interest on, whether it is because they interact with the system in some form, or because they affect or are affected by the results (e.g. a tax authority on a tax ledger module).



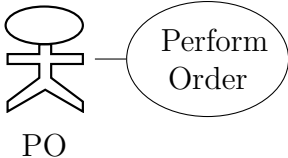
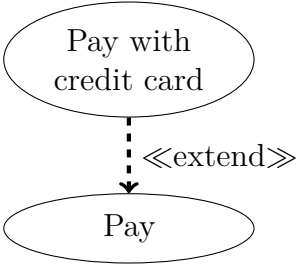
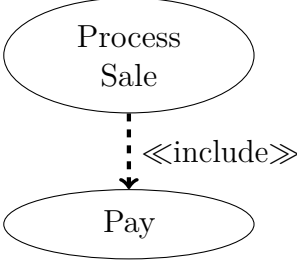
Use cases may be linked to each other as well, if the modeller considers their contents to be related in some form. A use case may include the steps followed by another use case, or may extend those of another use case.

Normally, use cases are presented inside a graphic element that represents the boundary of the system being modelled, and the actors are placed just outside this boundary. This

helps visualise what is the system supposed to do and not, and who is affected in some way by its operation.

The notation used for the use case diagrams in SODM is summarised in Table 4.3.

Table 4.3. UML use case diagram concepts used by SODM

Name	Notation	Meaning
<i>Actor</i>	 Salesman	Person or entity that is affected by or that interacts with the system in some form. For instance, the above <i>Salesman</i> in a system managing a general store.
<i>System boundary</i>		Limit between what the system will do and will not. Use cases will be placed inside this boundary, and actors will be placed outside it. Due to space restrictions, this table will only use a system boundary in this row.
<i>Use case</i>	 PO	Sequence of actions performed by the system to produce a specific result of some value to one or more of the actors in the diagram. The <i>Perform Order</i> use case on the left produces something of value for the <i>Salesman</i> actor and is therefore linked to it. Optionally, the link can have an arrow tip at the end next to the use case.
<i>Extension</i>		Relation between two use cases in which one of them extends the behaviour of the other. Here, <i>Pay with credit card</i> only extends <i>Pay</i> with the additional details required to handle this particular form of payment.
<i>Inclusion</i>		Relation between use cases in which one of them includes the behaviour of the other. For instance, in order to process a sale, the system will necessarily need to process the payment of the requested amount.

4.3.1.3 Activity diagrams

UML activity diagrams list the steps required to perform a specific task in the system. They specify the order constraints between these steps, how they may overlap and under which conditions certain steps will be executed or not. In this sense, they are quite similar to flowcharts.






In every activity diagram, execution starts from an initial node and continues through the control flows until a final node is reached. Intermediate nodes may be of the following types:

- Subactivities or actions that perform a concrete task.
- Decision nodes, which select one of several outgoing control flows depending on a set of conditions.
- Fork nodes, which continue execution concurrently through all their outgoing control flows.
- Join nodes, which wait for all incoming control flows to converge before continuing execution through its only outgoing control flow.
- Merge nodes, which reunite several execution branches into one. Unlike join nodes, execution will continue as soon as any of the incoming control flows is activated.

In addition to control flows, object flows may be used in order to pass information between steps. Nodes may be also placed on the swimlane of the stakeholder which will perform the task.







Table 4.4 summarises the notation used for the activity diagrams in SODM.

Table 4.4. UML activity diagram concepts used by SODM

Name	Notation	Meaning
<i>Initial node</i>		Unique starting point for all execution paths.
<i>Activity final node</i>		Ends all execution branches of the current activity.
<i>Flow final node</i>		Ends the current execution branch. No other execution branches will be affected.
<i>Fork node</i>		Divides the current execution branch into several concurrent branches. Fork nodes have several incoming flows and no outgoing flows.
<i>Join node</i>		Waits for all incoming flows to reach it, and then continues execution through its only outgoing flow.

Continues on next page

Continued from previous page

Name	Notation	Meaning
<i>Merge node</i>		Reunites several execution branches into one, like the join node. Unlike the join node, execution will continue as soon as any of the incoming flows is activated. Has several incoming flows and only one outgoing flow.
<i>Decision node</i>		Activates the outgoing control flow whose condition holds true. Has several outgoing flows and only one incoming flow.
<i>Activity or action</i>		Task to be performed by the system or a stakeholder. While activities can be decomposed into lower-level entities, actions represent atomic entities.
<i>Object node</i>		Object or message passed between two activities or actions. These are normally used to represent the inputs and outputs of the activities and actions.
<i>Control flow</i>		Link between two nodes indicating that once execution is over for its source node, it will continue through the target node.
<i>Object flow</i>		Link which connects an activity or action with an object node. If the object node is the target of the link, it means that it is an output of the activity or action. Otherwise, it is an input of the action.

4.3.2 Computation-independent models

In the previous section, the subset of UML used by SODM was described. The rest of this section will present the abstract and concrete syntaxes of each of the models used in every step of SODM, and how they are combined.

The first step in all methodologies based on the MDA approach by the OMG, as SODM, is modelling the business environment itself. This will help ensure that the system built is aligned to the needs of the organisation requesting it.

These models do not need to take into account any design or implementation detail of

the system to be built. Their intent is to formalise the current workings of the organisation, the stakeholders involved and their interests.

4.3.2.1 Value models

The first models required by this methodology are the *value models* proposed by Gordijn and Akkermans [15] and implemented in [16]. These models present the business model upon which the enterprise operates as a collection of value exchanges between the participants in its environment. The abstract concepts of the value models used in the present Thesis, their meaning and their graphical notations are listed in Table 4.5.

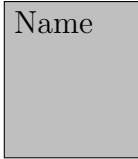
These exchanges are described at an abstract level, without going into details about how they are actually performed. For instance, the models do not specify whether a particular financial transaction is done in cash or by electronic means, or the means of transport used for supplying a certain kind of goods.

Reading a value model requires starting from the *start stimulus*, a special node that signals that a customer has a certain need that must be fulfilled through a sequence of value exchanges until reaching a set of *end stimulus*.

Creating a value model from a textual description of the organisation requires following these steps:

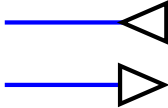
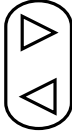
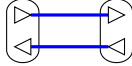
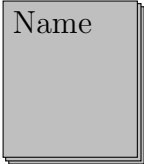

1. Identify the operational scenario, i.e. which products, services or experiences are requested by the customers.
2. Identify both kinds of actors: those whose needs will be fulfilled, and those who will help fulfil them.
3. Identify the objects by studying what is sent and received to and from the actors.
4. Identify the dependency paths from the start stimuli to the end stimuli, listing the sequences of value exchanges required to fulfil the needs of the customer.

Table 4.5. Abstract and concrete syntax of Gordijn value models

Name	Notation	Meaning
<i>Actor</i>		Self-sufficient economic entity, such as an enterprise or an end consumer. “Self-sufficient” means that it can obtain profits after a certain time period (for an enterprise), or that it can obtain a certain added value (for an end consumer). Valid business models must ensure that all actors are self-sufficient.
<i>Value object</i>	<u>Name</u>	Kind of service, product or experience which has economic value for at least one of the actors in the model.





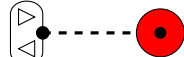
Continues on next page

Continued from previous page

Name	Notation	Meaning
<i>Value port</i>		Point from which an actor sends or receives value objects.
<i>Value offering</i>	(no graphical representation)	Everything sent or received by an actor during a particular exchange with its environment. It consists of a set of value ports in the same direction (sending or receiving), associated to the same value interface. It models the restriction that in order to be performed, value objects must be sent and/or received through all the ports involved at the same time.
<i>Value interface</i>		Pair of two value offerings: the offering made by an actor to its environment, and the offering that is expected to be received in exchange. In order to be activated, both offerings must be performed at the same time. These are drawn at the edges of the actors and market segments.
<i>Value exchange</i>		Link between two value ports, noting that the actors are willing to exchange value objects between each other.
<i>Market segment</i>		Category of actors that share the same value interfaces. A particular actor in a segment may have additional value interfaces: these will be modelled separately as a supplementary actor.
<i>Value activities</i>		Activities performed by an actor in order to obtain some kind of added value. These are drawn inside the actor that performs them.

Continues on next page

Continued from previous page

Name	Notation	Meaning
<i>Dependency node</i>	A 	Nodes which describe the way in which the needs of the end customers depend upon the execution of a sequence of value exchanges throughout the actors in the model, starting from a starting stimulus (A) and ending at a set of end stimulus (B). Some exchanges may require several other exchanges, which may all have to be performed (drawn as an AND fork, C) or at least one of them (OR fork, as in D). In other cases, some exchanges will have to be performed after every previous exchange is done (an AND join, C) or after at least one of the previous exchanges is done (an OR join, D).
	B 	
	C 	
	D 	
<i>Dependency path</i>		Connects a dependency node to a value interface.

4.3.2.2 Business process models

The first versions of SODM used UML activity diagrams to describe the business tasks needed to obtain a tangible result for the customer [7]. These tasks were high-level abstract tasks which had to be realised through several lower-level tasks in the information system: business process models are only intended to capture the current and desired states of the business practices or the organisations, and not for directly implementing the system.

Business process models for the latest versions of SODM have the same focus, but use the Business Process Modelling Notation 2.0 (BPMN 2.0) instead [8]. BPMN 2.0 was introduced in Section 2.4.4, together with an example based on a case study in Section 2.4.5 and Figure 2.24. Readers are suggested to refer to these sections for an overview of the abstract concepts and notation used.


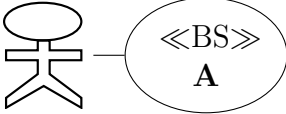
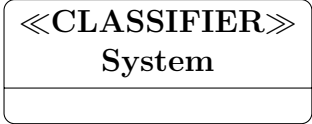
4.3.2.3 Business services list

The list of business services simply enumerates all the services that are provided by the organisation in order to satisfy the needs of the customers. Each entry includes the name of the service and the customer whose need is fulfilled.

Creating the list requires first identifying who will serve as customers for the services of the organisation. The value model is used as a starting point: every actor containing a start stimulus in some dependency path is a customer, requiring a sequence of value exchanges in order to fulfil her needs.

Next, the services required by the customers need to be identified. These can be extracted from the value models by traversing them from the start stimulus and collecting

Table 4.6. New or changed elements in the SODM use case models

Name	Notation	Meaning
<i>End consumer</i>	 Customer	Specific kind of actor that represents a user that will obtain a particular business service through the system.
<i>Business service</i>	 Customer	Service provided by the system that produces a result with tangible value for the end consumer.
<i>System boundary</i>		The meaning is the same as before, but the notation has been changed to that in the left.

the value activities that need to be performed and the value exchanges that need to be facilitated, or by examining the activities in the business processes involved.

4.3.3 Platform-independent models

Through the computation-independent models presented above, SODM can represent the enterprise and its environment and elicit the list of the business services required by the customers.

In this second stage, SODM will produce a collection of models which will describe the functionality required by the system, without regards for specific implementation details, such as the technologies to be used or the environment in which the system will be deployed.

4.3.3.1 Use case models

Use case models in SODM relate the business services to be provided by the system to their consumers. In general, these are fairly standard UML use case diagrams (as those in Section 4.3.1.2), with some slight changes (see Table 4.6).

These models describe a particular kind of use case, known as a “business service”. Each business service is linked with exactly one “end consumer”, a particular kind of actor. In addition, each end consumer uses exactly one business service.

In order to create the use case models, modellers will need to consider exactly which will be the scope of the system and who will use it. Not all business services may be directly offered as reusable services, and some services will not be used directly by the end consumers, but rather by some third parties (e.g. regular staff instead of management).

After deciding which business services will be provided as reusable services by the system and who will be the end consumers for these services, modellers will add the appropriate links to relate them.

4.3.3.2 Extended use case models

The extended use case models take the above business services and decompose them into “composite” use cases, which are then broken up into several “basic” use cases, which are easier to implement.

The decomposition of the business services needs to be based on the information in the value models and the business process models. For instance, it may be necessary to implement additional functionality in order to facilitate a specific value exchange, or to perform some of the actions in the service process model.

Furthermore, these extended use case models will include inclusion and extension links between the basic and composite use cases. These links will be used to create a first version of the service process models, as shown below.

SODM also classifies use cases as “structural” or “functional”. Structural use cases are associated with the day to day management of certain kinds of resources in the system (such as “Create order”), whereas functional use cases implement more advanced logic.

These deviations from the standard use case models in UML are collected in Table 4.7.

4.3.3.3 Service process models

The extended use case models created above present an overall view of the functionality that must be provided by the system. The next step is to specify how these pieces are assembled in order to render the services required by the end consumers. This is the role of the service process models.

Service process models are simplified UML activity diagrams which string “service activities” (a particular kind of activity) together. SODM can help users generate a first draft of the service process model from the extended use case models, by converting basic use cases into service activities and applying the rules in Table 4.8 to interconnect them. Service process models do not use lanes, as the assignment of the activities to particular stakeholders has not been considered yet. An example of a service process with some proposed extensions is later shown in Figure 4.6.

4.3.3.4 Service composition models

Service composition models take the previous service process models and decompose service activities into lower-level actions which can be directly implemented into the system. These actions are then assigned to specific participants or “business collaborators”, modelled as swimlanes. Collaborators send and receive information through the exchange of object nodes. An example of a service composition with some proposed extensions is shown below in Figure 4.7.

Business collaborators are extracted from the value models. Every service composition model should take into account every actor or market segment that performs a business activity related to the service composition.

4.3.4 Platform-specific models

The last stage of SODM defines a set of models which depend on the actual implementation technologies. According to the OMG MDA approach (§ 3.3.2.1), these are the Platform

Table 4.7. New or changed elements in the SODM extended use case models


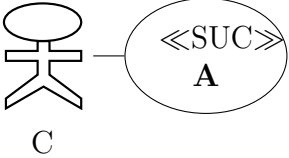
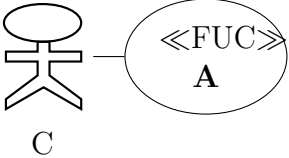
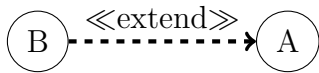
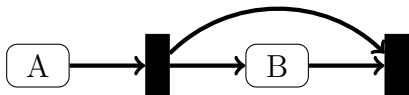
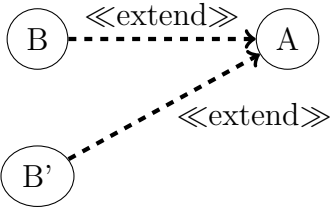
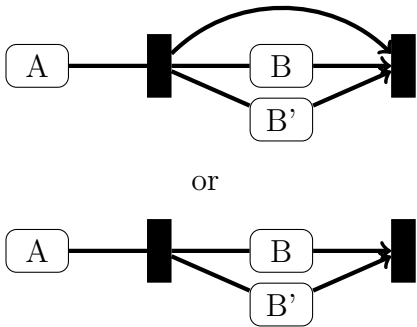
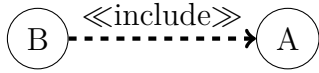

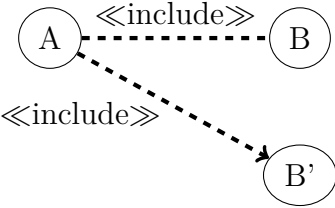
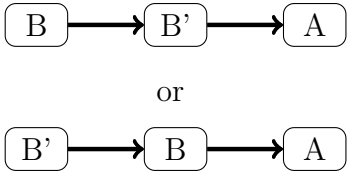
Name	Notation	Meaning
<i>End consumer</i>	 Customer	Specific kind of actor that represents a user that will obtain a particular business service through the system.
<i>Composite use case</i>	Depends on whether it is structural or functional.	Set of actions required to provide a business service. It consists of several basic use cases, and it can be either structural or functional.
<i>Basic use case</i>	Depends on whether it is structural or functional.	Set of actions required to provide a business service. It can be either structural or functional.
<i>Structural use case</i>	 C	Set of actions required to provide a business service that manipulates a specific kind of resource in the system. It can be either composite or basic.
<i>Functional use case</i>	 C	Set of actions required to provide a business service which implements a higher-level logic beyond that of a structural use case. It can be composite or basic.
<i>System boundary</i>	<div style="border: 1px solid black; padding: 5px; width: fit-content;"> <<CLASSIFIER>> System </div>	The meaning is the same as before, but the notation has been changed to that in the left.

Table 4.8. Transformation rules from SODM extended use case models to service process models

Extended use case model	Service process model
	
	 <p style="text-align: center;">or</p>
	
	 <p style="text-align: center;">or</p>

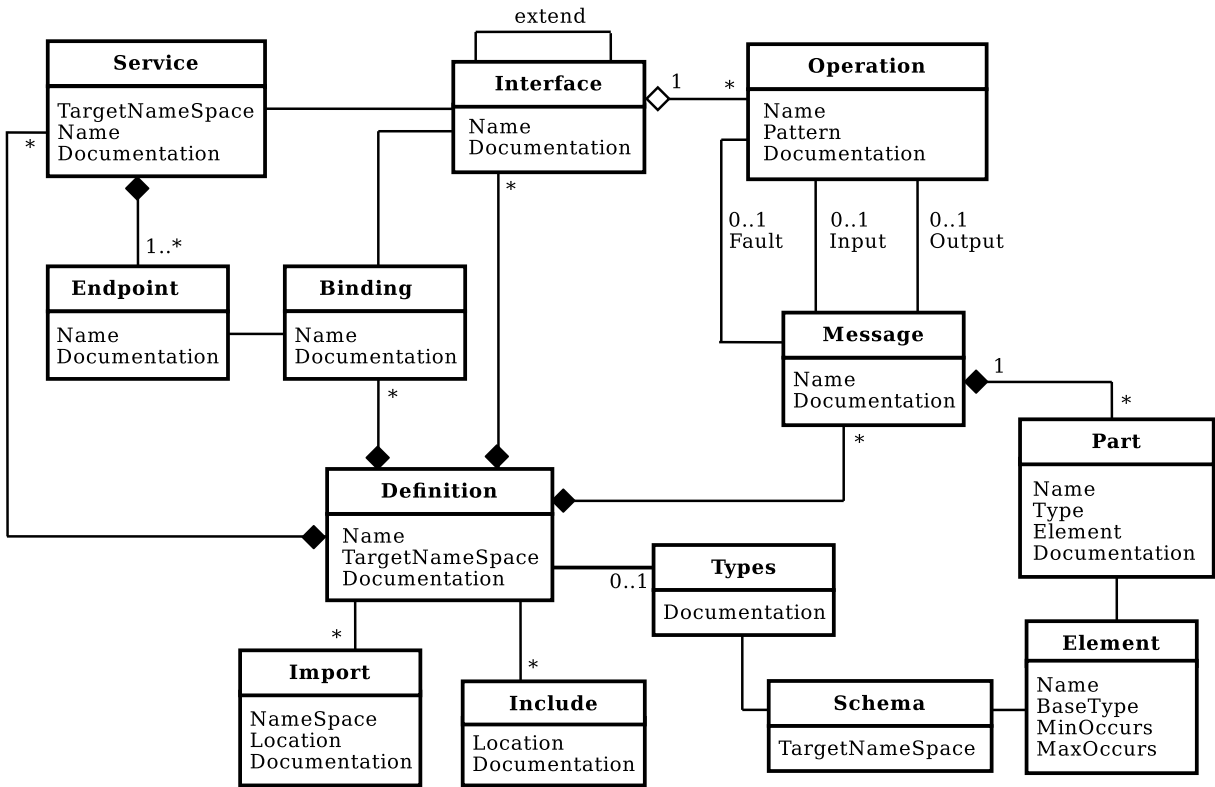


Figure 4.4. WSDL 2.0 metamodel [7]

Specific Models (PSM). SODM uses two kinds of PSMs: extended service composition models and Web Service (WS) interface models.

The SODM extended service composition models are almost the same as the original service composition models (§ 4.3.3.4), except for the new $\ll\text{WS}\gg$ stereotype. $\ll\text{WS}\gg$ marks the actions in the models which should be made available through the SOA as reusable services.

The metamodel for the SODM WS interface models is a straightforward mapping of the core concepts of the the WSDL 2.0 [34] standard. The WSDL 2.0 metamodel is shown in Figure 4.4, and consists of the following concepts:

- The root of every model is a *Definition* instance, with a human-readable *name* and a unique *TargetNameSpace* URI. This URI decorates the names of all the services, bindings, interfaces and messages in the model, reducing the risk of a name collision.

Definitions may also include *Import* objects to load definitions from external WSDL documents, or *Include* objects that reuse external XML Schema [33] type and element definitions.

- Each *Service* listens for requests at several *Endpoints*, which are associated with a particular technology (SOAP, REST, email and so on) through a *Binding*.
- Each *Binding* exposes a certain *Interface* through the specified technology. The information in the binding will be used to encode and decode the exchanged messages appropriately into and from the actual communication medium.

- The *Interface* of a *Service* is an aggregation of zero or more *Operations*. Each *Operation* is invoked through a certain message *Pattern* to access a specific functionality of the *Service*. The predefined message patterns in WSDL 2.0 may use zero or one *Input*, *Output* or *Fault Messages*.
- Every *Message* consist of zero or more *Parts*, which have a name and conform to a certain XML Schema *Type* or *Element*.

Based on this metamodel, SODM proposes a UML profile for class diagrams that captures most of its contents, leaving the rest for a later transformation. The metamodel is shown in Figure 4.5 in page 4.26: it closely mimics the concepts shown above. Most of the stereotypes are applicable to UML *Classes*, except for several *Association* stereotypes which relate *Operations* with *Messages*, *Parts* with *Elements* and *Definitions* with *Elements*.

4.4 Extending SODM for testing

From the detailed description above, it can be concluded that SODM is a lightweight methodology that focuses on the description of the business environment, the functional requirements of the system and the WS that should be built. Therefore, it could be used with small to medium companies that do not have the necessary resources to use a more extensive approach (such as SOMA) but still need to ensure that the system is aligned to their business.

Nevertheless, the models in SODM do not have all the information required by testers of the SOA being built. The functional requirements of the system are described using textual labels in the service process and service composition models, and non-functional requirements are not explicitly modelled at all.

This section will present several ways in which the models in SODM could be extended to assist testers. Myers [22] lists six different higher-order testing methods: module, integration, function, system, acceptance and installation testing. Among these six, only the three methods that require special treatment in SOA will be considered: integration, function and system testing. Acceptance tests could use the existing traceability relationships between the CIM and PIM models in SODM: rather than extending the models, it is a matter of using the right tools to relate these models.

In order to take advantage of the model-driven approach, these tests should be part of a feedback loop in which test results are used to further improve both the models and the system.

4.4.1 System tests: performance requirements

In the context of SOA, system tests would cover the entire architecture: the whole ecosystem formed by all exposed WS and their consumers. Since functional tests would be better suited to specific parts of the system, these larger tests would be more indicated to non-functional aspects such as scalability or latency. Having a way to detect as early as possible if the desired performance would not be met or could not be possibly achieved with the existing design could be useful. In many cases, these non-functional requirements are part of the Service Level Agreement (SLA) of the exposed Web Services.

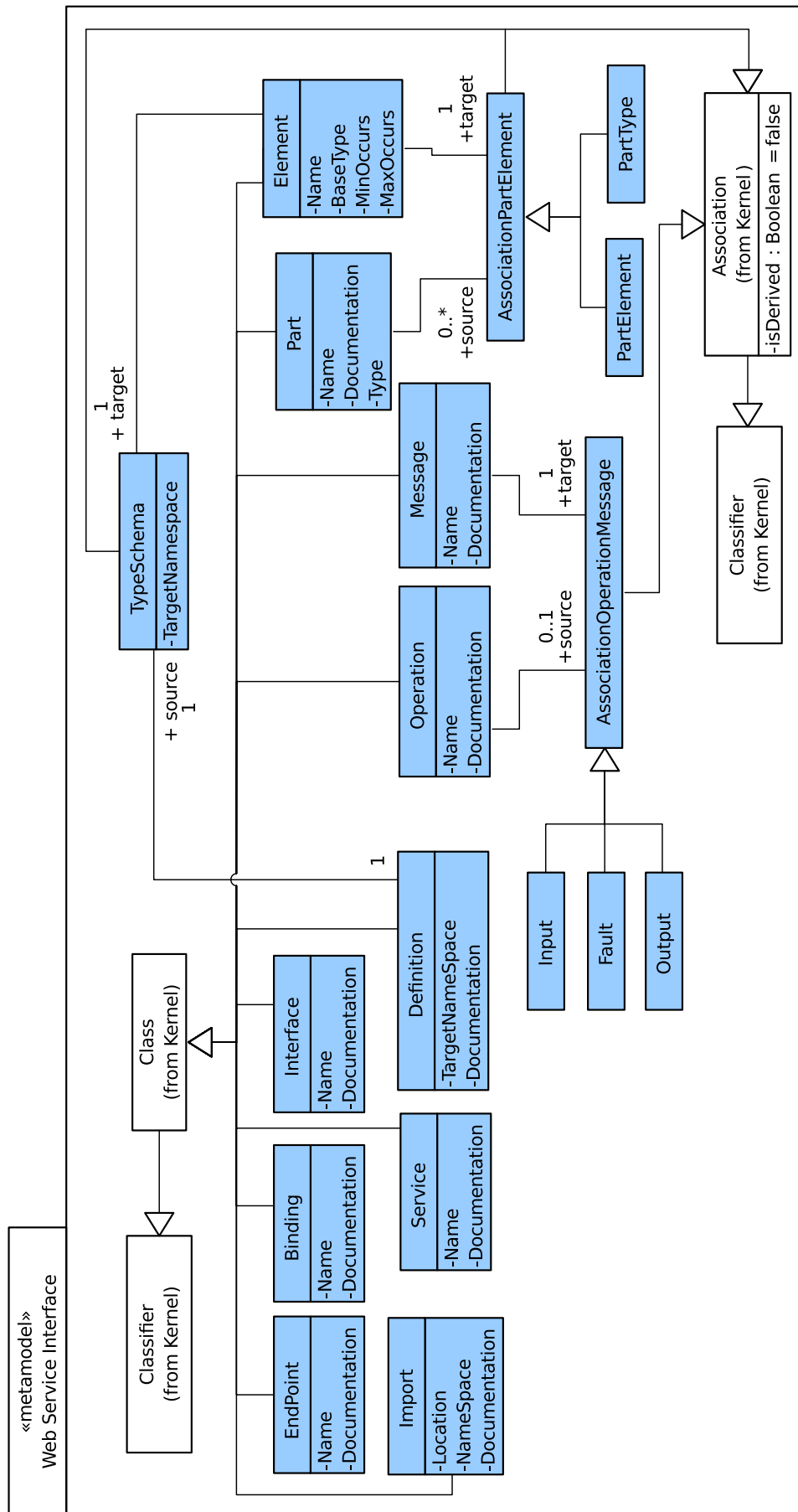


Figure 4.5. SODM WS interface metamodel (translated from [7]). New stereotypes are shaded in blue.

These requirements can be described as part of the PIM service process models, extending them with new annotations containing the interesting parts of the SLAs involved. SLAs could be applied at multiple levels: service processes (probably implemented as Web Service compositions), their service activities (individual Web Services) or the actions that make up the service activities (components used by the WS). Ideally, the SLAs for the lower-level elements should follow from those of the higher-level elements: if the global performance requirements change, these changes should be automatically propagated to the lower-level requirements. It may also be useful to check if the lower-level requirements are consistent with the high-level ones. These transformations and validations could be supported by a model-driven approach.

The resulting performance requirements could be then used to automatically generate a set of test artefacts for existing test performance tools, such as ContiPerf [5] (for Java classes), soapUI [13] (for SOAP-based Web Services) or The Grinder [1] (a general-purpose load testing framework).

Figure 4.6 illustrates how these requirements could be expressed in a model of “Handle Order”, a service process which handles an order from a customer. Several new stereotypes have been added, indicating the minimal number of concurrent requests per second that must be processed and the time limit for each of them. Conditional control flows are also annotated with an estimation of the probability with which they are activated. Using the information provided by the modeller (the filled nodes) the tools may be able to derive new information (the transparent nodes). For instance, since the probability that the order is accepted is $p = 0.8$, each of the two concurrent execution branches started if the order is accepted will need to handle at least $5p = 4$ requests per second.

4.4.2 Function and integration tests: service contracts

In the context of SODM, function tests will attempt to find defects in the way that a particular business service is implemented. The business service is implemented as a service process formed by multiple service activities, and some of these service activities will be eventually implemented as reusable WS.

In order to derive tests that try to find defects in the entire service process (at the integration testing level) or a specific service activity (at the function testing level), functional contracts for the service process and each of the service activities would be highly valuable. These contracts could be used to automatically derive test cases to be run on the final system, or to check specific properties.

The contracts must relate the inputs and outputs of the service process and service activities with the initial and final state of the system. Since SOA is heavily focused on having high-quality reusable WS, the service activities with the `<<WS>>` stereotype are especially important. These service activities appear at the PSM level, in the extended service composition models.

There are many notations that the models could use to represent the contracts. Since SODM uses UML models, the most obvious choice would be the textual preconditions and postconditions from the Object Constraint Language (OCL) of the Object Management Group (OMG) Object Management Group [26].

Figure 4.7 shows how these would look like for the service composition model derived from the service process model in Figure 4.6. The standard `<<precondition>>` and `<<postcondition>>` stereotypes present assertions that must hold before and after the

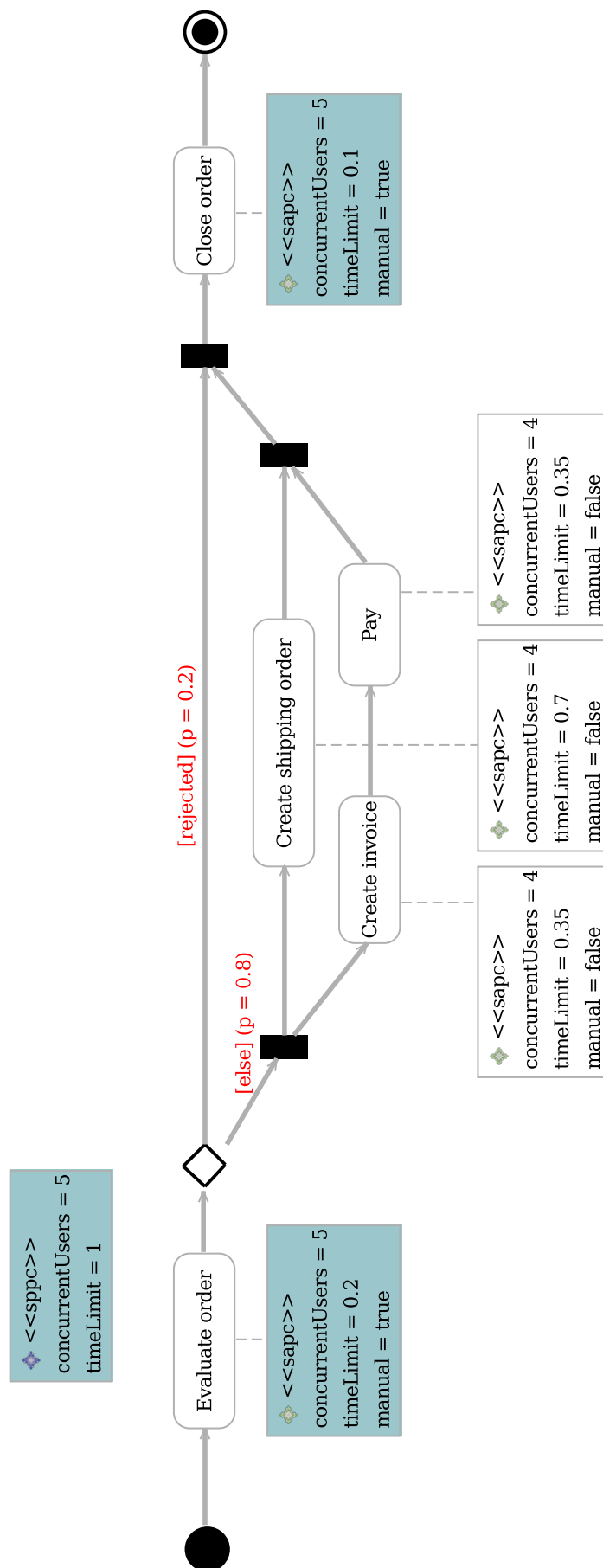


Figure 4.6. Model for the “Handle Order” service process, with non-functional testing extensions

Listing 4.1 Service contract for “Create Invoice” in JML

```

/*@
  @ requires o.accepted == true
  @       && o.open == true
  @       && o.articles.length > 0
  @       && !(\exists Invoice i;
  @           Invoice.all.contains(i)
  @           ⇒ i.order.id == o.id);
  @
  @ ensures \fresh(\result)
  @       && \result.entries.length == o.articles.length
  @       && (\forall int j;
  @           j ≥ 0 && j < \result.entries.length;
  @           \result.entries[j].price
  @           == o.articles[j].currentPrice)
  @       && (\sum int j;
  @           j ≥ 0 && j < \result.entries.length;
  @           \result.entries[j].price) == \result.total)
  @*/
public Invoice createInvoice(Order o) {
    // ...
}

```

execution of the service activity “Create invoice”, respectively. As it is marked with the $\ll WS \gg$, it will be eventually implemented as a Web Service. The preconditions on “Create invoice” require that the order should be initially open, accepted, not empty and without a matching invoice. After its execution, the postconditions require that a new invoice with the proper entries and total sum (including tax) has been created.

OCL is not the only textual language available for describing these preconditions and postconditions. There are other options, such as the Java Modelling Language (JML) by Burdy et al. [6] or the Spec# language by Barnett et al. [4], which are closer to familiar languages such as Java or C#, respectively. In fact, the relative success of JML (with support for some tools such as the static checker ESC/Java James and Chalin [18] or the Daikon dynamic invariant generator [12]) in the research community motivated the creation of WSCoL, an extension of JML for WS with support for temporal logic by Baresi et al. [3].

Listing 4.1 shows how the preconditions and postconditions in Figure 4.7 would be written in JML. Due to syntactic reasons, the service activity has been represented as a Java method.

JML and Spec# could be especially useful for integration tests. Both could decorate the Java or C# code that implements a particular service activity (as a WS) with a contract. The contracts could be read by special-purpose compilers or aspect-oriented programming to weave programs that checked that the contract was met every time, and notify providers and users about violations. The contracts could also be used as an additional *test oracles*, checking if the obtained results are valid. Contracts could be used to test the individual

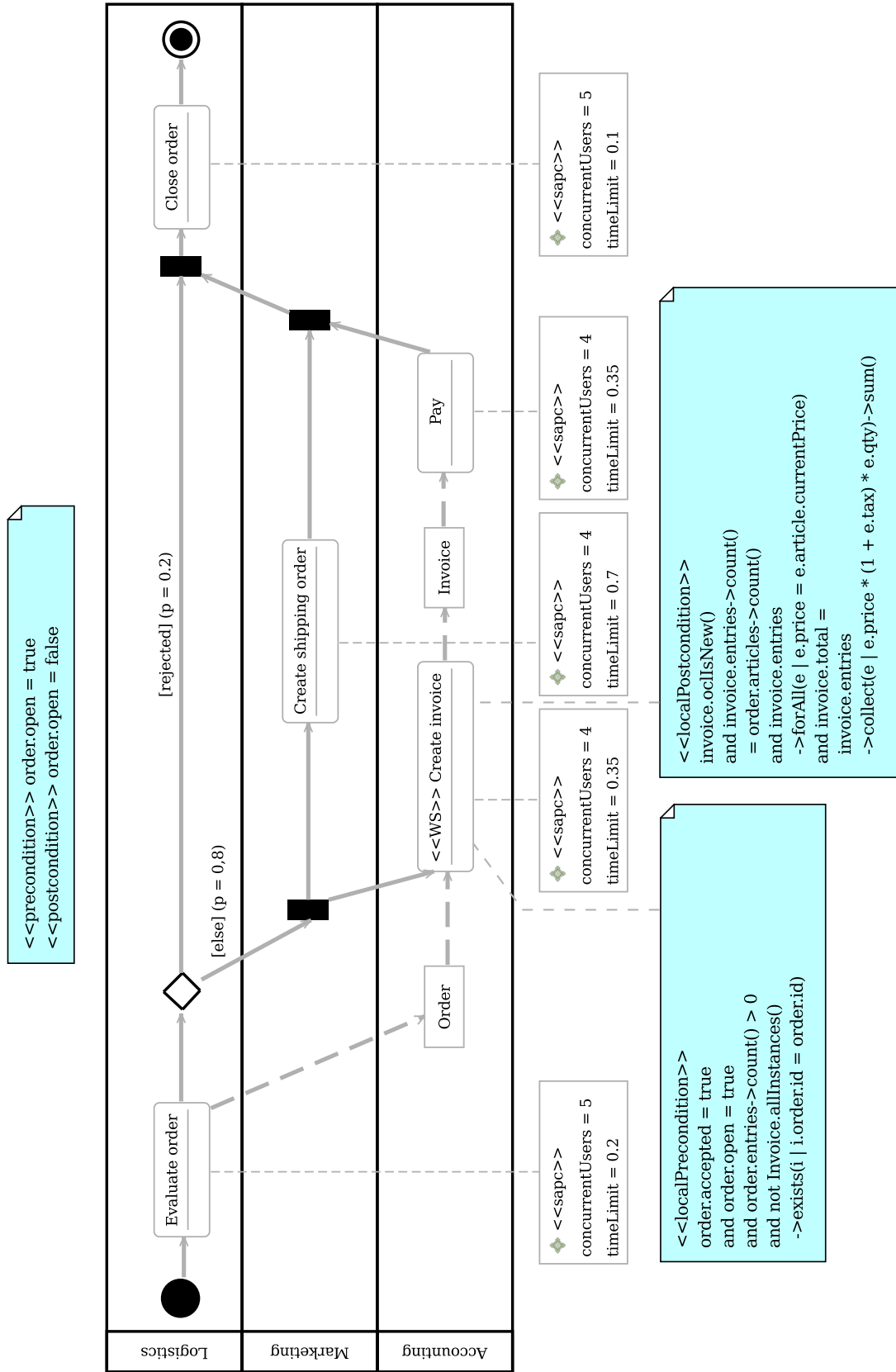


Figure 4.7. Service composition model for “Handle Order” with functional and non-functional testing extensions

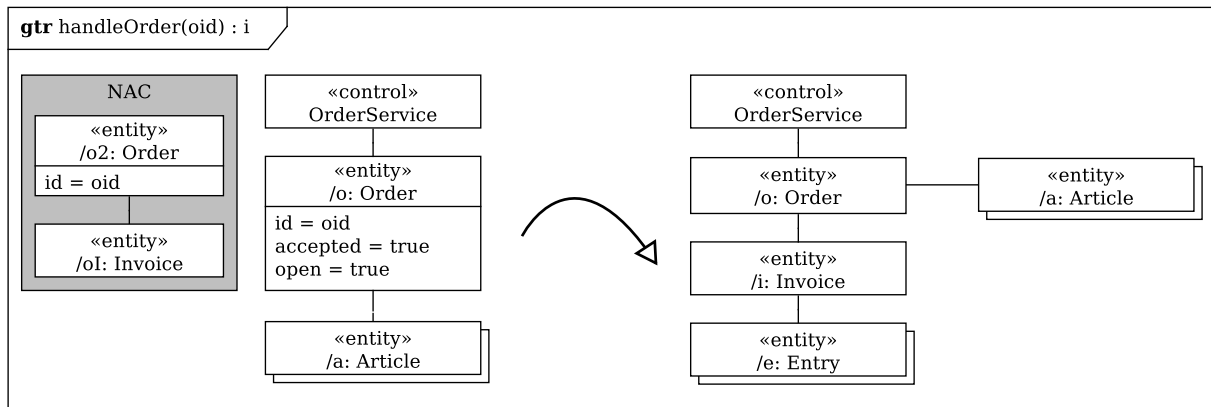


Figure 4.8. Example of a visual contract based on graph transformations for “Handle Order”

actions that make up the service activity as well.

It is important to note that there are also other options beyond traditional assertion-based contracts. Sinha and Paradkar [29] proposed using semantic descriptions based on a specific ontology, and Lohmann et al. [19, 20, 21] presented a graphical notation based on graph transformations. The notation by Lohmann is easier to use, but it does not have the same expressive power as OCL or JML.

The visual contracts proposed by Lohmann represent each operation as a set of graph transformation rules, in which the graphs in the left half (the preconditions) are transformed into the graphs in the right half (the postconditions). Negative conditions are put inside grey boxes with the `«NAC»` stereotype. Objects that only appear on the left half are deleted, while objects that only appear on the right half are newly created. Universal and existential quantifiers can be represented using multiobjects, which are collections of objects of the same type and are drawn as a stack of rectangles.

To illustrate these visual contracts and compare them with OCL and JML, the restrictions in Figures 4.7 and Listing 4.1 have been adapted accordingly in Figure 4.8. The notation can represent a considerable part of the required logic, but it lacks several important features. It could not model the constraints involving the total sum included in the invoice, or that the prices in the invoice were the currently listed prices.

4.5 Conclusion

Creating a service-oriented architecture is a complex task that can affect the entire organisation. To help tackle this complexity, several methodologies have been proposed at multiple levels of abstraction. In this chapter, the strengths and weakness of several of these methodologies have been compared: a summary of the results is shown in Table 4.1 in page 4.2.

SODM, BPSOM and Hoyer were found to be lightweight methodologies that could be applied with a manageable cost for a small-to-medium company for roughly the same tasks. SOMA was discarded as it was a proprietary methodology from IBM and would likely be applicable only to larger organisations. The recommendations from Erl and Papazoglou were too abstract, and the methodology by Stojanović did not explicitly define a set of

models to be created.

Another important conclusion is that none of the existing model-driven technologies explicitly included testing aspects in their models. SOMA did consider an explicit test stage in the development process, but the models themselves did not include any information that was specifically for testing. The other methodologies did not cover the testing stage at all.

It can be argued that a model-driven methodology for creating a SOA should consider testing aspects as well. For this reason, Section 4.3 defined a set of selection criteria to decide which of these methodologies should be extended. After applying these criteria, SODM was selected and analysed in further depth to see in which ways its models could be extended with testing aspects.

Several tentative extensions were proposed for testing functional and non-functional requirements in Section 4.4. The desired behaviour of the modelled service processes, service activities and actions could be annotated with services contracts written in languages such as OCL [26], JML [6] or (more appropriately) WSCoL [3]. The desired performance of the SOA could be also described at multiple levels, so the performance requirements for an entire service process could trickle down into all the service activities. The next chapters will further define and evaluate some of these proposed extensions.

References

- [1] P. Aston and C. Fitzgerald. The Grinder, a Java Load Testing Framework, 2012. URL <http://grinder.sourceforge.net/>. Last checked: November 6th, 2013. 4.27
- [2] C. Atkinson, P. Bostan, D. Brenner, G. Falcone, M. Gutheil, O. Hummel, M. Juhasz, and D. Stoll. Modeling components and Component-Based systems in Kobra. In *The Common Component Modeling Example*, volume 5153 of *Lecture Notes in Computer Science*, pages 54–84. Springer Berlin, Heidelberg, Alemania, 2008. ISBN 978-3-540-85288-9. doi: 10.1007/978-3-540-85289-6_4. 4.3
- [3] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. A timed extension of WSCoL. In *Proceedings of the IEEE International Conference on Web Services, 2007 (ICWS 2007)*, pages 663–670, 2007. doi: 10.1109/ICWS.2007.25. 4.29, 4.32
- [4] M. Barnett, K. R. M. Leino, and W. Schulte. The Spec# programming system: an overview. In *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices*, pages 49–69. Springer Berlin, 2005. 4.29
- [5] V. Bergmann. ContiPerf 2, September 2011. URL <http://databene.org/contiperf.html>. Last checked: November 6th, 2013. 4.27
- [6] L. Burdy, Y. Cheon, and D. R. Cok. An overview of JML tools and applications. *International Journal on Software Tools for Technology Transfer (STTT)*, 7(3):212–232, June 2005. 4.29, 4.32
- [7] M. V. de Castro. *Aproximación MDA para el desarrollo orientado a servicios de sistemas de información web: del modelo de negocio al modelo de composición de*

-
- servicios web*. PhD thesis, Universidad Rey Juan Carlos, March 2007. 4.1, 4.4, 4.19, 4.24, 4.26
- [8] V. De Castro, E. Marcos, and J. M. Vara. Applying CIM-to-PIM model transformations for the service-oriented development of information systems. *Information and Software Technology*, 53(1):87–105, 2011. ISSN 0950-5849. doi: 10.1016/j.infsof.2010.09.002. 4.4, 4.5, 4.6, 4.19
 - [9] A. Delgado, F. Ruiz, I. de Guzmán, and M. Piattini. Business process service oriented methodology (BPSOM) with service generation in SoaML. In H. Mouratidis and C. Rolland, editors, *Advanced Information Systems Engineering*, volume 6741 of *Lecture Notes in Computer Science*, pages 672–680. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-21639-8. 4.1, 4.5, 4.6
 - [10] D. F. D’Souza and A. C. Wills. *Objects, Components, and Frameworks with UML: The Catalysis(SM) Approach*. Addison-Wesley Professional, October 1998. ISBN 0201310120. 4.3
 - [11] T. Erl. *SOA: Principles of Service Design*. Prentice Hall, Indiana, EEUU, 2008. ISBN 0132344823. 4.1
 - [12] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao. The daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1-3):35–45, 2007. 4.29
 - [13] eviware.com. soapUI home page, 2012. URL <http://www.soapui.org/>. 4.27
 - [14] S. Ghosh, A. Arsanjani, and A. Allam. SOMA: a method for developing service-oriented solutions. *IBM Systems Journal*, 47(3):377–396, 2008. 4.1, 4.4
 - [15] J. Gordijn and H. Akkermans. Value-based requirements engineering: exploring innovative e-commerce ideas. *Requirements Engineering*, 8(2):114–134, July 2003. doi: 10.1007/s00766-003-0169-x. 4.5, 4.17
 - [16] J. Gordijn and H. Akkermans. e3value™ toolset, August 2006. URL <http://www.e3value.com/tools/>. 4.17
 - [17] P. Hoyer, M. Gebhart, I. Pansa, A. Dikanski, and S. Abeck. Service-oriented integration using a model-driven approach. *International Journal On Advances in Software*, 3(1):304–317, September 2010. ISSN 1942-2628. 4.1, 4.7, 4.8
 - [18] P. R. James and P. Chalin. Faster and more complete extended static checking for the java modeling language. *Journal of Automated Reasoning*, 44(1-2):145–174, February 2010. ISSN 0168-7433. doi: 10.1007/s10817-009-9134-9. 4.29
 - [19] M. Lohmann, S. Sauer, and G. Engels. Executable visual contracts. In *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 63–70, 2005. doi: 10.1109/VLHCC.2005.35. 4.31
-

- [20] M. Lohmann, G. Engels, and S. Sauer. Model-driven monitoring: generating assertions from visual contracts. In *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering, 2006 (ASE '06)*, pages 355–356, 2006. ISBN 1527-1366. doi: 10.1109/ASE.2006.52. 4.31
- [21] M. Lohmann, L. Mariani, and R. Heckel. A model-driven approach to discovery, testing and monitoring of web services. In *Test and Analysis of Web Services*, pages 173–204. Springer Berlin, 2007. ISBN 978-3-540-72911-2. doi: 10.1007/978-3-540-72912-9_7. 4.31
- [22] G. J. Myers. *The Art of Software Testing*. John Wiley & Sons, 2 edition, 2004. ISBN 0471469122. 4.25
- [23] D. K. Nguyen, W.-J. van den Heuvel, M. Papazoglou, V. de Castro, and E. Marcos. GAMBUSE: a gap analysis methodology for engineering SOA-Based applications. In *Conceptual Modeling: Foundations and Applications*, volume 5600 of *Lecture Notes in Computer Science*, pages 293–318. Springer Berlin Heidelberg, 2009. 4.1
- [24] Object Management Group. Query/View/Transformation (QVT) 1.1, January 2011. URL <http://www.omg.org/spec/QVT/1.1/>. Last checked: November 6th, 2013. 4.5
- [25] Object Management Group. Unified Modeling Language (UML) 2.4.1, August 2011. URL <http://www.omg.org/spec/UML/2.4.1/>. Last checked: November 6th, 2013. 4.10
- [26] Object Management Group. Object Constraint Language Specification (OCL) 2.3.1, January 2012. URL <http://www.omg.org/spec/OCL/2.3.1/>. Last checked: November 6th, 2013. 4.27, 4.32
- [27] Object Management Group. Service oriented architecture modeling language (SoaML) 1.0.1, May 2012. URL <http://www.omg.org/spec/SoaML/1.0.1/>. 4.5
- [28] M. P. Papazoglou and W.-J. V. D. Heuvel. Service-oriented design and development methodology. *Int. J. Web Eng. Technol.*, 2(4):412–442, 2006. 4.1
- [29] A. Sinha and A. Paradkar. Model-based functional conformance testing of web services operating on persistent data. In *Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications*, pages 17–22, Portland, Maine, 2006. ACM. ISBN 1-59593-458-8. doi: 10.1145/1145718.1145721. 4.31
- [30] Z. Stojanović. *A Method for Component-Based and Service-Oriented Software Systems Engineering*. PhD thesis, Delft University of Technology, 2005. 4.1, 4.3
- [31] J. Vara Mesa, E. Marcos, and M. V. de Castro. Obteniendo modelos de sistemas de información a partir de modelos de negocios de alto nivel: un enfoque dirigido por modelos. In *Actas de las IV Jornadas Científico-Técnicas en Servicios Web y SOA*, pages 15–28, Sevilla, España, October 2008. 4.4
- [32] M. Weidlich, G. Decker, A. Großkopf, and M. Weske. BPEL to BPMN: the myth of a straight-forward mapping. In *On the Move to Meaningful Internet Systems: OTM 2008*, volume 5331 of *Lecture Notes in Computer Science*, pages 265–282,

- Monterrey, Mexico, November 2008. Springer Berlin. ISBN 978-3-540-88870-3. doi: 10.1007/978-3-540-88871-0_19. 4.9
- [33] World Wide Web Consortium. XML Schema Part 0: Primer (Second Edition). Technical report, November 2004. URL <http://www.w3.org/TR/xmlschema-0/>. Last checked: November 6th, 2013. 4.24
- [34] World Wide Web Consortium. Web Services Description Language (WSDL) Version 2.0 Part 0: Primer, June 2007. URL <http://www.w3.org/TR/wsdl20-primer>. Last checked: November 6th, 2013. 4.24

5

SODM+T: extension of SODM for performance testing

In the previous chapter, the existing model-driven methodologies for developing service-oriented architectures were reviewed. The existing methodologies assisted users in modelling the business environment, deciding how the business processes should be supported by software and extracting a list of Web Services that exposed the functionality that was fit for reuse. However, they did not use the models to test the services themselves. For this reason, it was decided to extend the models and transformations within SODM to assist in the testing in the derived models.

In this chapter, the first version of SODM+T will be presented. This version of SODM+T extends SODM by adding local and global partial performance constraints to some of the models in SODM to help users define the performance requirements of the system. These constraints are extended to the entire system by several algorithms that automatically transform the model. After defining the algorithms, their theoretical and empirical performance is analysed on several classes of models.

5.1 Introduction

In order to reduce development costs and increase code reuse throughout the organisation, service-oriented architectures encourage developers to reuse services from third parties or other parts of the organisation whenever possible. This usually results in new services called *service compositions*. Such a combination of internal and external services implies that the overall Quality of Service (QoS) is not fully controlled by the original developer: it also depends on the quality of service of the integrated services.

Many different strategies have been proposed and combined in order to deal with this sort of QoS dependency in software systems [1, 3, 12]. Some of the most common approaches are to sign Service Level Agreements (SLAs) with the external providers and to monitor the services for performance degradations. However, defining the parameters of the SLA or the conditions for a performance degradation can be difficult: asking too much may be expensive for the service consumer, and asking too little may alienate its customers. Existing approaches have focused on computing the expected global QoS from the local QoS of each service, and using it to select services among several candidates so that the global requirements are met. Normally, this information is usually obtained by constantly monitoring the performance of the application.

However, there are many cases in which the expected QoS is not known for all the services involved, or those values are not trusted. The service may not be constantly monitored, or the data may not be publicly disclosed by the service provider. Alternatively, the service may not be even implemented yet. The only choice in that case is to make an educated guess. However, if the guess is wrong, all estimations will have to be manually revised, which can be not just tedious, but error-prone.

In this context, high-level models of the service compositions could be used to infer the QoS that should be required from the integrated services to meet the global QoS requirements. Section 4.4.1 suggested adding performance annotations to the SODM service composition models introduced in Section 4.3.3.4. This chapter will refine these extensions and present an algorithm to compute how many requests per second should be handled by the services, and two algorithms to infer their time limits. The first time limit inference algorithm is a simpler version that is used as a performance baseline and test oracle for the second algorithm, which has been heavily optimised.

The rest of this chapter is structured as follows: Section 5.2 shows how several of the SODM metamodels were extended to accommodate these new performance requirements. Section 5.3 presents the extended model editors. Section 5.4 defines two naïve algorithms for inferring the throughput and response times that should be imposed on each service. Due to the high cost of the naïve time limit inference algorithm, Section 5.4.5.2 presents a naïve graph-based algorithm that is optimised in Section 5.4.5.3 so it does not need to visit all candidate paths in the underlying graph. Section 5.5 discusses the practical limitations of the algorithms and evaluates their theoretical and empirical performance on several classes of models. Finally, Section 5.6 summarises the results obtained and outlines several ways in which the presented approach could be improved.

5.2 Extended metamodels

SODM used a subset of the UML activity diagrams for their service process and service composition models. This section will revise these models with the annotations needed to describe the global performance requirements of the service processes and service compositions and derive the local performance requirements of the actions within them. Some of the actions may include additional local information with partial knowledge about their expected performance.

5.2.1 Extended service process metamodel

Figure 5.1 shows a UML class diagram of the extended SODM+T service process metamodel. A service process model consists of a single *ServiceProcess* instance (highlighted in red), which contains a set of *ActivityNodes* connected by several *ActivityEdges*.

The available *ActivityNodes* are almost the same as in the original SODM metamodel, except for these changes:

- UML *StructuredActivityNodes* were added, so users can nest nodes inside each other and describe performance constraints hierarchically.
- UML *MergeNodes* were added: SODM did not include them, but in order to conform to UML they were added back.

The model editors can validate if a *MergeNodes* or a *JoinNodes* are reuniting the right kind of branches by computing the common branching point of all their parents. In graph theory, finding this branching point is known as finding the Least Common Ancestor (LCA) of all the parents of the *JoinNode*. Section 5.3.1 will discuss the implementation of a solution for finding the LCA.

The other extensions added by SODM+T for modelling the performance requirements of the process are also shown in green:

- *ControlFlows* that have a *DecisionNode* as their source¹ can now specify a condition for their traversal (e.g. “ $x > 0$ ” or “order accepted”) and an estimation of the probability that the condition holds true (between 0 and 1, both exclusive).

¹This is enforced by the model validation rules in Section 5.3.2.

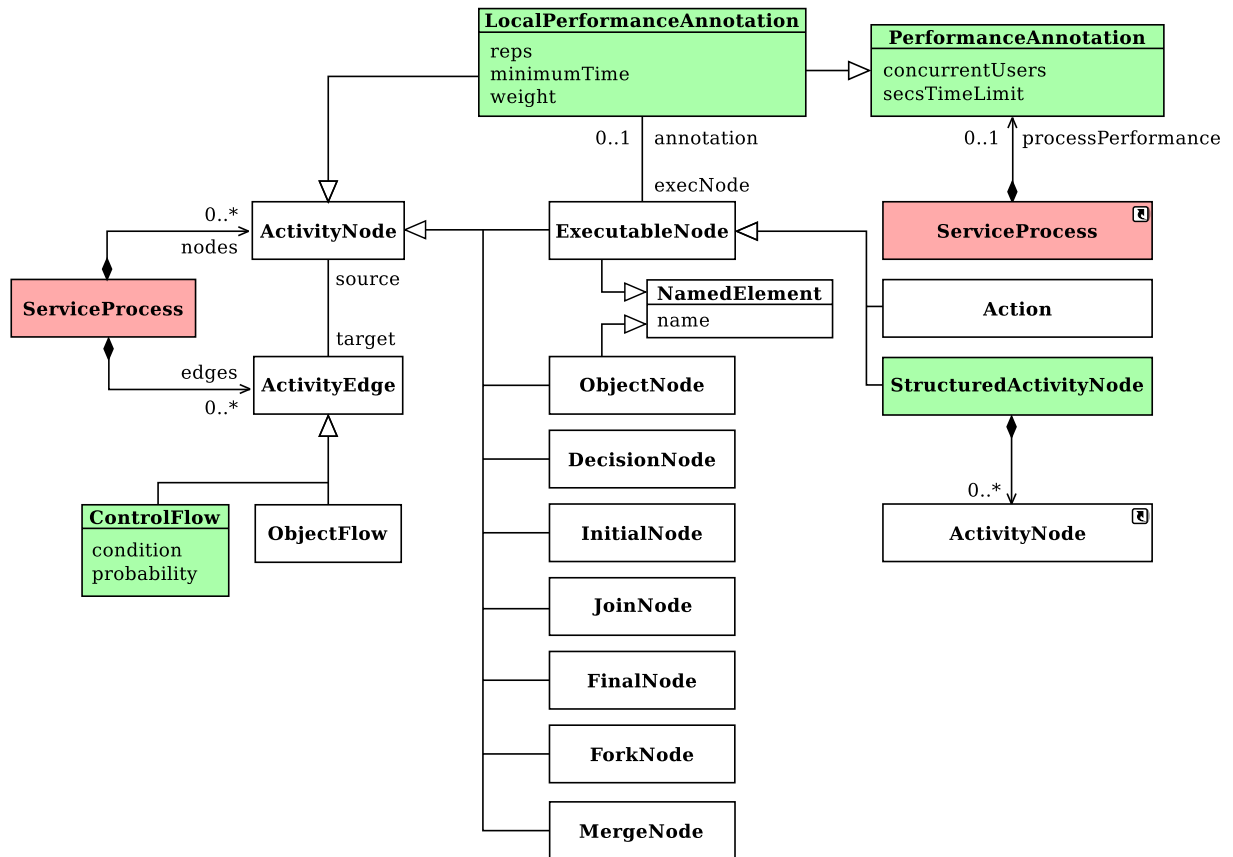


Figure 5.1. UML class diagram with the extended service process metamodel. Some classes have been repeated in the diagram for the sake of clarity: these are decorated with arrow emblems.

- *PerformanceAnnotations* are global performance annotations. They have two attributes: *concurrentUsers*, the number of users per second that will be invoking the activity, and *secsTimeLimit*, the time limit in seconds for each reply. Users are expected to set both attributes manually, according to their performance requirements for the workflow.
- *LocalPerformanceAnnotations* annotate individual *ExecutableNodes*. In this case, *concurrentUsers* and *secsTimeLimit* will be automatically computed from the *weight* and *minimumTime* attributes.

minimumTime is the minimum time in seconds that should be allotted to the node, while *weight* is a real number which serves as a relative estimation of the computational cost of the node. Having a weight of 3 roughly means that the node may take three times as much time as another with weight 1, after considering all the minimum times. Finally, *reps* is the expected number of iterations the node will go through: it should be 1 or more. Section 5.4 defines these concepts formally.

By default, *minimumTime*, *weight* and *reps* are set to 0, 1 and 1, respectively. These values model the simplest case: a node with no known minimum time, being run only once and with a unit weight.

Using this extended metamodel, the algorithms can iteratively “drill down” on increasingly local constraints from the global performance annotation. First, the global performance constraint will be used to derive the local performance annotations of the outermost nodes. Next, these local performance annotations will be used as the global performance constraint for the contents of the each of the *StructuredActivityNodes*. This process will continue recursively until only atomic *Actions* remain.

5.2.2 Extended service composition metamodel

Figure 5.2 shows the extended service composition metamodel in SODM+T. It is almost the same as the extended metamodel for service processes. The only change in the metamodel is that *ActivityNodes* may be split over several *ActivityPartitions*. Each of these partitions represents one of the business stakeholders, as in the original version from SODM.

According to the metamodel, service compositions can have *ActivityNodes* that do not belong to any partition. For this reason, new validation rules have been added that only allow *ObjectNodes* to be placed outside a partition. These are the nodes that will explicitly model the information to be exchanged between the stakeholders. These additional model validation constraints are listed in Section 5.3.2.

5.3 Extended model editors

After defining the extended SODM+T metamodels in Section 5.2, the next step is creating the editors required to create conforming models. The editors have been created using the model-driven process implemented by the Eclipse Graphical Modelling Framework (GMF) (§3.3.3.6, [4]).

The GMF models are first generated by EuGENia from annotated versions of the metamodels and then customised automatically using transformations written in the

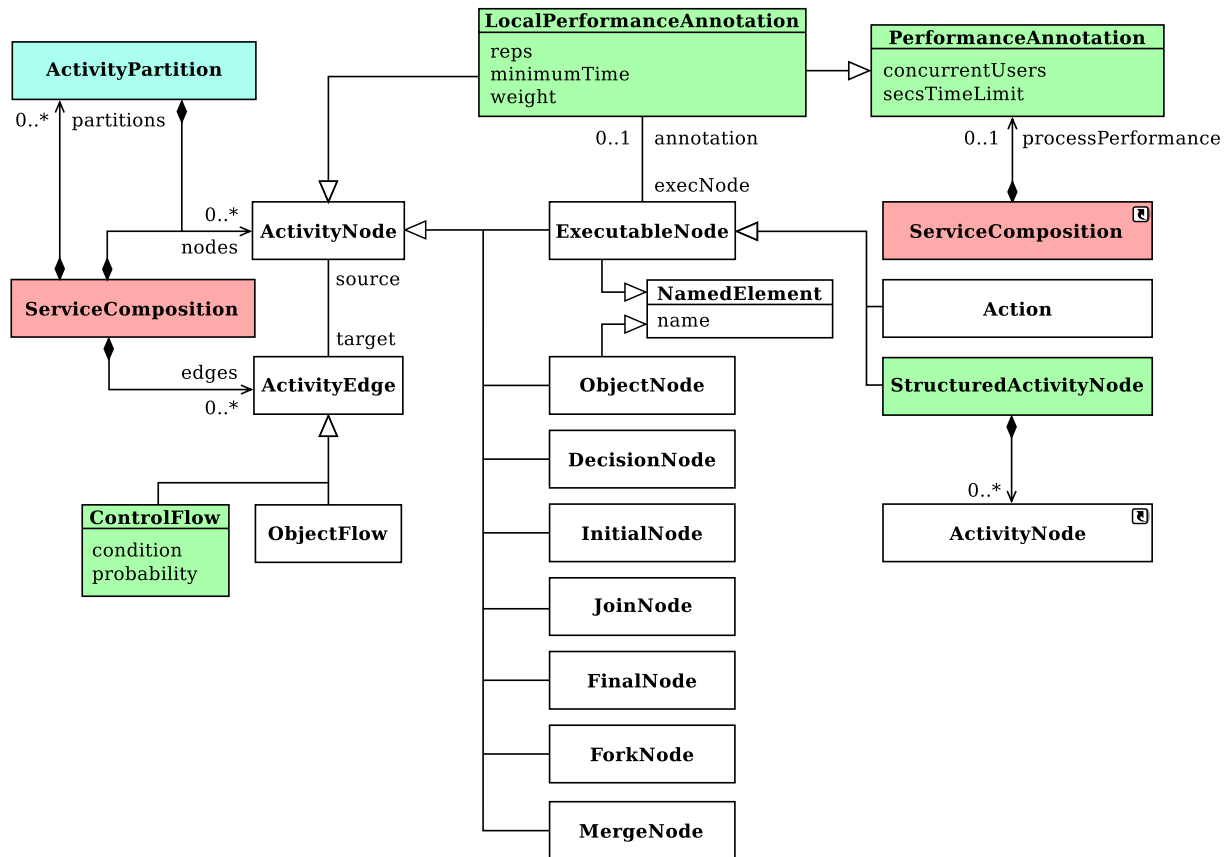


Figure 5.2. UML class diagram with the extended service composition metamodels. Some classes have been repeated in the diagram for the sake of clarity: these are decorated with arrow emblems.

Listing 5.1 Definition of the *InitialNode* class with EuGENia annotations, using the Emfatic textual notation for EMF-based metamodels.

```
1 @gmf.node(figure="svg", svg.uri="platform:/plugin/.../initial.svg",
2         size="30,30", label.placement="none", resizable="false",
3         margin="0", tool.name="Initial Node")
4 class InitialNode extends ActivityNode {}
```

Epsilon Object Language (EOL). EuGENia defines its own set of annotations, automating the most common tasks when creating graphical model editors based on GMF.

Listing 5.1 shows an example of how EuGENia is used in combination with the Emfatic textual notation for writing EMF-based metamodels [5]. The `gmf.node` annotation indicates that:

- instances of the class will be represented as nodes in the diagram (by using `gmf.node`),
- will be drawn using a certain Scalable Vector Graphics (SVG) image specified in `svg_uri` (since `figure` was set to “svg”),
- with a fixed size of 30 by 30 pixels (`size`),
- with no label and no margin (`resizable` and `margin`),
- and the tool for creating these nodes will be labelled “Initial Node” in the palette (`tool.name`).

Some of the additional customisations done beyond these annotations were contributed back as new annotations for EuGENia, such as support for vector-based images, custom polygons, OCL link restrictions and custom label parsers, among others². Other contributions to EuGENia included multiple bug fixes and automation support for the Eclipse integrated development framework.

The following subsections will focus on the ways in which the editors support the creation of valid and consistent models. Starting with a description of some of the supporting algorithms behind the editors, the semantic rules that are enforced by the editors will be listed, followed by a discussion of the usefulness of model migration as a concise notation to transform models conforming to similar metamodels.

5.3.1 Computing least common ancestors

The model editors implement several supporting algorithms for model validation using the Epsilon Object Language (EOL). In addition to being much more concise than Java for manipulating models, EOL scripts can be reused throughout all the Epsilon-based languages, including the Epsilon Validation Language (EVL), the Flock model migration language or the Epsilon Wizard Language (EWL) for in-place model transformations.

Most of these supporting algorithms are quite simple and well known, such as collecting all nodes reachable from a node in breadth-first order or detecting cycles by using a

²See the Eclipse bug reports #400569 or #359921.

depth-first traversal of the graph. This section will describe a less known problem that appears when needing to validate if a *JoinNode* or *MergeNode* are reuniting branches that split off at a *ForkNode* or *DecisionNode*, respectively.

Finding this branching point can be understood as finding the Least Common Ancestor (LCA) of the parent nodes of the *JoinNode*. There are several definitions of LCA in the literature, but in this work the Set of Least Common Ancestors (SLCA) of two nodes will be defined as the leaves of the subgraph that contains all their common ancestors.

From graph theory, it is said that a node A is an ancestor of B (likewise, that B is a descendant of A) when there is a path (a sequence of nodes connected by directed edges) from A to B . If the path goes through only one edge (has length 1), it is said that A is a parent of B , and that B is a child of A . A node is defined as a *leaf* when it has no descendants besides itself.

Several algorithms have been published to compute the LCAs of pairs of nodes in trees and Directed Acyclic Graphs (DAGs) [2]. Many of the existing algorithms preprocess the original tree or graph to speed up later queries into a simple table lookup. These algorithms have been designed for large and dense graphs with many edges, such as those commonly used in fields such as computational biology. However, service process and service composition models are usually not that large and not that dense, and it will not be necessary to compute the LCA of all the nodes, but rather only those of the parents of the *JoinNodes*. In fact, the results obtained by Bender et al. suggest that the naive algorithm (based on the above definition of SLCA) has acceptable performance for modest and less dense graphs such as ours. If necessary, it is always possible to switch to a more advanced algorithm in the future without changing the rest of the model editors.

The algorithms receive two nodes A and B from a graph G and returns $\text{lca}(A, B)$, another node in the graph, by following these steps:

1. The *depths* of the nodes A and B , $d(A)$ and $d(B)$ respectively, are computed and used to derive $M = \max\{d(A), d(B)\}$.

Here, the depth of a node is defined recursively: nodes with no incoming edges have depth 0, and the rest have the maximum depth of all their parents plus one. This definition requires graphs not to have any cycles in order to guarantee that the computation does not loop endlessly. In other words, the model must represent a DAG.

The depth of each node is only computed once per model: after it has been computed, it is cached by the EOL execution engine.

2. The ancestors of A with depth less than or equal with M are stored in set A_A .
3. A_B is computed from the ancestors of B in the same way.
4. $A = A_A \cap A_B$ is computed and one of the elements of the set $\text{SLCA}(A, B) = \{e | d(e) = \max_{a \in A} \{d(a)\}\}$ is returned. This set contains the elements common to A_A and A_B that have the maximum depth. As shown later, the model validation rules ensure that $|\text{SLCA}(a, b)| \leq 1$ for all pairs (a, b) of parents of a *JoinNode*.

Figure 5.3 illustrates how the algorithm works. In order to find the LCA of D and F , their ancestors are first computed and then divided into common ancestors (circles) and the rest (pentagons for D and rhombi for F). Among the common ancestors, the one with the maximum depth is picked: B , highlighted in bold.

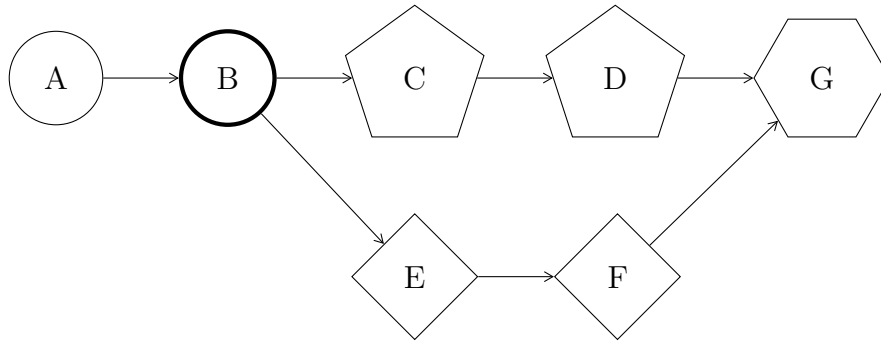


Figure 5.3. Finding the least common ancestor of D and F

5.3.2 Model validation

In order to produce the expected results and simplify their design to a reasonable degree, the performance inference algorithms must receive service process and service composition models that meet certain constraints beyond simply conforming to the metamodels in Section 5.2.

These constraints are largely divided into a common set for both metamodels, and two specific sets of constraints for service process and service composition models. These restrictions have been implemented using the Epsilon Validation Language (EVL) [9] and tested with the EUnit framework on Appendix B. By properly modifying the model editor generated by GMF, the restrictions are automatically checked upon saving the model. Some of these restrictions have been implemented in OCL as well, in order to forbid users from performing certain actions (e.g. creating a self-loop).

The common set is shown in Table 5.1, and the additional sets that are specified to service processes and service compositions are listed in Tables 5.2 and 5.3. It is said that a certain EVL restriction is satisfied by an instance of a certain type both when it is applicable to that instance (its *guard*, expressed as “If X...”, is satisfied) and when the restriction itself is upheld (the *check* is satisfied). Some EVL restrictions may depend on others: it may not make sense to bother the user with additional errors, or doing the check may produce an error or run into an endless loop.

Some numeric fields are checked by testing if their values belong to a small range, rather than simply testing them for equality. This is due to the approximate representation of a number that the standard IEEE-754 [8] used by Java provides.

One seemingly important restriction imposed upon the models is that they must have no cycles: the algorithms need the graphs to have a finite number of paths. Nevertheless, bounded iteration can be represented through the *reps* attribute in *LocalPerformanceAnnotation*.

Table 5.1. Common restrictions for service process and service composition models

Element	Restriction	Fixes
<i>ActivityEdge</i>	Both the source and the target ends must be set.	None.
	The source and the target ends must not be the same.	None.

Continues on next page

Continued from previous page

Element	Restriction	Fixes
<i>ActivityNode</i>	If it is neither an <i>InitialNode</i> nor a <i>PerformanceAnnotation</i> nor an <i>ObjectNode</i> , it must have at least one incoming edge.	None.
	If it is not a <i>JoinNode</i> nor a <i>MergeNode</i> and no cycles exist in the graph, it must have at most one incoming <i>ControlFlow</i> .	Reassign the multiple <i>ControlFlows</i> to a new <i>JoinNode</i> .
	If it is neither a <i>FinalNode</i> nor a <i>PerformanceAnnotation</i> nor an <i>ObjectNode</i> , it must have at least one incoming <i>ControlFlow</i> .	None.
	If it is neither a <i>ForkNode</i> nor a <i>DecisionNode</i> and there are no cycles in the graph, it must have at most one outgoing <i>ControlFlow</i> .	Reassign the multiple outgoing <i>ControlFlows</i> to a new <i>ForkNode</i> or a new <i>DecisionNode</i> (decided by the user).
<i>ControlFlow</i>	If the edge has a condition, the source must be a <i>DecisionNode</i> .	Remove the condition.
	If the source of the edge is a <i>DecisionNode</i> , it must have a condition.	Set “dummy” as condition to tell the user that it should be properly set.
	If the edge has a condition according to the above rules, its probability must be greater than 0 and less than 1.	None.
	If the edge does not have a condition according to the above rules, its probability must be exactly 1.	Set probability to 1.
	If the source is set, it must not be an <i>ObjectNode</i>	None.
	If the target is set, it must not be an <i>ObjectNode</i>	None.
<i>FlowEdge</i>	Both the source and the target of the edge should be attached to an <i>ActivityNode</i> .	None.

Continues on next page

Continued from previous page

Element	Restriction	Fixes
<i>FlowEdge</i>	The source and the target of the edge must not be the same.	Remove the edge.
	If both ends are set and the source or the target is a <i>StructuredActivityNode</i> , the source and the target should belong to the same container (whether it is the model or a <i>StructuredActivityNode</i>). In other words: edges must not cross over the boundary of a container.	None.
<i>ObjectFlow</i>	If both source and target are set, exactly one of the source and the target must be an <i>ObjectNode</i> , and the other one has to be an <i>ExecutableNode</i> .	None.
<i>DecisionNode</i>	If the restrictions from <i>ActivityNode</i> on incoming and outgoing edges are met, the node must have more than one outgoing <i>ControlFlow</i> .	Replace the <i>DecisionNode</i> and its only incoming and outgoing <i>ControlFlows</i> with a single edge from the source of the incoming edge to the target of the outgoing edge.
	If the probabilities of all the outgoing edges are valid, their total sum must be 1 ± 10^{-3} .	None.
<i>FinalNode</i>	There must be no outgoing edges.	Remove all outgoing edges.
<i>ForkNode</i>	If the restrictions from <i>ActivityNode</i> on incoming and outgoing edges are met, the node must have more than one outgoing <i>ControlFlow</i> .	Replace the <i>ForkNode</i> and its only incoming and outgoing <i>ControlFlows</i> with a single edge from the source of the incoming edge to the target of the outgoing edge.
<i>InitialNode</i>	If there are no cycles in the graph, all reachable paths must end at a <i>FinalNode</i> .	None.
	It must not have any incoming edges.	Remove all incoming edges.

Continues on next page

Continued from previous page

Element	Restriction	Fixes
	If the restrictions on the number of outgoing edges are met, none of the outgoing edges should point to a <i>FinalNode</i> .	Add an <i>Action</i> between this node and the <i>FinalNode</i> in question.
	No cycles may be reachable from this node.	Remove one of the edges that takes part in the first cycle found.
<i>JoinNode</i>	If the restrictions from <i>ActivityNode</i> on incoming and outgoing edges are met, the node must have more than one incoming <i>ControlFlow</i> .	
	Replace the <i>JoinNode</i> and its only incoming and outgoing <i>ControlFlows</i> with a single edge from the source of the incoming edge to the target of the outgoing edge.	
	Pairwise, all parent nodes must have the same LCA (§5.3.1), and it must be a <i>ForkNode</i> .	None.
<i>Local-Performance-Annotation</i>	The annotation must be linked to some <i>ExecutableNode</i> .	None.
	The minimum time must be greater than or equal to 0.	None.
	The weight must be greater than or equal to 0.	None.
	The number of repetitions must be greater than or equal to 1.	None.
	It must not have any incoming <i>ActivityEdges</i> .	Remove all offending edges.
	It must not have any outgoing <i>ActivityEdges</i> .	Remove all offending edges.

Continues on next page

Table 5.2. Additional constraints for service process models

Element	Restriction	Fixes
<i>ServiceProcess</i>	The composition must have a global <i>PerformanceAnnotation</i> .	Create a global annotation with the default values (weight 1 and minimum time limit 0).
	There must be at least one <i>FinalNode</i> .	None.
	There should be exactly one <i>InitialNode</i> .	None.

Continued from previous page

Element	Restriction	Fixes
<i>MergeNode</i>	If the restrictions from <i>ActivityNode</i> on incoming and outgoing edges are met, the node must have more than one incoming <i>ControlFlow</i> .	Replace the <i>JoinNode</i> and its only incoming and outgoing <i>ControlFlows</i> with a single edge from the source of the incoming edge to the target of the outgoing edge.
	Pairwise, all parent nodes must have the same LCA (§5.3.1), and it must be a <i>DecisionNode</i> .	None.
<i>NamedElement</i>	The name of the element must be defined.	None.
<i>ObjectNode</i>	It must have at least one incoming <i>ObjectFlow</i> .	None.
	It must have at least one outgoing <i>ObjectFlow</i> .	None.
	It must have no incoming and no outgoing <i>ControlFlows</i> .	Remove all offending edges.
<i>Performance-Annotation</i>	The number of concurrent users must greater than 0.	None.
	The time limit must be greater than 0.	None.
<i>Structured-ActivityNode</i>	It must have at most one <i>InitialNode</i> .	None.

Table 5.3. Additional constraints for service composition models

Element	Restriction	Fixes
<i>Service-Composition</i>	The composition must have a global <i>PerformanceAnnotation</i> .	Create a global annotation with the default values (weight 1 and minimum time limit 0).
	There must be at least one <i>FinalNode</i> .	None.
	There must be at least one <i>ActivityPartition</i> .	None.
	The only <i>ActivityNodes</i> outside the <i>ActivityPartitions</i> should be <i>ObjectNodes</i> .	Remove all nodes outside the partitions.
	There should be exactly one <i>InitialNode</i> across all partitions.	None.
<i>ActivityPartition</i>	Every partition should have a name.	None.

5.3.3 Migration of service processes to service compositions

The editor can also automatically transform service process models into service composition models. As it can be seen in Figures 5.1 and 5.2, their metamodels are quite similar to each other. A traditional model-to-model transformation would need to specify many rules that would essentially only copy elements from the service process model to the service composition model.

In this context, a transformation for a specialised *model migration* tool such as Epsilon Flock [11] can be much more concise and easier to maintain. Flock can be told to *retype* the elements of a certain type of the service process metamodel to elements of the service composition metamodel, implicitly copying all their information. If necessary, more advanced migration rules for specific types can be described imperatively.

Table 5.4 summarises the model migration rules. The migration can be invoked from the Eclipse Integrated Development Environment (IDE). In comparison with a previous version that was based on a traditional model-to-model transformation language, the Flock-based transformation has over 74% less lines.

5.4 Performance inference algorithms

In the previous sections, the extended service process and service composition metamodels were introduced, and several interesting aspects of the editors created to edit conforming models were introduced. This section will present several algorithms that infer local performance requirements from a mix of global and local annotations, considering required throughput and response time limits. The first version of the time limit inference will be used as test oracle and performance baseline for the following versions. All algorithms can be

SP type	SC type	Strategy
Action	Action	Retype.
ControlFlow	ControlFlow	Retype.
DecisionNode	DecisionNode	Retype.
FinalNode	FinalNode	Retype.
ForkNode	ForkNode	Retype.
InitialNode	InitialNode	Retype.
JoinNode	JoinNode	Retype.
MergeNode	MergeNode	Retype.
LocalPerformanceAnnotation	LocalPerformanceAnnotation	Retype.
ObjectFlow	ObjectFlow	Retype.
ObjectNode	ObjectNode	Retype.
PerformanceAnnotation	PerformanceAnnotation	Retype.
ServiceProcess	ServiceComposition	Retype, then move all nodes into a new <i>ActivityPartition</i> with name “Main”.
StructuredActivityNode	StructuredActivityNode	Retype.

Table 5.4. Migration strategy from service process (SP) models to service composition (SC) models. The “retype” strategy uses the default Flock migration strategy, copying all the information from the original element to the migrated element.

invoked on demand by modellers from the graphical model editors.

5.4.1 Input and output values

The input variables of the inference algorithms are the workload and time limit specified in the global *PerformanceAnnotation* and the weights, minimum times and repetitions in the *LocalPerformanceAnnotations*. The algorithms update the *concurrentUsers* and *secsTimeLimit* attributes of every *LocalPerformanceAnnotation* with the inferred values.

Depending on how they are combined, *minimumTime* and *weight* can model several common scenarios depending on what the modeller knows about the expected time limit of an executable node. Let $m \geq 0$ be the minimum time and $w \geq 0$ be the weight:

- $m = 0, w = 0$: the node costs nothing. This is not used for *ExecutableNodes*. Rather, they are the implicit values for every *ActivityNode* that is not an *ExecutableNode*. This makes them effectively invisible to the inference algorithms, except for the ways in which they branch and join the execution paths. Without loss of generality, it is assumed that the cost of performing decisions, forks or joins is negligible in comparison to that required by the actions themselves.³
- $m > 0, w = 0$: the node has a fixed time limit. This usually means that there is a strict Service Level Agreement (SLA) for the WS or software component represented by the node, which ensures that it will finish exactly within m seconds.

³In any case, decision costs, for example, can be modelled by *ad hoc* actions preceding decision nodes.

- $m = 0, w > 0$: all the time will be allotted automatically. There is no known SLA or any estimates of how long it could take. Instead, the modeller must compare the cost of the node with the rest of nodes in the workflow.
- $m > 0, w > 0$: part of the allotted time is set manually, and the rest is inferred automatically. This can be useful if previous measures that point to a certain minimum time exist, but it is still desired to grant some of the remaining time.

It is important to note that the *LocalPerformanceAnnotations* of a *StructuredActivityNode* can be used as global constraints for the paths directly inside them, after their *minimumTime* and *concurrentUsers* have been computed. This allows the algorithms to descend recursively through the model, starting at the outermost level and proceeding until only atomic actions are left.

5.4.2 Basic definitions

As listed in Section 5.3.2, all algorithms require that the models do not contain cycles, that they only have one *InitialNode* per container, and that every action is reachable from exactly one of them.

Several shared concepts for all algorithms need to be described first:

- $s(e)$ and $g(e)$ are the source and target *ActivityNodes* of *ActivityEdge* e , respectively.
- $i(n)$ and $o(n)$ are the incoming and outgoing edges of node n , respectively.
- $L > 0$ is the global time limit of the model, in seconds.
- The set of all valid minimum time and weight constraints under a global time limit L is $C(L) = \{(m, w) \mid 0 \leq m \leq L \wedge w \geq 0\}$.
- $c(n) = (m(n), w(n)) \in C(L)$ is the *constraint* of the node n , where $m(n)$ is the minimum time limit of n and $w(n)$ is its weight.
- $r(n)$ is the estimated number of times that node n will be run. If n is inside a *StructuredActivityNode*, it is the number of times that n will be run each time its container is run.
- Each path p has also a constraint $c(p) = (m(p), w(p)) \in C(L)$, with $m(p) = \sum_{n \in p} m(n)r(n)$ and $w(p) = \sum_{n \in p} w(n)r(n)$ being the total minimum time and weight of p after considering the repetitions of each node in p .
- If path p is allotted $t(p)$ seconds, $s(p) = t(p) - m(p)$ can be defined as the *slack* of path p : the time available beyond its total minimum time. By definition, $t(p) = L$ if p starts at an *InitialNode*.

Ideally, it should be distributed proportionally to the weight of each node, according to $S_w(p) = s(p)/w(p)$, which is defined as the *slack per unit of weight* of path p . As a special case, if $w(p) = 0$, then $S_w(p) = 0$.

Consistent models should have $s(p) > 0$ whenever $w(p) > 0$, so every path p with a non-zero weight has some slack to distribute.

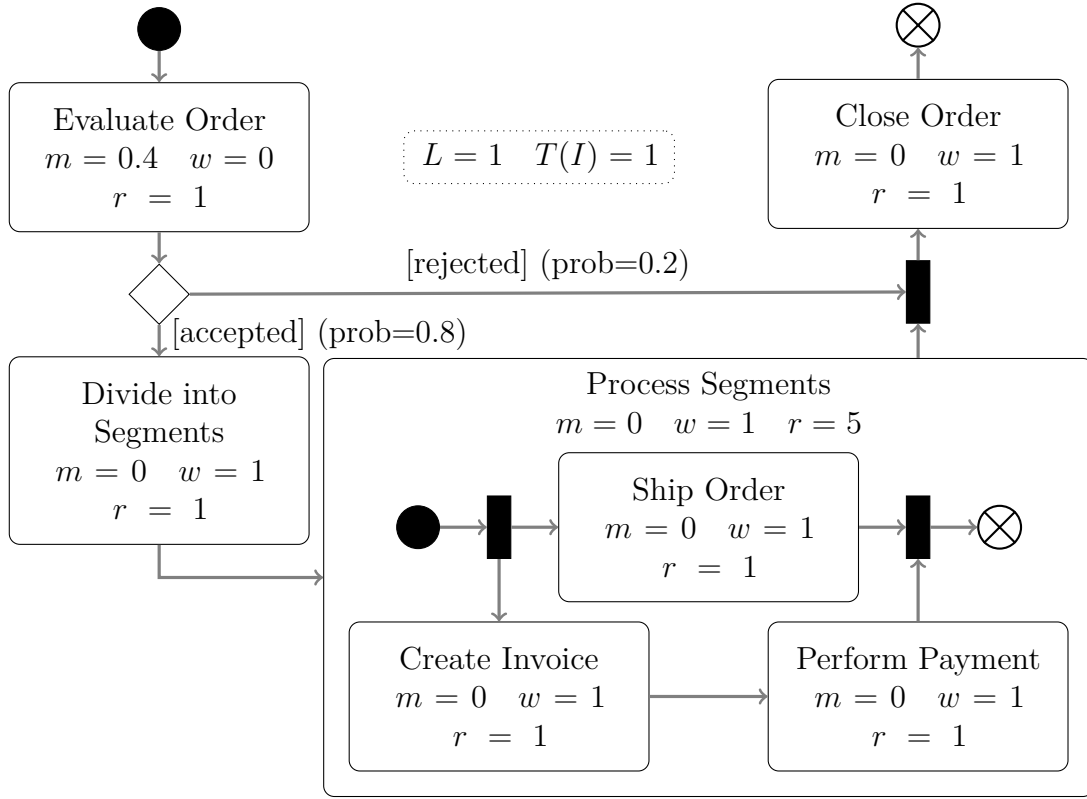


Figure 5.4. Annotated model for the running example. The model includes a global $(L, T(I))$ pair with the global time limit for all execution paths and the number of requests per second (throughput) entering the outermost *InitialNode* I , respectively. *ExecutableNodes* include (m, w, r) tuples with the minimum times, weights and expected iterations of their local performance annotations. Decision branches include estimated traversal probabilities.

- $P_S(n)$ is the set of all paths starting at the node n .
- $d(n)$ is the *depth* of the node n . It is 0 if n does not belong to any *StructuredActivityNodes* and $1 + d(S)$ if n belongs to the *StructuredActivityNode* S .
- Level D of the model is defined as the set of every node n with $d(n) = D$. Level 0 is the *global* level, containing the topmost *InitialNode*. This definition of depth should not be confused with the one used for computing LCAs in Section 5.3.1.

5.4.3 Running example

In order to illustrate the algorithms, the simple example shown in Figure 5.4 will be used. This model represents a Web Service composition that processes an order for a certain item by following these steps:

1. Receive a message from the client with the order.
2. Evaluate the order using a set of internal business rules.
3. If the order is rejected, close the order and notify the client.

4. If the order is accepted:

- a) Divide the order into segments, to ensure that the customer receives each item as soon as possible.
- b) For each segment, create the invoice and perform the payment at the same time the shipment order is sent. This is done by invoking the Web Services provided by the other departments.
- c) Close the order and notify the client.

The model is divided into two levels:

- Level 0 contains the outermost nodes: the *Actions* “Evaluate Order”, “Divide into Segments”, “Process Segments”, “Close Order”, the *DecisionNode* and *MergeNode* and the outermost *InitialNode* and *FinalNode*.
- Level 1 is comprised of the nodes inside “Process Segments”: the *Actions* “Ship Order”, “Create Invoice” and “Perform Payment”, the *ForkNode* and *JoinNode* and the innermost *InitialNode* and *FinalNode*.

The composition has $L = 1$ s as global time limit while handling $T(I) = 1$ request per second. *minimumTime* is set to 0.4 s for “Evaluate Order”, ensuring it receives at least 0.4 s. Since its *weight* is set to 0, it will not get any more time than that: it will be allotted exactly 0.4 s. This combination of values is used to represent the situation in which a strict SLA saying that it should never take longer than 0.4 s had been signed.

All *Actions* had *reps* set to 1. However, the *StructuredActivityNode* “Process Segments” was originally a loop. For that reason, *reps* is set to an estimate of the maximum number of segments expected in a typical order (5).

5.4.4 Throughput inference

After having described the inputs of the algorithms, this section will describe the simplest of the two inference algorithms, which is dedicated to computing locally expected throughput from the globally expected throughput.

The algorithm propagates the global throughput requirement $T(I)$ (where I is the *InitialNode* at the global level) through the graph. T is defined as a function which takes a node or edge and produces its expected throughput. The formula to be applied depends on the type of element passed to it:

- For an *ActivityEdge* e , $T(e) = \mathcal{P}(e)T(s(e))$, where $\mathcal{P}(e)$ is the probability of e being traversed.
- For the *InitialNode* I , $T(I)$ is equal to the *throughput* of the global performance annotation if I is not part of any *StructuredActivityNode*. Otherwise, $T(I)$ is equal to the throughput of the *StructuredActivityNode* it belongs to.
- For a *JoinNode* n , $T(n) = \min_{e \in i(n)} T(e)$, since requests in the least performing branch set the pace.
- For a *MergeNode* n , $T(n) = \sum_{e \in i(n)} T(e)$, as requests from mutually exclusive branches are reunited.

- For any other node n , $T(n) = T(e_1)$, where $e_1 \in i(n)$ is its only incoming edge.

For example, the following process computes $T(\text{Create Invoice})$ for the model shown in Figure 5.4, which needs to handle $T(I) = 1$ request per second:

$$\begin{aligned}
T(\text{Create Invoice}) &= T(\text{incoming edge of Create Invoice}) \\
&= T(\text{ForkNode in Process Segments}) \\
&= T(\text{incoming edge of ForkNode in Process Segments}) \\
&= T(\text{InitialNode in Process Segments}) \\
&= T(\text{Process Segments}) \\
&= T(\text{incoming edge of Process Segments}) \\
&= T(\text{Divide Segments}) \\
&= T(\text{incoming edge of Divide Segments}) \\
&= 0.8 T(\text{Evaluate Order}) \\
&= 0.8 T(\text{incoming edge of Evaluate Order}) \\
&= 0.8 T(I) \\
&= 0.8
\end{aligned}$$

The actual implementation of the algorithm avoids computing the same values several times by annotating each node in level 0 in topological order, then in level 1, and so on until no more nodes are left. The attribute *concurrentUsers* of the local performance annotation of each *ExecutableNode* n is updated to $T(n)$.

5.4.5 Time limit inference

Inferring the time limits of each action inside an activity is considerably more complex than inferring their required throughputs. Section 5.4.5.1 will first describe an algorithm that estimates the time limit of each node by solving one or more Linear Programming (LP) problems. This algorithm will serve as a test oracle and performance baseline for two graph-based algorithms. This section concludes by showing how the algorithms operate on the running example from Figure 5.4.

5.4.5.1 Linear programming-based algorithm

The simplest way to describe the desired time limits is by expressing them as an optimisation problem. In particular, their computation will be formulated as a linear programming problem, as there are efficient tools for solving them.

For the sake of simplicity, it can be assumed for now that the model has only one level, i.e. there are no *StructuredActivityNodes* and there is only one *InitialNode*, I . Let N be the set of all *ExecutableNodes*, $P = P_S(I)$ be the set of paths starting from I and $P_C(n) = \{q \mid q \in P \wedge n \in q\}$ be the set of all the paths in P that contain n .

The optimisation problem will operate on one variable $S_w(n)$ for each $n \in N$, trying to maximise the amount of time that is used over all paths (resulting in more lenient time limits whenever possible):

$$\text{maximise } \sum_{p \in P} \sum_{n \in p} (S_w(n)w(n) + m(n)) r(n).$$

Here, $S_w(n)$ is defined as the *slack per unit of weight* of n . It is the additional time beyond $m(n)$ that is assigned per unit of weight to n . The main idea is that $S_w(n)$ should be higher for nodes with higher weights, whenever possible. The resulting time limit $l(n)$ of a node n will be then $S_w(n)w(n) + m(n)$.

The problem is then subject to these two validity constraints:

- R1. $S_w(n) \geq 0$, for all $n \in N$. Assigned times must not be negative.
- R2. $\sum_{n \in p} (S_w(n)w(n) + m(n)) r(n) \leq L$, for all $p \in P$. The assigned times must meet the global time limit on all paths.

However, these are not enough. With only these constraints, assigning all time to a single node in a path and giving no time to the rest would be still acceptable. The next constraints will try to ensure that times are as evenly distributed as possible according to the weight of each task.

The concept of an even distribution is that if an *ExecutableNode* n only belonged to path p , an optimal $S_w(n)$ would be equal to $S_w(p)$. This can be extended to multiple nodes and paths as:

- R3. $S_w(n) \geq \min\{S_w(p) \mid p \in P_C(n), w(p) > 0\}$, for all $n \in N$. The assigned slack per unit of weight must be greater than or equal to the one available on the strictest path that n belongs to. In other words: we may never impose stricter constraints than those that would be inferred from the strictest path alone.
- R4. $S_w(m) = S_w(n)$, for every pair of $m, n \in N$ so that $P_C(m) = P_C(n)$. If two nodes belong to the same set of paths, time should also be distributed proportionally to their weights.

This formulation is a standard linear programming problem that can be readily solved: the time limit for each *ExecutableNode* n will be $S_w(n)w(n) + m(n)$. Generalising this approach to models with more than one level is simple: the algorithm is first run on level 0, producing time limits for all the level 0 *ExecutableNodes* (including the *StructuredActivityNodes*). Then, it is run on the contents of each *StructuredActivityNode* in level 0 (i.e. level 1) using the previously computed time limits as its “global” requirement. This process continues until all levels of the model have been annotated.

If any of the generated problems are unfeasible, the user has placed inconsistent requirements on the composition (for instance, imposing a time limit of 10 seconds and having an action with 15 seconds of minimum time). Detecting and reporting these situations concisely to the user can be difficult when using linear programming, however.

This formulation is easy to understand and implement. However, the size of the problem to be solved increases exponentially as more and more paths are added to the model. For this reason, this algorithm may not be usable for reasonably large models. The next section will show a graph-based algorithm that alleviates this by culling uninteresting subpaths as soon as possible.

5.4.5.2 Exhaustive graph-based algorithm

The exhaustive graph-based algorithm is a loop that iterates over every path from I , refining the existing results until every *ExecutableNode* is annotated with its time limit $l(n)$. It follows these steps:

1. Compute all the paths in $P_S(I)$. These are all the paths from the *InitialNode* to each of the *FinalNodes*.
2. Sort each path in $P_S(I)$ with $w(p) > 0$ in increasing order of $S_w(p)$, so the strictest paths come first.
3. Visit each path p in $P_S(I)$:
 - a) If $s(p) = 0$, the minimum time limits are taking up all the available time. If $w(p) > 0$, there will be nodes that should receive some additional time and will not be able to. In that case, the user is notified and execution is aborted.
 - b) If $s(p) < 0$, the sum of the minimum time limits in p exceeds the global time limit. The user is notified and execution is aborted.
 - c) Otherwise, $s(p) > 0$. Split the *ExecutableNodes* in p into two disjoint subsets depending on whether $l(n)$ has been set (R , the set of *restricted* nodes) or not (U , the set of *unrestricted* nodes).

Let $l(R) = \sum_{n \in R} l(n)r(n)$ be the total time used by the restricted nodes, $m(U) = \sum_{n \in U} m(n)r(n)$ be the total minimum time used by the unrestricted nodes and $w(U) = \sum_{n \in U} w(n)r(n)$ be the total weight of the unrestricted nodes. For every n in U , set $S_w(n)$ to $(L - l(R) - m(U))/w(U)$ if $w(U) > 0$, distributing the remaining slack evenly over the unrestricted nodes, or to 0 otherwise.

Each node $n \in U$ is assigned the time limit $l(n)$:

$$l(n) = m(n) + S_w(n)w(n) \quad (5.1)$$

This algorithm basically tries all paths, sorting them in such a way that each node will be visited first in the path from which its strictest time limit can be inferred. Once it has a time limit, it will not be changed again: instead, its time limit will be used to calculate the time limits for the nodes in the following (and less strict) paths.

Once more, generalising this algorithm to models with multiple levels only requires running the algorithm first for the *InitialNode* in level 0, then for all the *InitialNodes* in level 1 (using the inferred values as global time limits), and so on.

The algorithm is simple, but it has the same inherent flaw as the LP-based algorithm: it needs to visit all the paths in $P_S(I)$, whose number grows exponentially as *ForkNodes* and *DecisionNodes* are added to the graph. This means that the algorithm is impractical except for simple graphs.

5.4.5.3 Incremental graph-based algorithm

The previous formulations suffered from exponentially growing problem sizes, since they needed to check all paths from the *InitialNode*. This section defines an optimised graph-based algorithm that builds the set of paths under study incrementally, reducing this explosive growth.

Again, the description will be simplified by assuming that the model has only one level. Multiple levels are handled in the same way as in the previous algorithms.

The algorithm can be described as a recursive function taking a node n and the time assigned to it, $t(n)$. Initially, $n = I$ and $t(n) = L$, the global time limit. The algorithm follows these steps:

1. Select two paths from $P_S(n)$:
 - Let $p_{ms}(n)$ be the path with the minimum $S_w(p)$ when $t(n)$ seconds are available. In case of a tie, pick the path with maximum $w(p)$.
 - Let $p_{Mm}(n)$ be the path with the maximum $m(p)$.
2. If $s(p_{Mm}(n)) < 0$, the minimum time limits cannot be satisfied: notify the user and abort the execution.
3. If $s(p_{ms}(n)) = 0$ and $w(p_{ms}(n)) > 0$, there is no slack in a path with a non-zero weight: notify the user and abort the execution.
4. Set the time limit of n , $l(n)$, to $m(n) + S_w(p_{ms}(n))w(n)$. The remaining time will be $T_R = t(n) - l(n) r(n)$ seconds. Mark n as visited.
5. Sort each edge $e \in o(n)$ in increasing order of $S_w(p_{ms}(g(e)))$ with $r(g(e)) = T_R$. This ensures that the algorithm continues through the subpath starting at n that has the minimum slack per unit of weight when T_R seconds are available.
6. Visit each edge in $o(n)$:
 - a) If the target of e has been visited before, check if the time which was sent to it, T'_R , is strictly less than T_R , the time which would have been sent through e .
 In that case, try to reuse the surplus $T_R - T'_R$ seconds on the source of e and its ancestors, and send T'_R seconds through e . Go back in the graph from the source of e , collecting nodes with non-zero weights into C until a node with more than one incoming or outgoing edge is found. Increase the time limit of each collected node by $(T_R - T'_R)w(n)/w(C)$, where $w(C) = \sum_{n \in C} w(n)r(n)$.
 - b) If the target of e has not been visited before, invoke this algorithm recursively, setting n to the target of e and $r(n) = T_R$.

It is important to note that the graph-based algorithm uses several optimisations to improve its performance. First of all, a path p is not represented by its sequence of nodes, but by its constraint $c(p) = (m(p), w(p))$, saving much memory.

Second, to select $p_{Mm}(n)$ at each node it is necessary to know the maximum $m(p)$ for each path $p \in P_S(n)$, which can be noted as $m(p_{Mm}(n))$ or simply $M_m(n)$. It can be computed in advance with the following equation:

$$M_m(n) = m(p_{Mm}(n)) = m(n)r(n) + \max\{M_m(g(e)) \mid e \in o(n)\} \quad (5.2)$$

Since (5.2) is recursive, it can be evaluated incrementally, starting from the *FinalNodes* (for which $M_m(n) = 0$) and going back up to the *InitialNode* in reverse topological order.

Third, selecting $p_{ms}(n)$ at each node requires knowing the strictest path starting from it. This cannot be computed in advance, as it depends on the time received by the node, $t(n)$, which is not known *a priori*. Instead, redundant paths from $P_S(n)$ will be removed. This reduced set will be called $P'_S(n)$. A path $p_a \in P_S(n)$ is removed when it is said to be *always less or just as strict* than some other path $p_b \in P_S(n)$, independently of the time

received by n or the common ancestors of p_a and p_b . Let $(a, b) = c(p_a)$ and $(c, d) = c(p_b)$. The formal comparison between p_a and p_b is defined as follows:

$$\begin{aligned}
 (a, b) \preceq_{s(L)} (c, d) \equiv & \\
 & \forall t \in [0, L] \forall x \in [0, L] \forall y \geq 0 \\
 & a + x \leq t \wedge c + x \leq t \wedge \\
 & b + y > 0 \wedge d + y > 0 \Rightarrow \\
 & \frac{t - (a + x)}{b + y} \geq \frac{t - (c + x)}{d + y}
 \end{aligned} \tag{5.3}$$

After a lengthy simplification (shown in Appendix A.1), the right side of (5.3) can be replaced by:

$$a \leq c \wedge (b \leq d \vee (b - d)L \leq bc - ad) \tag{5.4}$$

Equations (5.3) and (5.4) would consider all pairs of the form (L, x) to be just as strict. The number of paths to be evaluated could be further reduced while obtaining the same results by considering the (L, x) pair with the highest value of x as the strictest one. This results in the revised and final definition of $\preceq_{s(L)}$:

$$a \leq c \wedge (b \leq d \vee (a < L \wedge (b - d)L \leq bc - ad)) \tag{5.5}$$

It can be proved that this defines a partial order (a reflexive, antisymmetric, and transitive binary relation) on $C(L)$. The proof is shown in Appendix A.2.

Finally, like $M_m(n)$, $P'_S(n)$ can also be computed incrementally by traversing the graph in reverse topological order. Let n_i be a child of n . Let p_a and p_b be two paths in $P_S(n_i)$, so that $c(p_a) \preceq_{s(L)} c(p_b)$. By definition, p_a is less or just as strict as p_b regardless of their common ancestors, so $\langle n \rangle + p_a$ will also be discarded from $P'_S(n)$ over $\langle n \rangle + p_b$. This means that instead of comparing every path in $P_S(n)$ for every node n , $P'_S(n)$ can be built by adding n at the beginning of the paths in $P'_S(n_i)$, for every child n_i of n , and then filtering the redundant paths using $\preceq_{s(L)}$.

Let $\max_{\preceq_{s(L)}} S$ select the paths in S which are not always less strict than any other (the maximal elements according to $\preceq_{s(L)}$). $P'_S(n)$ is defined as:

$$\begin{aligned}
 P'_S(n) = \max_{\preceq_{s(L)}} \{ & (m(n)r(n) + M, w(n)r(n) + W) \mid \\
 & e \in o(n) \wedge (M, W) \in P'_S(g(e)) \}
 \end{aligned} \tag{5.6}$$

Please, notice that $P'_S(f) = P_S(f) = \{(0, 0)\}$ when f is a final node.

5.4.5.4 Examples

The previous sections defined the linear programming-based and graph-based algorithms and described the key optimisations used within the latter. The algorithms will now be applied to the example in Figure 5.4, showing how they arrive at the same results through different means. The global time limit will be set in both cases to $L = 1$ s. Action names will be shortened to their initials when necessary: “Evaluate Order” will be simply “EO”.

Listing 5.2 GMPL model used by the algorithm in Section 5.4.5.1

```

1 set A; # All executable nodes
2 set P; # All paths
3 param G; # Global time limit
4 param m{a in A}, default 0; # Minimum times
5 param w{a in A}, default 1; # Weights
6 param r{a in A}, default 1; # Repetitions
7 param paths{a in A, p in P}, default 0, binary; # Paths (1 if action belongs to path)
8
9 # Total minimum time and weight of each path
10 param mp{p in P} := sum{a in A} paths[a, p] * m[a] * r[a];
11 param tw{p in P} := sum{a in A} paths[a, p] * w[a] * r[a];
12
13 # Minimum slack per unit of weight by task
14 param msuw{a in A} := min{p in P: paths[a, p] == 1 && tw[p] > 0} (G-mp[p])/tw[p];
15
16 # Slack per unit of weight for each action (must be positive)
17 var suw{a in A} ≥ 0;
18
19 maximize usage: sum{a in A, p in P} (paths[a, p] * (suw[a]*w[a] + m[a]) * r[a]);
20
21 subject to glimit {p in P}: sum{a in A} (paths[a, p] * (suw[a] * w[a] + m[a]) * r[a]) ≤ G;
22 subject to minslack {a in A}: suw[a] ≥ msuw[a];
23 subject to samepaths
24   {a in A, b in A: a < b && forall {p in P} paths[a, p] == paths[b, p]}: suw[a] == suw[b];
25
26 solve;

```

Listing 5.3 GMPL data for level 0 of the example in Figure 5.4

```

set A := "Evaluate Order", "Divide into Segments", "Close Order", "Process Segments";
set P := p_1 p_2;
param G := 1.0;
param m := "Evaluate Order" 0.4;
param w := "Evaluate Order" 0.0;
param r := "Process Segments" 5.0;
param paths :=
  [*, p_1] "Evaluate Order" 1 "Close Order" 1
  [*, p_2] "Evaluate Order" 1 "Divide into Segments" 1 "Process Segments" 1 "Close Order" 1;
end;

```

Listing 5.4 GMPL data for level 1 of the example in Figure 5.4. Minimum times, weights and repetitions use the default values of 0, 1 and 1, respectively.

```

set A := "Perform Payment", "Create Invoice", "Ship Order";
set P := p_1 p_2;
param G := 0.08571428571428572;
param paths :=
  [*, p_1] "Create Invoice" 1 "Perform Payment" 1
  [*, p_2] "Ship Order" 1;
end;

```

5.4.5.4.1 Linear programming-based algorithm The linear programming formulation described above can be written in any of the existing mathematical programming languages. In this Thesis GNU MathProg Language (GMPL) was selected, which is part of the GNU Linear Programming Kit (GLPK)⁴. GMPL is a very concise notation for linear programming and is a subset of the A Mathematical Programming Language (AMPL) language [6].

In GMPL, the problem can be split into two sections. The *model* section describes the available parameters, variables, constraints and objective function. The *data* section provides values for some of the parameters. This is useful for reusing the same problem with different data.

The shared model for all problems generated in the algorithm is shown in Listing 5.2. Lines 1 and 2 define the set of actions or nodes (A) and the set of all paths (P) over which the constraints will be defined. The global time limit is modelled as a parameter (G) in line 3. Lines 4–6 model the minimum times, weights and estimated repetitions as additional parameters with default values 0, 1 and 1 respectively. Lines 10 and 11 define derived parameters with the total minimum time and weight of every path. From these, line 14 derives another parameter with the minimum slack per unit of weight for each action in A . Using these parameters, the *suw* positive variables (according to R1) defined in line 17 need to maximise the total available time as defined in line 19, subject to the constraints in lines 21 (R2), 22 (R3) and 23–24 (R4).

The algorithm solved the problem using the data for level 0 in Listing 5.3 first. The resulting time limits were of 0.086 s for “Evaluate Order” (EO), “Divide into Segments” (DS), “Close Order” (CO) and each repetition of “Process Segments” (PS).

Using these results, it produced the data in Listing 5.4 and solved the problem for level 1. The resulting time limits were of 0.043 s for “Perform Payment” (PP) and “Create Invoice” (CI) and 0.086 s for “Ship Order” (SO).

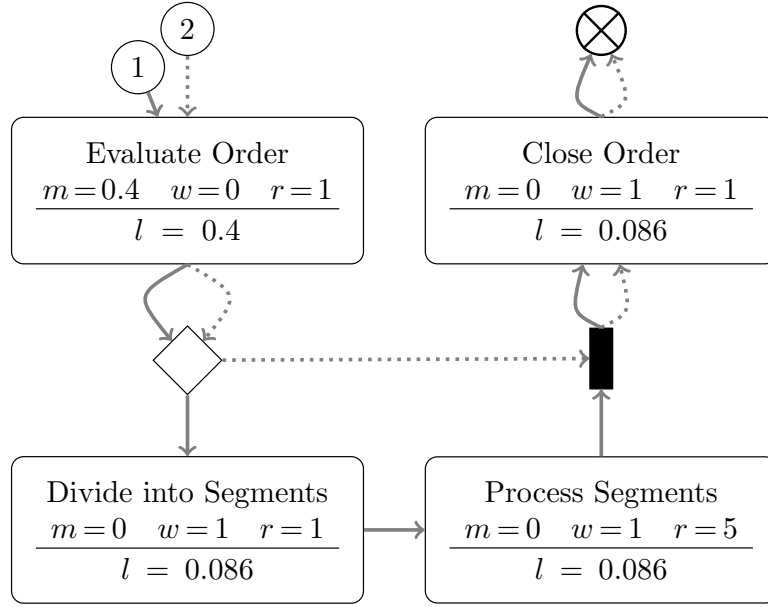
5.4.5.4.2 Exhaustive algorithm The exhaustive algorithm produces the execution traces shown in Figure 5.5. The paths from the *InitialNodes* to the *FinalNodes* have been labelled with numbers, ordering them from most to least restrictive.

On level 0, there are two paths: the strictest one goes through “Divide into Segments” and “Process Segments” and the other does not. The algorithm visits the strictest path p_1 first: all its nodes are unrestricted (as no time limits have been set), so $m(U) = m(p_1) = 0.4$ and $w(U) = w(p_1) = 0 + 1 + 5 + 1 = 7$. Therefore, $S_w(n) = (1 - 0.4)/7 \simeq 0.086$ for every n in p_1 . The second path does not have any unrestricted nodes (all nodes were annotated in the previous path).

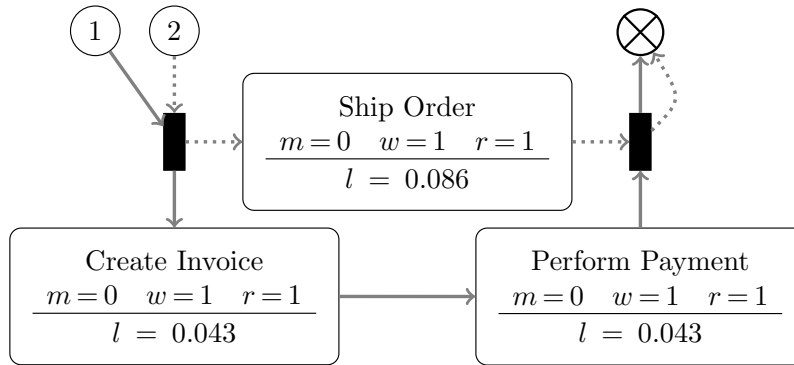
Next, level 1 (with the contents of “Process Segments”) is examined, with $L = 0.086$ as previously computed. The strictest path goes through “Creative Invoice” and “Perform Payment”, with a zero total minimum time and total weight equal to 2 units. Therefore, each action receives half of the available 0.086 s. The next path goes through “Ship Order”, which receives the full 0.086 s.

5.4.5.4.3 Incremental graph-based algorithm The execution trace of the graph-based algorithm over levels 0 and 1 of the running example is shown in Figure 5.6. First, $M_m(n)$ and $P'_s(n)$ are precomputed over level 0:

⁴<http://www.gnu.org/software/glpk>



(a) Level 0



(b) Level 1

Figure 5.5. Execution trace of the exhaustive time limit algorithm on levels 0 and 1 of the running example in Figure 5.4, with $L = 1$ second. Every *Activity* lists its minimum time limit m , its weight w and the computed time limit t and its number of repetitions r . The available execution paths are sorted from least to most restrictive.

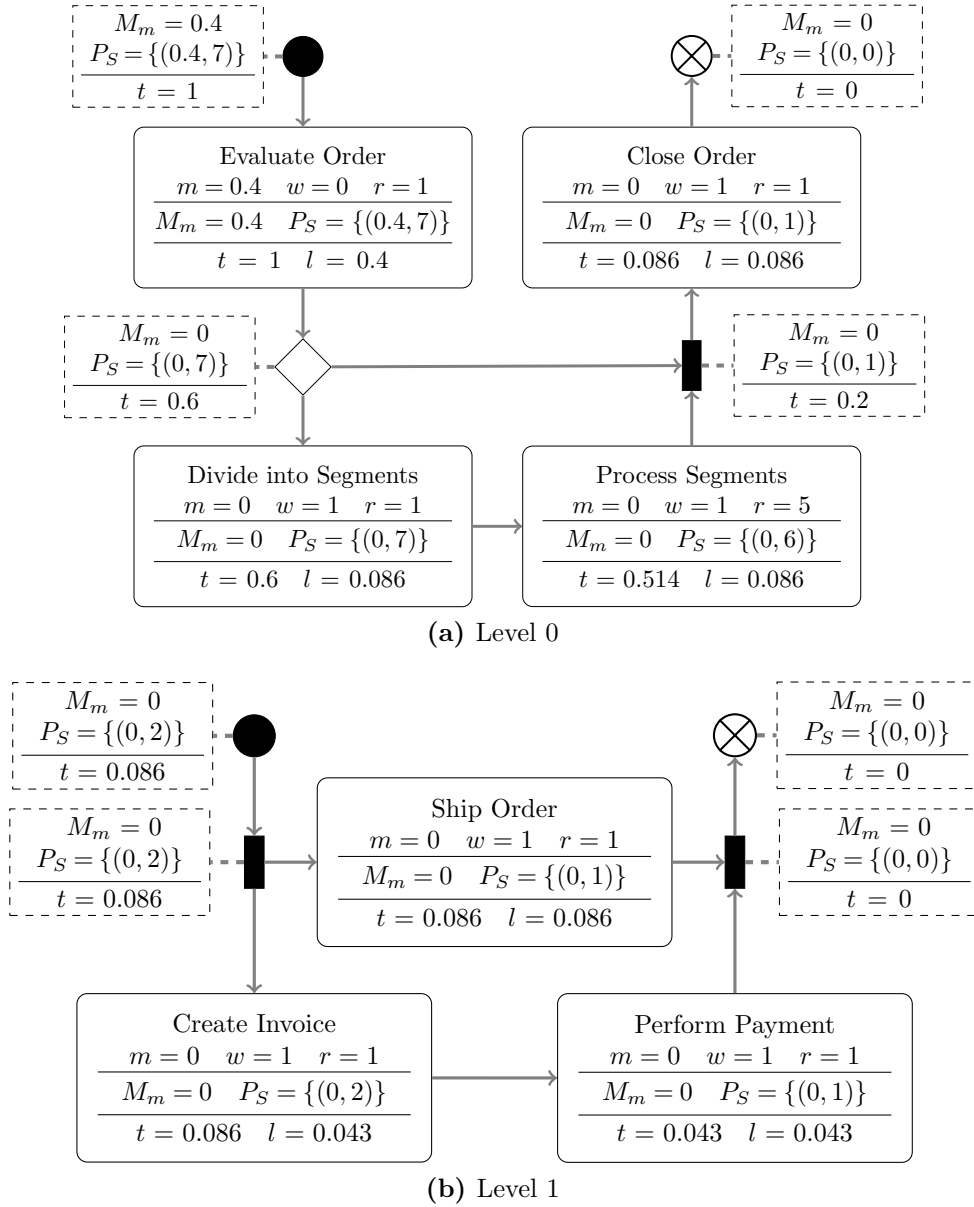


Figure 5.6. Execution traces for the incremental graph-based time limit algorithm on levels 0 and 1 of the running example in Figure 5.4, with $L = 1$ second on level 0. Every *ExecutableNode* was annotated with its minimum time limit m , its weight w and its number of repetitions r . The first pass produced from all subpaths starting at each node n the maximum total minimum time limit $M'_m(n)$ and the maximal constraints $P'_S(n)$. Nodes received t seconds on the second pass, from which each *ExecutableNode* took a part for its time limit l and propagated the rest. Level 1 is comprised of the contents of “Process Segments”, and its global time limit $L = 0.086$ was computed in level 0.

- $M_m(\text{CO}) = 0$, $P'_S(\text{CO}) = \{(0, 1)\}$.
- $M_m(\text{PS}) = 0$, $P'_S(\text{PP}) = \{(0, 6)\}$ (since PS has $r = 5$).
- $M_m(\text{DS}) = 0$, $P'_S(\text{DS}) = \{(0, 7)\}$.
- $M_m(\text{EO}) = 0.4$, $P'_S(\text{EO}) = \{(0.4, 7)\}$.

Using the maximal constraint $(0.4, 7)$, it follows that the slack per unit of weight on the strictest path will be $(1 - 0.4)/7 \simeq 0.086$ seconds. After that, the algorithm sends the available time ($L = 1$ s) into the *InitialNode* and then into EO. EO takes 0.4 s and sends the remaining 0.6 s through the *DecisionNode* to DS, which takes 0.086 s and sends the remaining 0.514 s to PS. PS takes 0.086 s for each of its 5 repetitions and sends the remaining 0.086 s to CO through the *JoinNode*.

The algorithm then continues over the contents of “Process Segments”, using its time limit as the new global time limit. $M_m(n)$ and $P'_S(n)$ are computed in the same way as before. SO takes the full 0.086 s, being the only *ExecutableNode* in its path. CI and PP take half of L each: 0.043 s.

5.5 Evaluation

The previous sections have described the models used by the inference algorithms and the algorithms themselves. This section will study their limitations and evaluate their performance on the above reference implementations.

5.5.1 Limitations

All algorithms require that the graph underlying the model has no cycles. This seems like an important restriction, as loops are common in most processes, but there are ways to model iteration without cycles. In the above models, a loop is represented as a *StructuredActivityNode* with an appropriate estimate of the number of repetitions in its *reps* attribute. Loops can be arbitrarily nested as well in this way, and the time limit inference algorithms scale down the inferred values according to the number of repetitions.

All *ExecutableNodes* need to be annotated with a minimum time limit and a weight. It could be argued that the problem of defining their time limits was simply shifted to defining their local performance annotations. However, these are easier to write and maintain, as they only depend on the individual nodes and the relative values of the weights of the others. In particular, they do not depend on the global time limit. Users satisfied with the default zero time limit and unit weight only need to set the global time limit and the conditional branch probabilities in order to use the inference algorithms.

All *ActivityEdges* starting at a *DecisionNode* need to be annotated with probabilities. The only way to estimate these is through the modeller’s domain knowledge. Alternatively, it may be simply assumed that every decision branch is equally probable, in absence of other information.

In their current form, the algorithms do not take into account the fact that the same task may be used in several workflows, or several times in the same workflow. However, the linear programming-based time limit inference algorithm could be extended to include

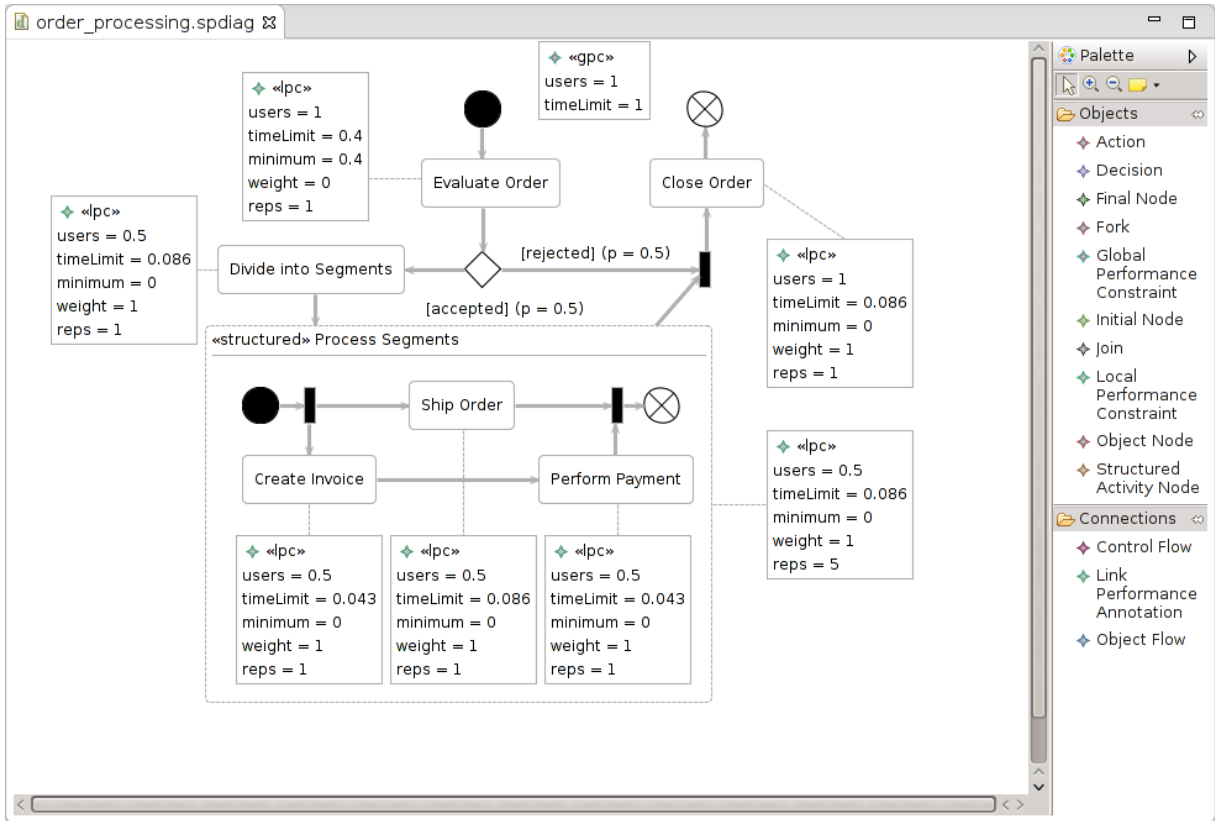


Figure 5.7. Screenshot of the Eclipse-based model editor

some of these restrictions in the generated problems. The graph-based algorithm would not accommodate those additional restrictions as well, but it could take strictest time limit inferred among all its occurrences.

Similarly, the algorithms assume the best-case scenario in which the modelled process is the only one currently being run. This requires that the global time limit and throughput requirements and the minimum times take into account the delays introduced by a system that is also running other processes concurrently.

For the sake of simplicity, this chapter has used a custom set of annotations. Nevertheless, the algorithms can be applied with standard performance annotations, such as those from the MARTE UML [10], as shown in Chapter 6.

5.5.2 Implementation

The above models and algorithms have been implemented as a set of Eclipse plug-ins (see Figure 5.7). The source code is freely available under the Eclipse Public License v1.0 [7] and uses a mix of several task-specific model handling languages from the Epsilon family (§3.3.3).

The models can be created using a graph-oriented graphical editor. To ensure that the inference algorithms can be applied, the tool is able to validate the models automatically, providing error and warning markers and “quick fixes” to assist the user in correcting invalid models. The algorithms can be launched from the contextual menu of the graphical editor.

The throughput inference algorithm and the graph-based time limit inference algorithm have been unit tested using a set of manually designed test cases, using the EUnit framework included in Epsilon (described in Appendix B). In addition, the graph-based time limit inference algorithm has been tested with a large set of automatically generated models, ensuring that its results were within 0.1% of those of the linear programming-based algorithm⁵.

5.5.3 Theoretical performance

The previous sections described the inherent limitations to the algorithms and presented their reference implementations. Before evaluating their empirical performance, this section will infer some upper bounds on their execution costs from their definitions.

This section will also define several representative graph shapes for the models. These shapes will be used throughout the theoretical and empirical performance analyses of the time limit inference algorithms.

5.5.3.1 Throughput inference

The performance of this algorithm is quite simple to analyse. If the nodes are visited in topological order, the algorithm will only need to visit each node once. For each node, the algorithm will compute a constant-time expression on every incoming edge (multiplications for conditional edges, scalar comparisons for *JoinNodes* and sums for *MergeNodes*).

Let the model have n nodes and e edges. Since the algorithm visits each node and edge exactly once and spends a constant amount of time on each of them, the algorithm will require $O(n + e)$ operations. If the underlying graph is dense, then $e \in \Theta(n^2)$ and $O(n + e)$ becomes simply $O(n^2)$.

5.5.3.2 LP-based time limit inference

The time limit inference algorithms are harder to analyse than the throughput inference algorithm, as their performance depends on the structure of the underlying graph. For this reason, several graph shapes will be defined for the models (shown in Figure 5.8):

Sequence models consist of an *InitialNode* followed by one or more *Action* nodes in sequence, with a *FinalNode* at the end.

Fork-join models have an *InitialNode*, followed by a sequence of f “levels”. Each level has a *DecisionNode* with two branches with a single *Action*, merged before the next level. The model has $n = 2 + 4f \in \Theta(f)$ nodes and $e = 1 + 5f \in \Theta(f)$ edges in total, and there are 2^f paths from the *InitialNode* to the *FinalNode*.

Dense models have an *InitialNode*, followed by a sequence of f “levels”, like the fork-join models. However, the structure of each level is different: a *DecisionNode* picks between running an *Action* or jumping to any of the following levels. The model has $n = 2 + 3f \in \Theta(f)$ nodes and $e = 1 + 3f + \sum_{i=1}^f i \in \Theta(f^2)$ edges. There are $(f + 1)!$ paths from the *InitialNode* to the *FinalNode*. These models have many more edges

⁵Results may vary within this range due to floating point errors.

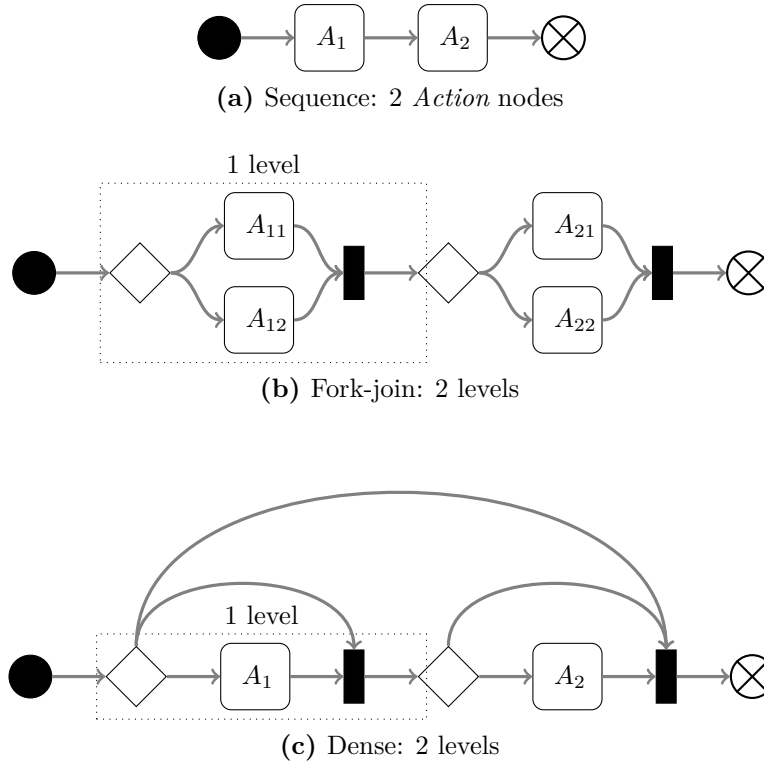


Figure 5.8. Graph shapes used in the performance analyses

and paths than the fork-join models: in fact, they represent the densest graph that can be built with this combination of nodes.

With these shapes in mind, the analysis of the performance of the linear programming-based time limit inference algorithm can be performed. Since there are many methods for solving LP problems (some of them very efficient on the average case), the focus will be instead placed on the size of the resulting LP problem. The LP problem will have n variables and will consist of an objective function which can be generated in $O(np)$ time and the following restrictions:

- The slack per unit of weight must be positive (R1): one per node. Each restriction can be generated in $O(1)$ time.

Component	$O(\text{r. count})$	$O(\text{gen. time})$	$O(\text{total time})$
Objective function		np	np
R1: $S_w(n) \geq 0$	n	1	n
R2: L per path	p	n	np
R3: minimum S_w	n	np	n^2p
R4: same paths	n^2	p	n^2p
Total	$n^2 + p$		n^2p

Table 5.5. Restriction counts and individual and total generation times for the LP-based algorithm, by component.

Shape	$O(\text{nodes})$	$O(\text{paths})$	$O(\text{r. count})$	$O(\text{total time})$
Sequence, n nodes	n	1	n^2	n^2
Fork-join, f levels	f	2^f	2^f	$f^2 2^f$
Dense, f levels	f	$f!$	$f!$	$f^2 f!$

Table 5.6. Restriction counts and generation costs for the LP-based algorithm, by graph shape, using the results from Table 5.5.

- The global time limit must be honored (R2): one per path, generated in $O(n)$ time by traversing every node in the path.
- The lower bound on the slack per of unit weight for every node (R3): one per node, generated in $O(np)$ time by traversing every path and computing $m(p)$ and $w(p)$ for it.
- The slack per unit of weight must be equal for all (m, n) pairs of nodes in the same paths (R4): one for every such pair, generated or discarded in $O(p)$ time by comparing $P_C(m)$ and $P_C(n)$.

These results are aggregated in Table 5.5. It can be concluded that $O(n^2 + p)$ restrictions will be generated in $O(n^2 p)$ operations. Table 5.6 applies these results to each of the three graph shapes in Figure 5.8. It can be seen that the rapidly increasing number of paths in the model is the main limiting factor for applying the algorithm to more complex models.

5.5.3.3 Exhaustive graph-based time limit inference

In the case of the exhaustive time limit inference algorithm, enumerating the p existing paths requires a depth-first traversal of the graph, which must traverse the $e \in O(n^2)$ edges in the graph. Each path in the graph will then have to be traversed a constant number of times to compute several expressions which require $O(n)$ operations.

Therefore, the algorithm requires $O(e + np)$ operations. Sequences only have $O(n)$ edges and $p = 1$, so they only require $O(n)$ operations in total. However, fork-join models would require $O(f + n2^f) = O(n2^f)$ operations, and dense models would require $O(f^2 + n(f+1)!) = O(nf!)$ operations. It can be seen that the rapidly increasing number of paths in the model is also the main limiting factor for applying this algorithm to more complex models.

5.5.3.4 Incremental graph-based time limit inference

Analysing the incremental graph-based time limit inference algorithm is harder than analysing the LP-based algorithm. For this reason, analysis will be limited in this case to fork-join models, proceeding by parts in the worst case:

- Computing $M_m(n)$ in advance for each node always takes $O(1)O(n) = O(n)$ operations, as it requires evaluating an arithmetic expression over the $O(1)$ incoming edges of each of the n nodes.

- Computing $P'_S(n)$ in advance for each node is actually the most expensive part of the algorithm: in the worst case, $O(2^f)$ paths need to be considered at every node and selecting the strictest ones takes $O(4^f)$ operations per node and $O(n4^f)$ in total.
- The last step depends on the number of elements of $P'_S(n)$ for each node n in the graph: in the worst case, $|P'_S(n)| = |P_S(n)|$ (no paths have been removed) for every node n and $O(n2^f)$ operations are required.

Joining the three parts of the algorithm yields a time of $O(n4^f)$ operations in the worst case for a fork-join model. The absolute worst case is very expensive. However, it is also very rare, as shown in Section 5.5.4.4.

5.5.4 Empirical performance

The previous section studied the definitions of the algorithms to derive several upper bounds for their execution times. It was concluded that the throughput algorithm required $O(n^2)$ operations, the LP-based time limit inference algorithm required $O(f^2 2^f)$ operations for fork-join models with f levels and the graph-based time limit inference algorithm required $O(n4^f)$ operations for the same fork-join models.

However, it was also concluded that these were very loose upper bounds, due to limitations in the analysis. For this reason, this section will discuss the results of several experiments based on the actual execution of the algorithms on a set of automatically generated models. It will be shown that the incremental graph-based algorithm requires much less time to run in practice than the LP-based and exhaustive graph-based algorithms, and that it does not show the exponential growth which would be expected from the previous upper bound. This is because the worst case becomes harder to find as models become more complex, as shown at the end of this section.

The performance tests were run in a machine with an Intel Core Duo T2250 CPU at 1.73GHz with 3.2GiB DDR3-1066 RAM, using Eclipse Juno (build ID M20130204-1200) and the latest development snapshot of Epsilon (revision 2253) to date. Wall clock times were measured using the facilities provided by the Java Virtual Machine, ensuring other processes remained idle during the tests. The studies in Sections 5.5.4.1, 5.5.4.2 and 5.5.4.3 were conducted using an Eclipse plug-in (see Figure 5.9) that was built for this study. Parts of the graph-based time limit algorithm were ported to C++ for the study in Section 5.5.4.4. All relevant code is freely available at [7].

5.5.4.1 Throughput inference

Figure 5.10 shows the average execution times for the throughput inference algorithm for the three graph shapes described in Figure 5.8. The generated sequence and dense models had between 0 and 50 actions, and fork-join models had between 0 and 25 levels.

The algorithm shows the expected level of performance for fork-join and sequence models. A model with 50 actions in a sequence model only takes 0.04 s, and a fork-join model with 50 actions takes 0.07 s. A dense model with 50 actions takes somewhat longer, requiring 0.46 s. This confirms the previous $O(n + e)$ bound for the algorithm: sparse graphs exhibit linear growth, while dense graphs show quadratic growth.

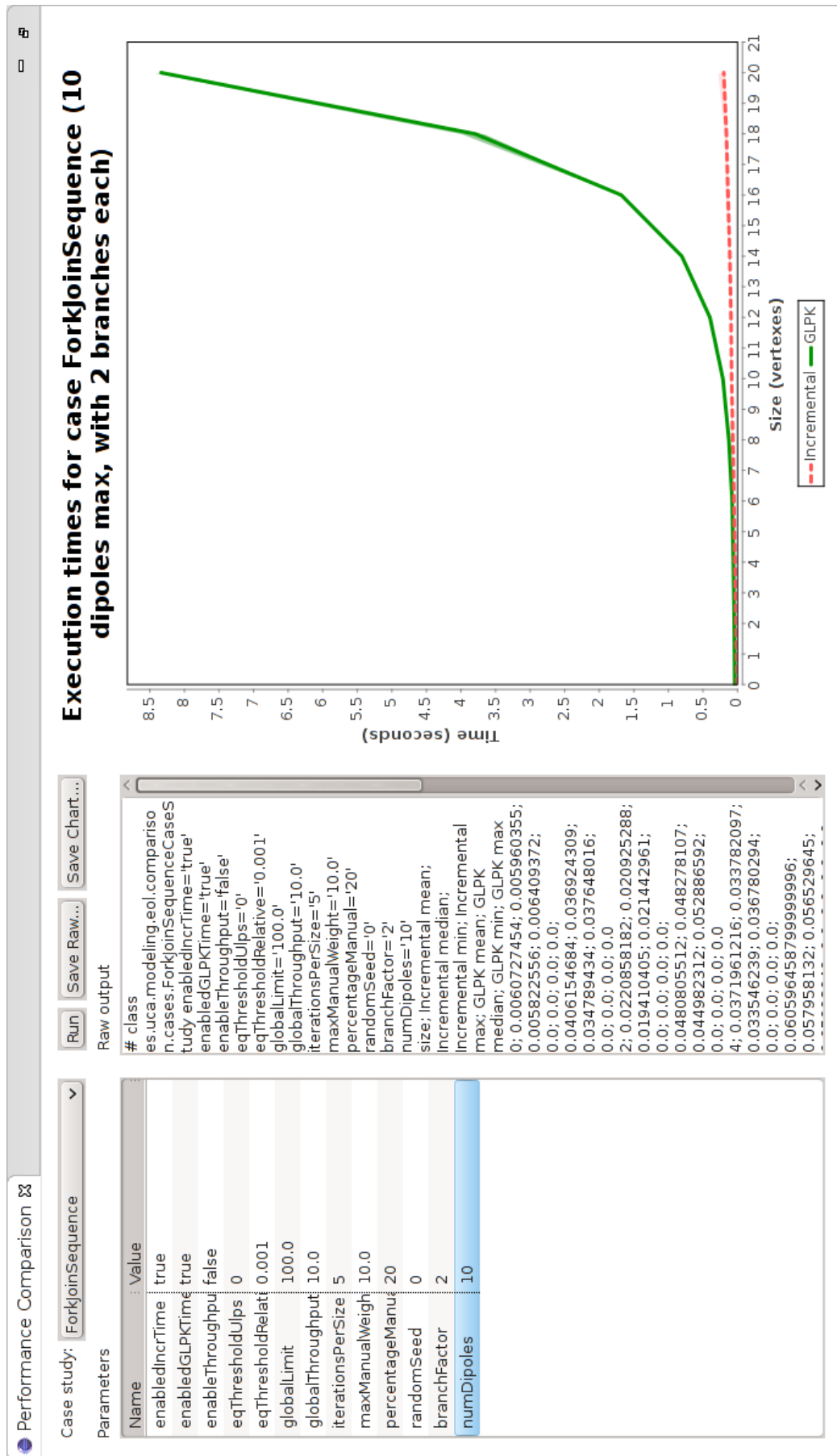


Figure 5.9. Screenshot of the automated Eclipse-based performance comparison tool. “GLPK” refers to the linear programming-based algorithm, “Exhaustive” refers to the exhaustive graph-based algorithm and “Incremental” refers to the incremental graph-based algorithm.

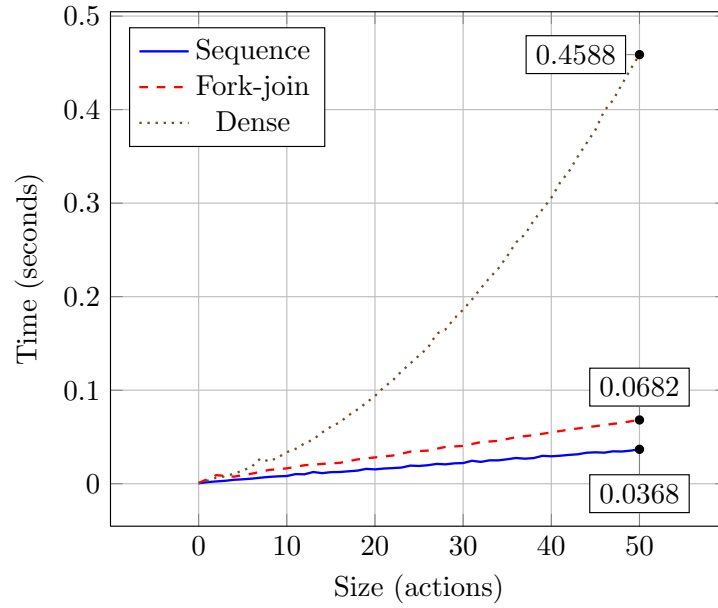


Figure 5.10. Average execution times over 10 runs for the throughput inference algorithm, by graph shape and size.

5.5.4.2 Comparison of the time limit algorithms

Figure 5.11 compares the times required by the LP-based and graph-based time limit inference algorithms for the three graph shapes listed in Figure 5.8. In this case, because of the rapid increase in cost of the LP-based algorithm, fork-join and dense models were limited to 20 and 12 actions, respectively.

For the simple sequence graphs, the LP algorithm is the fastest, as it relies on a native LP solver instead of an interpreted EOL program. However, the complexity of generating the LP problem and solving it quickly takes effect in the fork-join and dense graphs: it is always slower than one of the graph-based algorithms.

On the other hand, the incremental graph-based algorithm is only slower than the exhaustive graph-based algorithm for very small models. This may be due to the additional overhead imposed by its more complex design. Nevertheless, it quickly becomes faster than the exhaustive graph-based algorithm as the models become only slightly larger.

These tests checked that the results produced by both algorithms were the same, except for a minimal error margin (0.1%) due to floating-point rounding and error propagation. More formally, if r_l and r_g were the results of the LP-based and graph-based algorithm for the same input model, it was verified that $|r_l - r_g| / \max\{r_l, r_g\} \leq 0.001$.

5.5.4.3 Influence of annotations on the incremental graph-based time limit algorithm

After concluding that the graph-based time limit algorithm was clearly superior to the LP-based algorithm, it was decided to study the effect of the manual performance annotations on the graph-based time limit algorithm. Depending on the actual values used in this annotation, the algorithm may be unable to discard some paths, reducing the effectiveness of its optimisations over the LP-based algorithm.

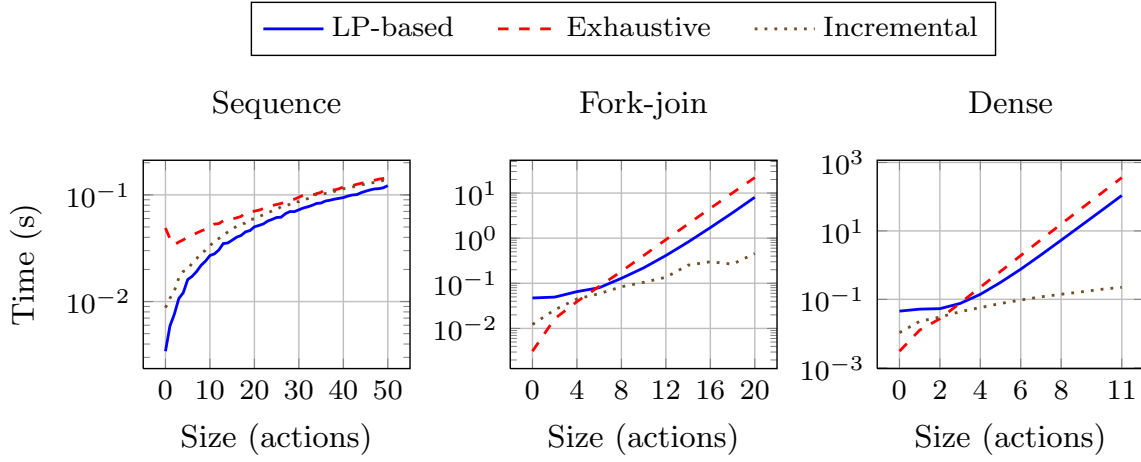


Figure 5.11. Average execution times over 10 runs for the time limit inference algorithms, by graph shape and size, with $L = 100$ s. All *Action* nodes were annotated with uniform random minimum times (between 0 and $0.5L$) and weights (up to 10). Execution times are represented in a base-10 logarithmic scale.

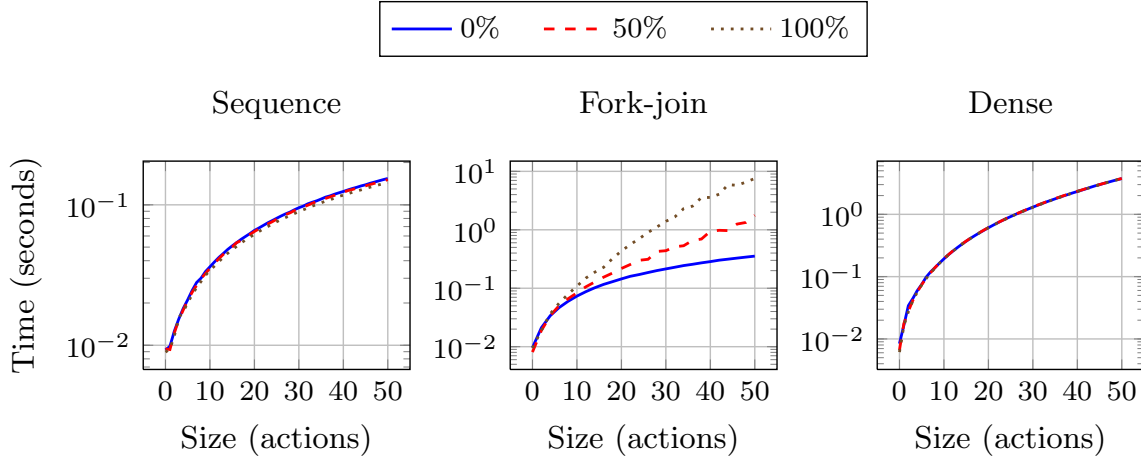


Figure 5.12. Average execution times over 100 runs for the incremental graph-based time limit inference algorithm, by graph shape, graph size and percentage of *Activity* nodes with uniformly random performance annotations. L was set to 100 s, minimum time limits ranged between 0 and $0.5L$ and weights ranged between 0 and 10. Execution times are represented in a base-10 logarithmic scale.

To study the impact of this issue on performance, the average time required by the graph-based time limit algorithm over 100 runs for each graph shape and size was measured. Either 0%, 50% or 100% of all *ExecutableNodes* with randomly generated performance annotations were annotated. Results are shown in Figure 5.12.

It is interesting to note that only fork-join models show notable differences between annotating 0%, 50% or 100% of all *ExecutableNodes*. This is obvious for sequence models, which only have 1 path, but it might be surprising for dense models, which have $f!$ paths for a model with f levels. This is because of (A.5) and the structure of the dense models. When choosing between a subpath (m, w) that does not run a certain node with minimum time m_a and weight w_a , and a subpath $(m + m_a, w + w_a)$ that does, (A.5) will discard (m, w) and only keep $(m + m_a, w + w_a)$. For this reason, at each *DecisionNode* only one subpath will need to be considered to find the strictest path from the *InitialNode* to the *FinalNode*. The observed faster-than-linear growth for dense models can be attributed to the need to traverse all $O(f^2)$ edges to precompute $M_m(n)$ for each node.

Going back to fork-join models, it can be seen that annotating all *ExecutableNodes* with custom local performance annotations is more expensive than always using the default zero minimum time and unit weight. This can also be explained through (A.5): when using the default performance annotations, it is always the case that $a = c = 0$ and (A.5) can be simplified into $b \leq d$, which is a total order. In that case, many more paths can be removed and the optimisations are much more effective. Otherwise, some paths may not be comparable (as $\preceq_{s(L)}$ is a partial order) and execution costs will increase. Nevertheless, even when all *ExecutableNodes* are annotated, execution times do not show the exponential growth of the exhaustive algorithm.

5.5.4.4 Worst case of the incremental graph-based time limit inference algorithm

So far, it has been shown that removing redundant paths is effective in avoiding the exponential growth in cost that affected the exhaustive time limit inference algorithm. However, its effectiveness depends on the values of the annotations used in the model. A closer look at (A.5) shows that it depends on the relative magnitude of the minimum time limits and weights with regards to the global time limit L . The left operand of $(b-d)L < bc-ad$, part of (A.5), grows as L increases and reduces the number of comparable pairs of paths.

An additional study was performed to clarify how common the absolute worst case was and study its relationship with L . A sample was conducted with $L = 0.5$ s and $L = 1.5$ s of the space of all fork-join models with 3 levels which contained a 2-level fork-join with 4 incomparable paths. Minimum times for the *ExecutableNodes* ranged from 0 to $\min\{L, 1\}$, in steps of 0.1 s. Weights ranged from 0 to 10, in steps of 1 unit. Inconsistent models were discarded. For each model, the number of incomparable paths at the initial node (“top-level paths”) was measured: in a 3-level fork-join model, there can be between 1 and $2^3 = 8$ such paths.

Evaluating 1.99×10^6 fork-join activities for $L = 0.5$ s and 7.16×10^9 for $L = 1.5$ s produced the results in Figure 5.13. For $L = 0.5$ s, less than 10% of these models had more than 1 incomparable path. With $L = 1.5$ s, less than 20% of the models had more than 2 incomparable paths.

Furthermore, it is interesting to note that for $L = 1.5$ s, while 31.8% of all 1-level fork-join models were in the worst case, only 2.5% 2-level fork-join models were in the

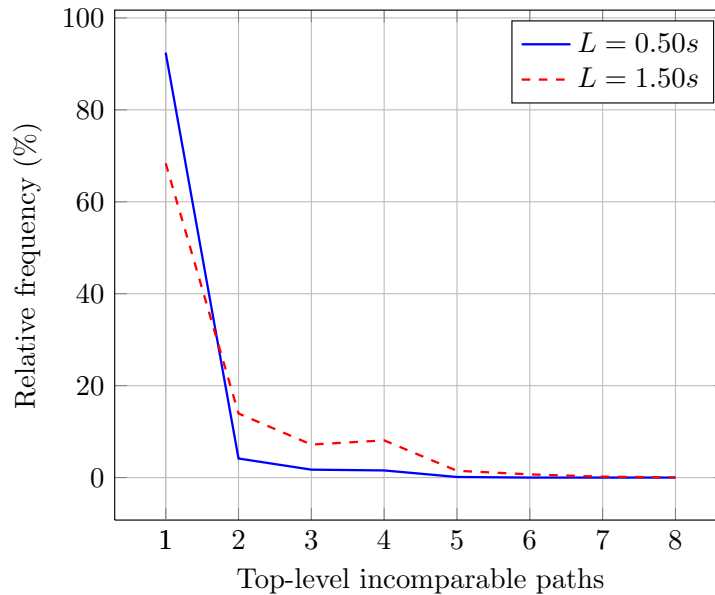


Figure 5.13. Percentages of sampled 3-level fork-join models with a certain number of top-level incomparable paths, 4 incomparable paths at the second level and 2 incomparable paths at the last level, by values of the global time limit L .

worst case. With 3 levels, it was further reduced to 0.05%. This suggests that the absolute worst case becomes harder to find as models become more complex, explaining why average times did not grow exponentially in Figure 5.12. Additionally, it indicates that the worst case becomes more common as L grows in relation to the values used in the annotations.

5.6 Conclusions

This chapter has presented the first version of SODM+T, an extended version of the SODM methodology which provides guidance on defining the performance requirements of the service-oriented system. SODM+T extends the service process and service composition models with global performance requirements and local performance annotations, which are combined to derive local performance requirements.

The current extensions model throughput and response times using a custom notation backed by automated validation, Eclipse-based model editors and several performance inference algorithms. Bounded and well-structured iteration can be represented in the models using the appropriate annotations.

The throughput inference algorithm is a simple algorithm based on traversing the graph in topological order. Inferring time limits was much more complex: the first formulation was based on producing a sequence of linear programming problems, but had exponential cost as models became larger. For this reason, two graph-based algorithms were developed: a simple exhaustive algorithm that visited all the paths, and a more advanced incremental algorithm that discarded uninteresting subpaths as soon as possible.

An initial theoretical evaluation of the algorithms showed that the throughput inference algorithm had $O(n^2)$ cost for dense graphs and that all the time limit inference algorithms

had exponential upper bounds, due to the potentially exponential number of paths from the initial nodes in the graph.

This initial evaluation was checked by running the algorithm against three kinds of automatically generated models, confirming the expected bounds of the inference algorithm and the LP-based and exhaustive time limit inference algorithms. However, the incremental time limit inference algorithm did not show an exponential order of growth in the average case. A later study backed this claim by verifying that the absolute worst case for the algorithm became more rare as models increased in size, thanks to its optimisations. This means that the incremental time limit inference algorithm can be used with models of considerable size, even in the presence of an exponential number of paths.

References

- [1] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: a survey. *IEEE Transactions on Software Engineering*, May 2004. doi: 10.1109/TSE.2004.9. 5.1
- [2] M. A. Bender, G. Pemmasani, S. Skiena, and P. Sumazin. Finding least common ancestors in directed acyclic graphs. *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, pages 845–853, 2001. doi: 10.1.1.15.9161. 5.7
- [3] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web*, April 2004. doi: 10.1016/j.websem.2004.03.001. 5.1
- [4] Eclipse Foundation. Graphical Modeling Project, 2013. URL <http://www.eclipse.org/modeling/gmp/>. Last checked: November 6th, 2013. 5.4
- [5] Eclipse Foundation. Emfatic project homepage, 2013. URL <http://www.eclipse.org/modeling/emft/emfatic/>. Last checked: November 6th, 2013. 5.6
- [6] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: a modeling language for mathematical programming*. Thomson/Brooks/Cole, California, EEUU, 2003. ISBN 9780534388096. 5.24
- [7] A. García-Domínguez. Homepage of the SODM+T project, April 2013. URL <https://neptuno.uca.es/redmine/projects/sodmt>. 5.28, 5.32
- [8] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991. doi: 10.1145/103162.103163. 5.8
- [9] D. S. Kolovos, L. M. Rose, R. F. Paige, and A. García-Domínguez. The Epsilon book, 2013. URL <http://dev.eclipse.org/svnroot/modeling/org.eclipse.epsilon/trunk/doc/org.eclipse.epsilon.book/EpsilonBook.pdf>. Last checked: November 6th, 2013. 5.8

- [10] Object Management Group. UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) 1.1, June 2011. URL <http://www.omg.org/spec/MARTE/1.1/>. Last checked: November 6th, 2013. 5.28
- [11] L. M. Rose, D. S. Kolovos, R. F. Paige, and F. A. C. Polack. Model migration with Epsilon Flock. In D. Hutchison, T. Kanade, J. Kittler, et al., editors, *Theory and Practice of Model Transformations*, volume 6142, pages 184–198. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-13687-0. 5.13
- [12] G. A. Silver, A. Maduko, J. Rabia, J. Miller, and A. Sheth. Modeling and simulation of quality of service for composite web services. In *Proceedings of 7th World Multiconference on Systemics, Cybernetics and Informatics*, pages 420–425. Int. Institute of Informatics and Systems, November 2003. 5.1

6

Generation of test artefacts with SODM+T and MARTE

The previous chapter presented the first version of SODM+T, an extended version of SODM which added custom performance annotations to the service process and service composition models. These annotations were used to do an early estimation of the performance requirements of each service, according to the global requirements of each model.

This chapter will improve on SODM+T by mapping the annotations to the standard OMG MARTE profile, which has better support from industrial tooling. This will be followed by some minor refinements that were later made to the performance inference algorithms. The rest of the chapter will discuss a novel approach to generating performance test artefacts from the annotated models and the final implementations of the services, by using model weaving. This approach is applied to two scenarios: repurposing unit tests for Java code, and generating language-independent performance tests for WSDL-based WS.

6.1 The MARTE profile

UML is widely used as a general purpose modelling language for software systems (for a short introduction, see Section 4.3.1). However, UML cannot model non-functional aspects such as performance requirements.

For this reason, the OMG proposed in 2005 the SPT profile [17], which extended UML with a set of stereotypes describing scenarios that various analysis techniques could take as inputs. In 2008, OMG proposed the QoS/FT profile [18], with a broader scope than SPT and a more flexible approach: users formally defined their own quality of service vocabularies to annotate their models.

When UML 2.0 was published, OMG saw the need to update the SPT profile and harmonise it with other new concepts. This resulted in the MARTE profile [19], published in 2009. Like the QoS/FT profile, the MARTE profile defines a general framework for describing quality of service aspects. The MARTE profile uses this framework to define a set of pre-made UML stereotypes, as those in the SPT profile.

The rest of this section presents the architecture of the MARTE specification and focuses on the key subset on which the performance annotations have been mapped.

6.1.1 Architecture

The MARTE profile is a complex specification, spanning over 700 pages. It is organised into several subprofiles and includes a normative model library with predefined types and concepts and an embedded expression language known as the Value Specification Language (VSL). Figure 6.1 lists each of the packages that constitute MARTE and their elements:

- The “MARTE foundations” package defines the core concepts that are used throughout the other profiles, such as the concept of a non-functional property (NFP), how to model time or resources or how to map the functional elements to the resources. Resources are modelled through the General Resource Modelling (GRM) subprofile.
- The “MARTE analysis model” package is used to annotate application models to support analysis of system properties. The GQAM subprofile uses the foundations package and the normative model library to provide a base set of concepts for the

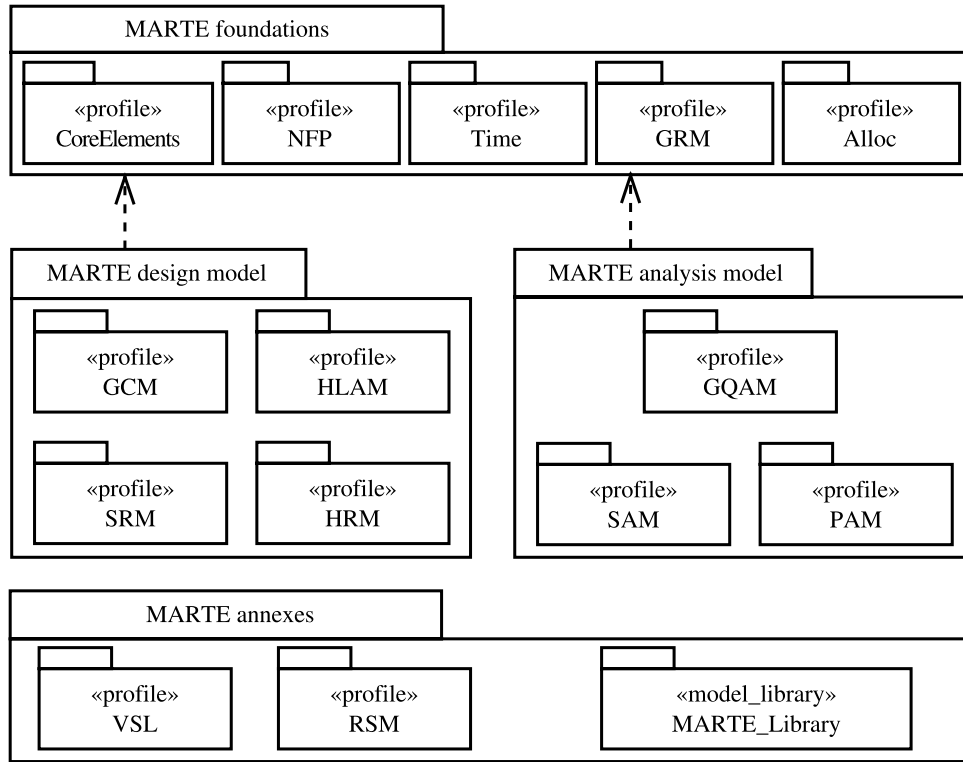


Figure 6.1. Architecture of the MARTE profile [19]

two kinds of analysis supported by MARTE: schedulability analysis and performance analysis. Schedulability analysis predicts whether a set of software tasks meets its timing constraints and is modelled using the Schedulability Analysis Modelling (SAM) subprofile. Performance analysis determines whether a system with non-deterministic behaviour can provide adequate performance, and is supported through the PAM subprofile.

- The “MARTE design model” package provides the required concepts for modelling the features of real-time and embedded (RT/E) systems. The Generic Component Modelling (GCM) subprofile provides additional core concepts for Real Time/Embedded (RT/E) systems. The High-Level Application Modelling (HLAM) subprofile provides the concept of a real-time execution unit that manages several resources and a queue of messages with various real-time requirements. Finally, the Detailed Resource Modelling (DRM) subprofile provides facilities for describing the software and hardware resources used by the system.
- The “MARTE annexes” package includes the Value Specification Language (VSL) used for all MARTE expressions, the Repetitive Structured Modelling (RSM) package for describing available software and hardware parallelism, and the normative MARTE model library.

The normative MARTE library defines the set of standard primitive types (such as real numbers or integers) and derived types (such as vectors of integers or NFPs involving a real value), among many other concepts.

6.1.2 GQAM

Using the GQAM subprofile requires the definition of an *AnalysisContext*, which is formed by a *WorkloadBehavior* object (the workload to be run) and a *ResourcesPlatform* object (the resources to be used). An *AnalysisContext* may also include a set of user-defined context parameters, which will be available as variables in the VSL expressions of the NFP.

The workload is then divided into the *WorkloadEvent* describing the request arrival pattern, and the *BehaviorScenario* specifying how these requests should be handled and the NFPs for them. A *BehaviorScenario* is further divided into *Steps* which are ordered using *PrecedenceRelations* of several kinds, such as sequential, branching, merging, forking or joining relations. Each *Step* may have NFPs of its own. These NFPs include response time, throughput, utilisation or the expected number of repetitions.

Finally, the GQAM concepts are mapped to UML stereotypes. For instance, *AnalysisContext*, *BehaviourScenario* and *Step* are mapped to the `<<GaAnalysisContext>>`, `<<GaScenario>>` and `<<GaStep>>` stereotypes, respectively.

6.1.3 VSL

As mentioned above, the GQAM *BehaviorScenario* and *Step* classes can contain NFPs for many aspects. However, properly describing the value of a NFP requires more than a simple scalar value: it is required to describe aspects such as measurement sources, measurement unit, precision and so on. In addition, the value of a NFP may be derived from a complex expression using several context parameters. All these features can be described using the Value Specification Language (VSL) embedded within the MARTE profile.

VSL provides a set of datatypes that extends the primitive types available in UML with composite types (such as intervals, collections or tuples) and subtypes. It also provides a textual syntax for complex expressions that may use conditional, arithmetic or relational operators, evaluate time expressions or process collections, among other features. Both can be combined: for instance, $(expr=2+3*f, ms, req)$ is a VSL tuple that represents a duration in milliseconds (*ms*) that has been required by the developer (*req*) and is computed from the *f* context parameter as $2 + 3f$.

6.2 Changes in SODM+T for MARTE

Having introduced MARTE more in depth, this section will present the changes that were performed in the SODM+T models and algorithms from Chapter 5 to better support MARTE and handle more complex UML constructions.

6.2.1 Revised annotations

The models are used for early performance inference, and so the Performance Analysis Modelling (PAM) subprofile would appear to be the best starting point. However, the focus of the algorithms is different than the one favoured by PAM, which is predicting the performance of the whole system from its parts. Instead, the algorithms infer the

From	To	Changes
Performance- Annotation <i>a</i>	<i>GaScenario s</i> , <i>GaAnalysis- Context c</i>	<i>s.respT</i> = <i>a.secsTimeLimit</i> (<i>source</i> = <i>req</i>). <i>s.throughput</i> = <i>a.concurrentUsers</i> (<i>source</i> = <i>req</i>). <i>c.contextParam</i> has one entry per <i>ExecutableNode</i> (its slack per unit of weight).
Local- Performance- Annotation <i>a</i>	<i>GaStep s</i>	<i>s.rep</i> = <i>a.reps</i> . <i>s.hostDemand</i> = <i>m</i> + <i>ws</i> , where <i>m</i> is <i>a.minimumTime</i> , <i>w</i> is <i>a.weight</i> and <i>s</i> is the context parameter for <i>a.execNode</i> (<i>source</i> = <i>req</i>). Inferred time limits are added to <i>s.hostDemand</i> (<i>source</i> = <i>calc</i>). Inferred throughputs are added to <i>s.throughput</i> (<i>source</i> = <i>calc</i>).
<i>ControlFlow c</i> (annotated)	<i>ControlFlow c'</i> (not annotated), <i>GaStep s</i>	<i>s.prob</i> = <i>c.probability</i> .

Table 6.1. Mapping from the SODM+T custom annotations to MARTE

performance needed in each part of the system from the global requirements. For this reason, only the generic analysis core has been used: the GQAM subprofile.

To keep the models simple, the notation only uses the three stereotypes in Section 6.1.2. The mapping is summarised in Table 6.1. Figure 6.2 shows an example model, which is simpler than the one in Figure 5.4 due to space restrictions. Inferred annotations are highlighted in bold:

1. The activity is annotated with a `«GaScenario»` stereotype, in which *respT* specifies that every request is completed within 1 second, and *throughput* specifies that 1 request per second needs to be handled. These expressions have their *source* attribute set to *req*, as they represent explicit requirements from the developer.
2. In addition, the activity declares a set of context parameters in the *contextParam* field of the `«GaAnalysisContext»` stereotype. These variables represent the slack per unit of weight that must be allocated to their corresponding activity in addition to the minimum required time. Their values are computed by the time limit inference algorithm.
3. Each action in the activity is annotated with `«GaStep»`, using in *hostDemand* a VSL expression of the form *m* + *ws*, where *m* is the minimum time limit, *w* is the weight of the action for distributing the remaining time, and *s* is the context parameter linked to that action. These expressions also have their *source* attribute set to *req*, for the same reasons as those in `«GaScenario»`. The estimated number of repetitions is stored in *reps*.

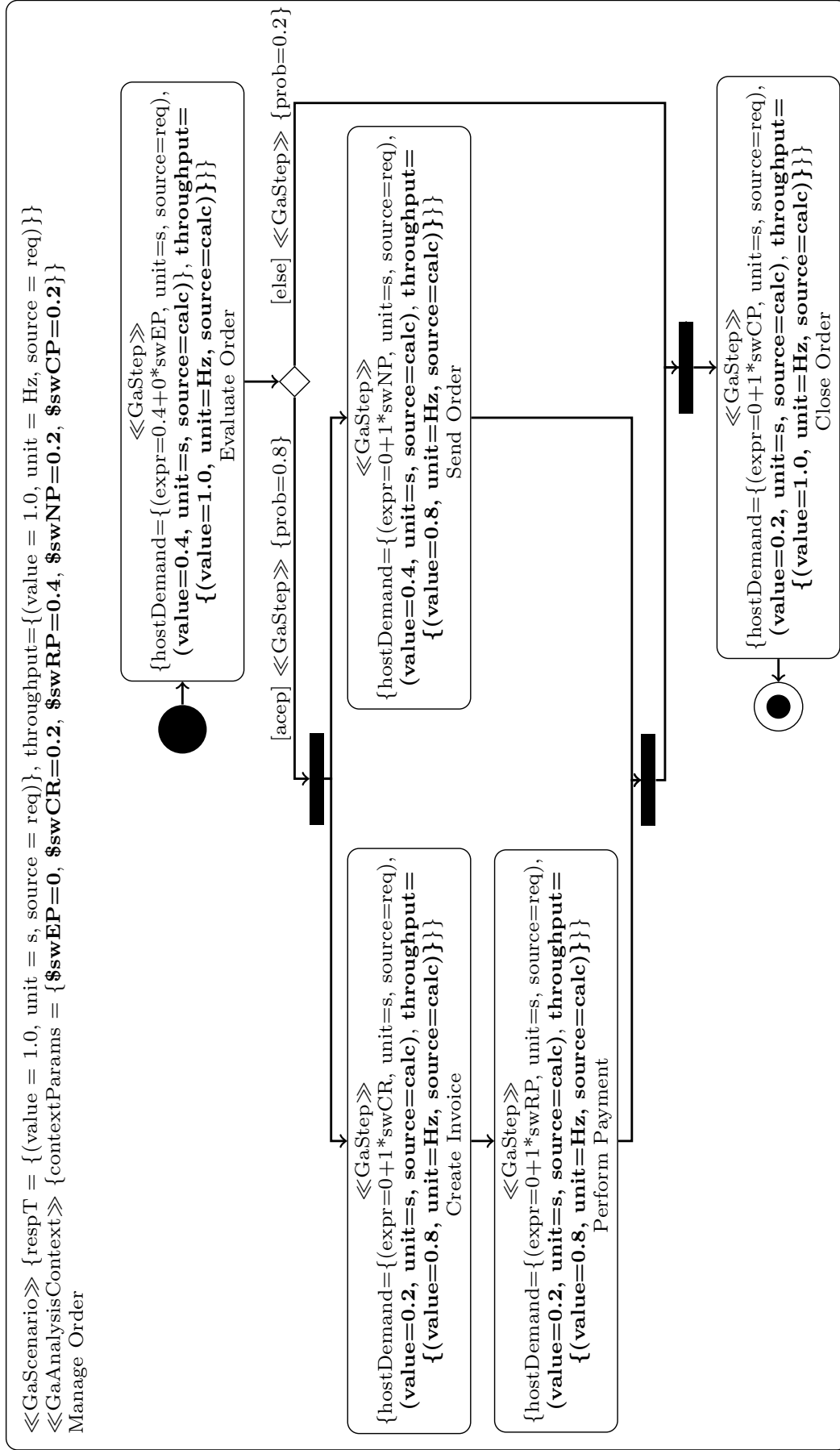


Figure 6.2. Simple example model annotated by the performance inference algorithms

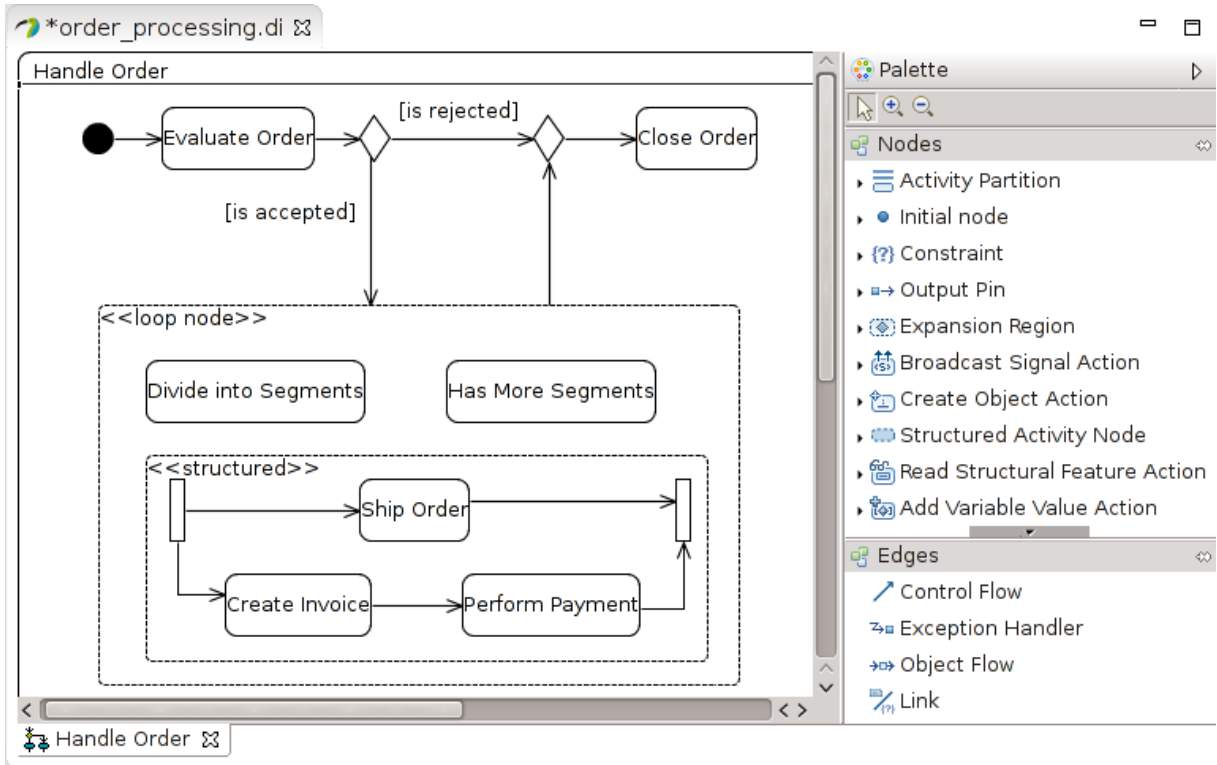


Figure 6.3. Screenshot of the Eclipse Papyrus editor, showing the example from Figure 5.4 ported to MARTE and the Eclipse MDT UML2 metamodel.

The time limit inference algorithm adds a new constraint to *hostDemand*, indicating the exact time limit to be enforced. The throughput inference algorithm extends *throughput* with a constraint that lists how many requests per second should be handled. As these constraints have been automatically inferred, their *source* attribute is set to *calc* (calculated).

4. Outgoing edges from condition nodes also use `<<GaStep>>` but only for the *prob* attribute, which is set by the user to the estimated probability it is traversed.

6.2.2 Revised algorithms

Among the algorithms described in Chapter 5, the throughput algorithm and the incremental graph-based time limit inference algorithm were migrated to MARTE annotations. In addition to this change, the algorithms were improved in several other ways:

1. The algorithms now operate on models conforming to the metamodel from the Eclipse Model Development Tools (MDT) UML2 project [9]. Instead of using a custom model editor, the models are produced using the Papyrus model editor [10] (shown in Figure 6.3), which supports both UML2 and MARTE. Users apply the MARTE stereotypes and invoke the inference algorithms through several extensions written on the Epsilon Wizard Language [16].
2. Instead of only having *Actions* and *StructuredActivityNodes*, the algorithms now partition all UML activity model elements into two classes: the subclasses of

ExecutableNode (which can be annotated and have performance requirements) and the rest (*DecisionNodes*, *ForkNodes* and so on).

3. After revisiting the full UML metamodel, it was found that another *ActivityNode* could contain nodes in a different way than the *StructuredActivityNode*: the *LoopNode*.

Instead of simply nesting nodes inside each other, these nodes divide their contents into three sections: *setup* (which is only run once, before the rest), *test* (which is run once per iteration until it fails) and *body* (which is run once per iteration if the test passed). Figure 6.3 shows a modified version of the running example from Figure 5.4 using a *LoopNode*: the *setup* section is the “Divide into Segments” *Action*, the *test* section is the new “Has More Segments” *Action* and the *body* section is the original *StructuredActivityNode* with the contents of the loop itself.

Supporting these *LoopNodes* required changing two assumptions. First, not every set of activities (nested or not) started at an *InitialNode* and ended at one or more *FinalNodes*: instead, they started at a *source* and ended at one or more *sinks*. A source is a node that has no incoming edges, and a sink is a node that has no outgoing edges. The sources of the entire *Activity* and any *StructuredActivityNodes* are easy to find. For the *LoopNode*, the algorithm tries to find the sources first in the setup part, then in the test part and then in its body part. An equivalent approach is followed for the sinks.

Second, the execution precedence of the *ExecutableNodes* was not always indicated through *ControlFlows*: e.g. the setup part went before the test part without any such edge connecting them. Instead, the concepts of the *next* and *previous* nodes of a certain node *n* were defined. If a node has incoming edges, the previous nodes are the sources of these edges: otherwise, the previous nodes are the sinks of the previous *LoopNode* section. Likewise, if a node has outgoing edges, the next nodes are the targets of these edges: otherwise, the next nodes are the sources of the next *LoopNode* section.

6.3 Overall approach for test artefact generation

The models used so far are dedicated to early performance requirement design, and are thus entirely abstract: at that level of detail, they cannot be executed automatically. They need to be implemented through some other means.

After a model has been implemented, it would be useful to take advantage of the model to generate the performance test artefacts for its implementation. However, the model lacks the required details to produce executable artefacts. To solve this issue, several approaches could be considered:

1. The abstract model could be extended with additional information, but that would clutter it and make it harder to understand.
2. On the other hand, the implementation models could be annotated with performance requirements, but this would also pollute their original intent.

3. Finally, a separate model that links the abstract and concrete models could be used. This is commonly known as a *weaving model*. Several technologies already exist for implementing these, such as the AMW [8] or Epsilon ModeLink [15]. While AMW uses a generic weaving metamodel, ModeLink is a more lightweight approach that requires defining custom weaving metamodels for every pair of metamodels.

In order to preserve the cohesiveness of the abstract performance model and the design and implementation models, the third approach was chosen. The weaving model needs to allow users to annotate the links with the additional information required by the testing process, the target technologies and the generation process itself. Target technologies refer not only to the performance testing framework or tool which will run the generated tests, but also all the components which will be part of the test infrastructure. As Section 6.5 will later show, this may include IDEs (e.g. Eclipse) or build automation tools (e.g. Maven).

Some of the information may be shared by a set of tests (possibly all of them), and some of the information will be specific to a particular link between a design/implementation artefact and a performance requirement. For instance, while the number of threads used to exercise the system under test may need to be the same for all the tests, the interpretation of the time limit requirement as a median, an average or a percentile may change from test to test.

After establishing the required links, the next step is generating the tests themselves. To do so, a regular M2T transformation could be used, written in a specialised language such as the Epsilon Generation Language (EGL) [16]. In case it were necessary to slightly refine or validate the weaving model before, an intermediate M2M transformation could be added. Figure 6.4 illustrates the models and steps involved in the overall approach.

In some cases, it may be desirable to allow users to easily customise certain interesting parts of the tests, while abstracting them from the details that are less interesting. These interesting parts could be written into a custom domain-specific language instead of code, which would be interpreted as the tests were executed by augmenting the testing infrastructure accordingly. Section 6.5.4 will later present such an example, written in the TestSpec language.

The next sections will show two applications of the overall approach in Figure 6.4, using different technologies to assist in generating performance test artefacts in different environments. Both approaches have been implemented and are freely available under the open source Eclipse Public License (EPL) at [12]. In order to develop these transformations, a bottom-up approach was used: a manually developed performance test environment was gradually replaced by automatically generated fragments until only the weaving model remained. After the entire process had been automated, the generators were refined to allow for more flexibility and convenience.

6.4 Reusing Java unit tests as performance tests

Generating executable performance test cases from scratch automatically will usually require many detailed models and complex transformations, which are expensive to produce and maintain. The initial effort required may deter potential adopters. An alternative inexpensive approach is to repurpose existing functional tests as performance tests as a starting point. This is the aim of libraries such as ContiPerf [6].

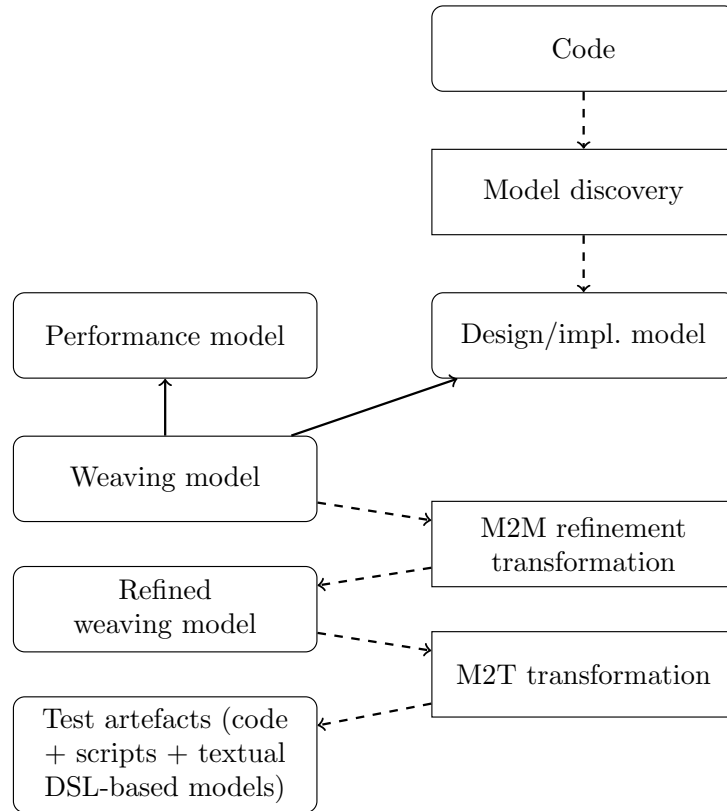


Figure 6.4. Overall approach for generating performance test artefacts from abstract performance models

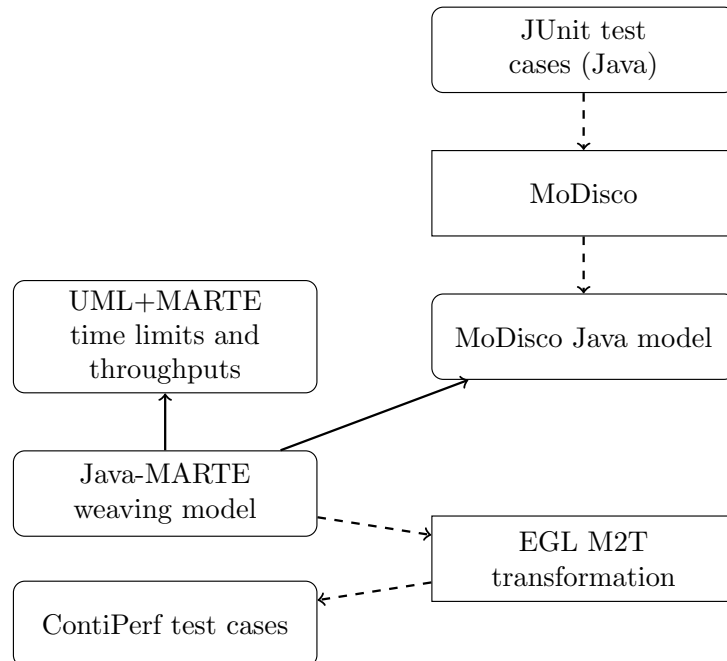


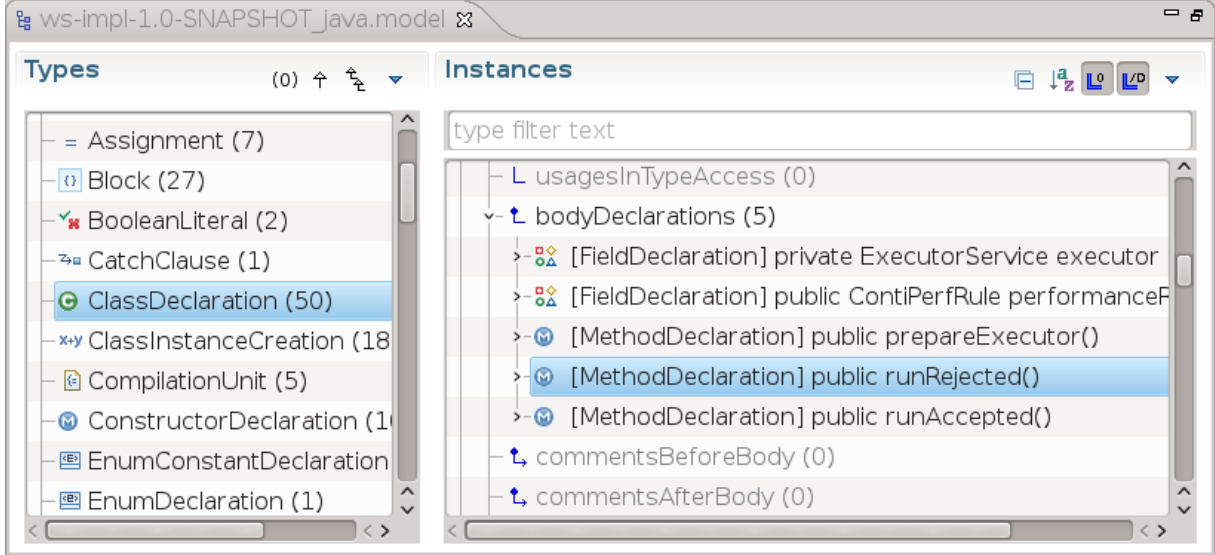
Figure 6.5. Instance of the above approach for wrapping JUnit tests into ContiPerf tests

Listing 6.1 Java code wrapping the *TFunctionalJUnit4* JUnit 4 test suite with ContiPerf

```

@RunWith(ContiPerfSuiteRunner.class)
@SuiteClasses(TFunctionalJUnit4.class)
@PerfTest(invocations = 100, threads = 10)
@Required(max=1000)
public class InferredLoadTest {}

```

**Figure 6.6.** MoDisco model browser showing a model generated from a Java project

Listing 6.1 shows how ContiPerf is normally used. Instead of using Java objects, ContiPerf uses Java 6 annotations, which are easier to generate automatically. The `@PerfTest` annotation indicates that the test will be run 100 times using 10 threads, so each thread will perform 10 invocations. `@Required` indicates that each of these invocations should finish within 1000 milliseconds at most. `@SuiteClasses` points to the JUnit 4 test suites to be reused for performance testing, and `@RunWith` tells JUnit 4 to use the ContiPerf test runner.

The rest of the section will show how the overall approach in Figure 6.4 was customised for this particular use case. The resulting transformation chain is shown in Figure 6.5.

6.4.1 Model extraction

The code above is straightforward to generate. However, the generated code must integrate correctly with the existing code, and if the code was not produced using a model-driven approach, there will not be a design or implementation model to link to. Instead, this approach extracts a model from the existing code using the Eclipse MoDisco model discovery tool [7].

Eclipse MoDisco can generate models from Java code such as that shown in Figure 6.6. The models can span entire projects if desired, but in this case only the test sources need to be inspected. MoDisco provides its own metamodel for the Java language, which is quite complex, having 126 classes¹. Nevertheless, it is enough to know which classes and

¹<http://help.eclipse.org/kepler/topic/org.eclipse.modisco.java.doc/>

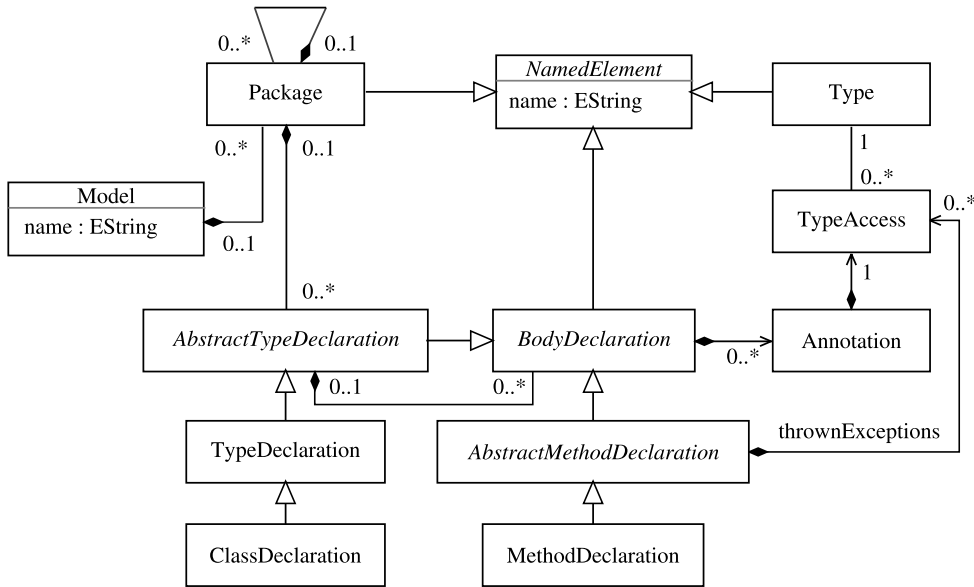


Figure 6.7. UML class diagram of the simplified subset of the MoDisco Java metamodel used for weaving and code generation

methods use the Java annotations from the JUnit test framework. A simplified version of this subset of the MoDisco metamodel is shown in Figure 6.7: a *Model* refers to a hierarchy of *Packages*, which contain *ClassDeclarations*. These *ClassDeclarations* and their *MethodDeclarations* may have *Annotations* of a certain *Type*. Since a *Type* is a *NamedElement*, these can be filtered by the attribute *name*. In some cases, it may also be necessary to know the exceptions thrown by a certain method: these are stored in the *thrownExceptions* association of the *AbstractMethodDeclaration*.

6.4.2 Weaving metamodel

Once the performance and the implementation models have been produced, the next step is to link them using a new weaving model that conforms to the metamodel in Figure 6.8. Some of the types in the weaving metamodel refer to types in the *uml* and *java* packages from the UML metamodel and the above MoDisco Java metamodel, respectively.

Each model consists of an instance of *PerformanceRequirementLinks*, which provides several global configuration parameters and contains a set of *PerformanceRequirementLink* instances. Users can set the number of samples which should be collected for each test, the number of threads over which these should be distributed and the directory under which the code should be generated.

Every *PerformanceRequirementLink* relates an UML *ExecutableNode* with a Java class: if no *MethodDeclarations* are specified, all tests will be reused. Otherwise, only the selected methods will be reused. Finally, the target time limit may be enforced as a maximum value (*MAX*), average (*AVERAGE*), median (*MEDIAN*) or a percentile (the rest).

6.4.3 Code generation

Models are populated by combining the standard EMF tree-based editors and the three-pane Epsilon ModeLink editor (as in Figure 6.9). ModeLink provides a drag-and-drop

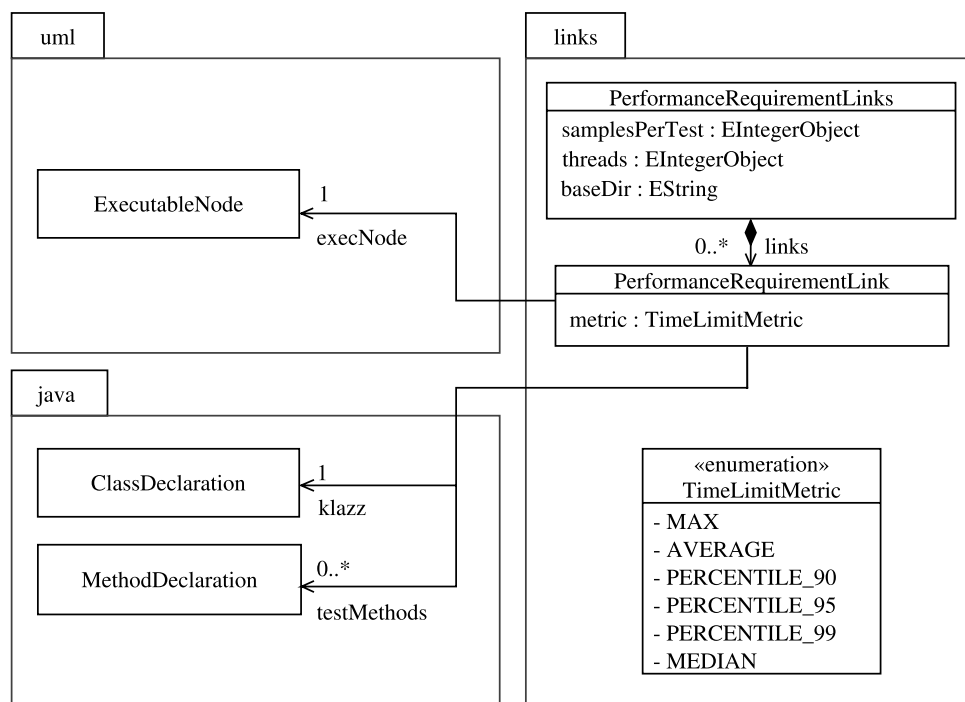


Figure 6.8. Java-MARTE weaving metamodel

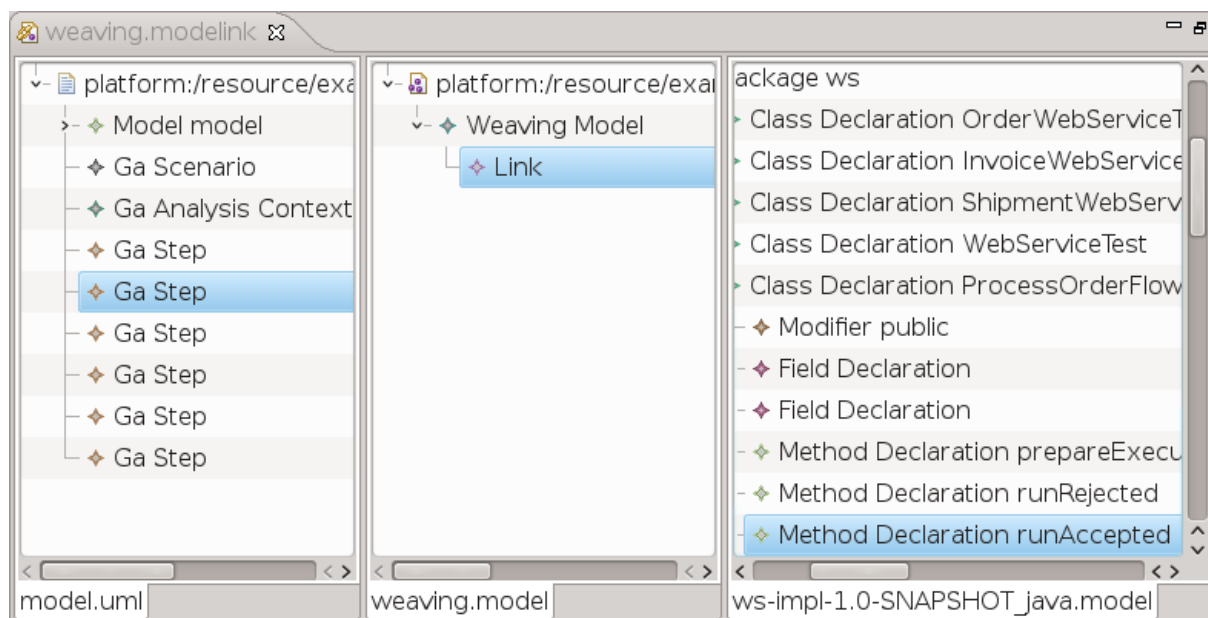


Figure 6.9. Screenshot of the Epsilon ModelLink editor weaving the MARTE performance model and the MoDisco model

Listing 6.2 Java code wrapping one test from *OriginalSuite* using ContiPerf

```
@Required(throughput=2, max=400)
public class WrapSomeTests extends OriginalSuite {
    @Rule public MethodRule f = new FilterByClassRule(this.getClass());

    @Rule public ContiPerfRule i = new ContiPerfRule();

    @PerfTest(invocations=1000, threads=5) @Test @Override
    public void first() throws Exception {
        super.first();
    }

    // protected region customTests off begin
    // Add your own tests here
    // protected region customTests end
}
```

approach to model linking that is convenient for model weaving. The EMF editors have been manually customised so users may only pick JUnit 4 test suites and test methods.

The code is generated using a set of EGL templates. When all tests are reused as performance tests, the generated code will use the test runner class *ContiPerfSuiteRunner*, as in Listing 6.1. In addition, the EGL template creates a *protected region* where developers may add their own custom code. This code will be preserved even if the rest of the file is overwritten.

However, when only some tests are wrapped the code will resemble that in Listing 6.2. The *ContiPerfRule* would normally convert all tests into performance tests. By using the *FilterByClassRule* helper class (also generated with EGL), the generated code will be able to specify that only some of those tests need to be reused as performance tests.

6.5 Generating performance tests for WSDL-based Web Services

In the previous section, the approach was applied to existing JUnit test cases, repurposing them as performance test cases. This approach could be applied to Web Services as is: Listing 6.3 shows an example fragment of Java code that implements a simple “HelloWorld” Web service using standard Java API for XML Web Services (JAX-WS) [14] annotations. This example could be easily tested as a regular Java method using standard Java tooling.

However, testing in this way would not exercise the serialisation and deserialisation of the input and output messages and other aspects that may raise issues during real world usage. Ideally, the WS should be tested using realistic invocations that exercised the entire technology stack involved. Instead of a Java unit testing framework, a proper WS performance testing tool should be used.

In practice, the WS developed by JAX-WS and other similar frameworks are consumed through the Web Services Description Language (WSDL) [22] documents they produce from the code. This XML-based document is an abstract and language-independent description of the available operations for the service and the messages to be exchanged between the service and its consumers. Therefore, it is an good starting point for a weaving model for generating performance test artefacts in a language-agnostic manner.

The rest of this section will discuss how to generate performance test artefacts for a Web

Listing 6.3 Java code using JAX-WS for a “HelloWorld” Web Service

```
@WebService public class HelloWorld {  
    @WebMethod public String greet(@WebParam(name="name") String name) {  
        return "Hello_" + name;  
    }  
}
```

Service, from scratch and in a language agnostic manner, using a specialised performance testing tool. The implemented solution is outlined in Figure 6.10.

6.5.1 Target performance testing tool: The Grinder

The previous section reused unit tests written in a particular language (Java) and a particular framework (JUnit). Therefore, the target technology was an extension upon this framework (ContiPerf). However, since the WSDL description of a Web Services (WS) does not depend on the language that it is implemented in, an approach based on a WSDL document would not be limited to a specific language for the tests. In particular, it could use a dedicated performance testing tool that is independent of the implementation language of the software under test.

For this Thesis, the following tools were evaluated according to the ease with which test specifications could be generated for them, by developing a simple performance test on a single service with each of them and studying the files required by the tools:

- The Grinder [4] used textual configuration files to configure the test environment, which executes Jython scripts that use the public API provided by the tool.
- Apache JMeter [2] used reflective XML documents. Most of their contents were directly translated into API calls of the underlying Java code, tightly coupling the transformation to their internal code structure.
- Eviware loadUI [11] had the most complicated input format out of the three. It used both binary and textual artefacts. Some of the textual artefacts were trees of Java classes, which would have to be generated and then packed together with the binaries.

The Grinder was selected among the available tools, as its input format was the easiest to generate and provided the most flexibility. In addition, it is easy to scale up depending on the testing requirements. It can launch several processes that spawn a certain number of threads which will repeatedly run the test and optionally distribute work over several machines: one of them provides a graphical console and acts as the master, and the rest are agents that manage a set of worker processes.

6.5.2 Model extraction

Since WSDL documents are declarative and language-independent descriptions of the WS, they were originally intended to be used as design models. After transforming automatically the XML Schema description of the WSDL document format into a regular

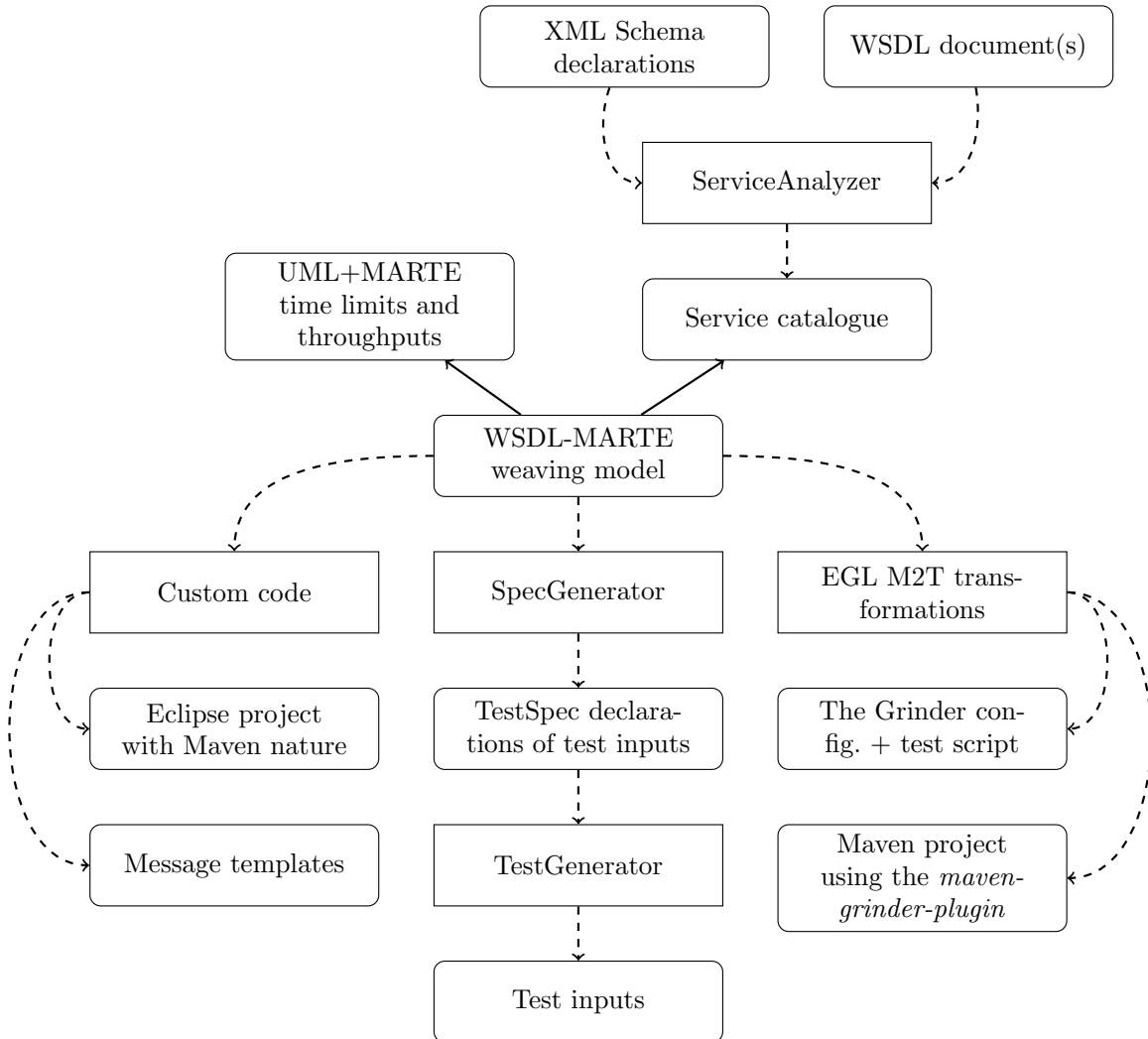


Figure 6.10. Instance of the above approach for WSDL-based Web Services. In comparison with the approach specifically targeted for Java, this approach requires integrating several technologies, such as a build automation tool (Maven), three custom WS-oriented tools (ServiceAnalyzer, SpecGenerator and TestGenerator) and a dedicated performance testing tool (The Grinder), among others.

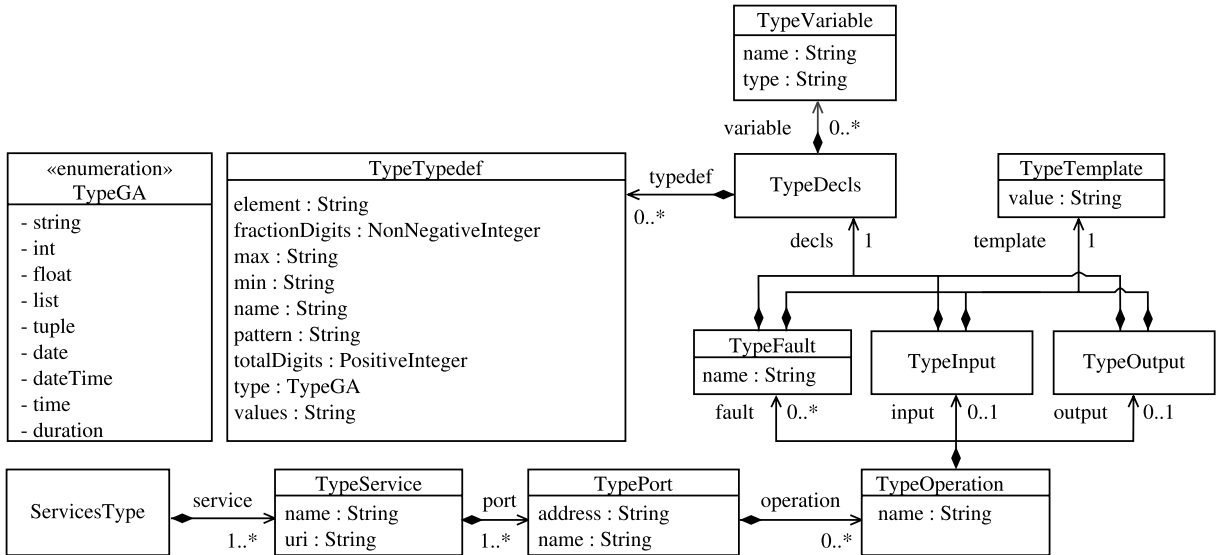


Figure 6.11. ServiceAnalyzer service catalogue metamodel

ECore metamodel [20], WSDL documents would be loaded as regular Eclipse Modeling Framework models, reusing most of the technologies mentioned in Section 6.4.

In practice, however, WSDL documents are too complex to be used as-is for model weaving and model transformation. WSDL documents can be divided across multiple files and machines and combine type and message declarations in the WSDL and XML Schema formats. In addition, XML Schema and WSDL are highly flexible, allowing many possibilities that may or may not be implemented by vendors. This has led to the definition of specifications such as the Web Services Interoperability Basic Profile (WS-I BP) [21], which restricts these standards to a consistent subset that is well-implemented across vendors.

Therefore, it was decided to extract models from the WSDL documents themselves using a new custom tool, ServiceAnalyzer [12]. ServiceAnalyzer produces a “service catalogue” from a set of local or remote WSDL documents that conform to the WS-I BP. Service catalogues can be loaded as an EMF model by using their XML Schema definition, as originally intended for WSDL.

The service catalogue metamodel is shown in Figure 6.11. Models are instances of *ServiceType*, which contains a set of *TypeServices* with their own *TypePorts*. Each *TypePort* has a collection of *TypeOperations* that may have an input, an output, and/or several fault messages. Message descriptions are divided into a *TypeTemplate* containing an Apache Velocity [1] template, and a *TypeDecls* that declares the variables used within the Velocity template. Variables may belong to one of the predefined types in *TypeGA*, which are based on the XML Schema primitive types, or they may belong to a custom type defined with a *TypeTypedef*.

Services store their names and namespace URIs, ports store their names and the URLs they are listening at, and operations and faults store their names. Type definitions must specify at least a name and a base type, but they usually specify additional restrictions such as a pattern based on a regular expression (*pattern*), minimum or maximum values (*min* or *max*) or a set of accepted *values*, among others.

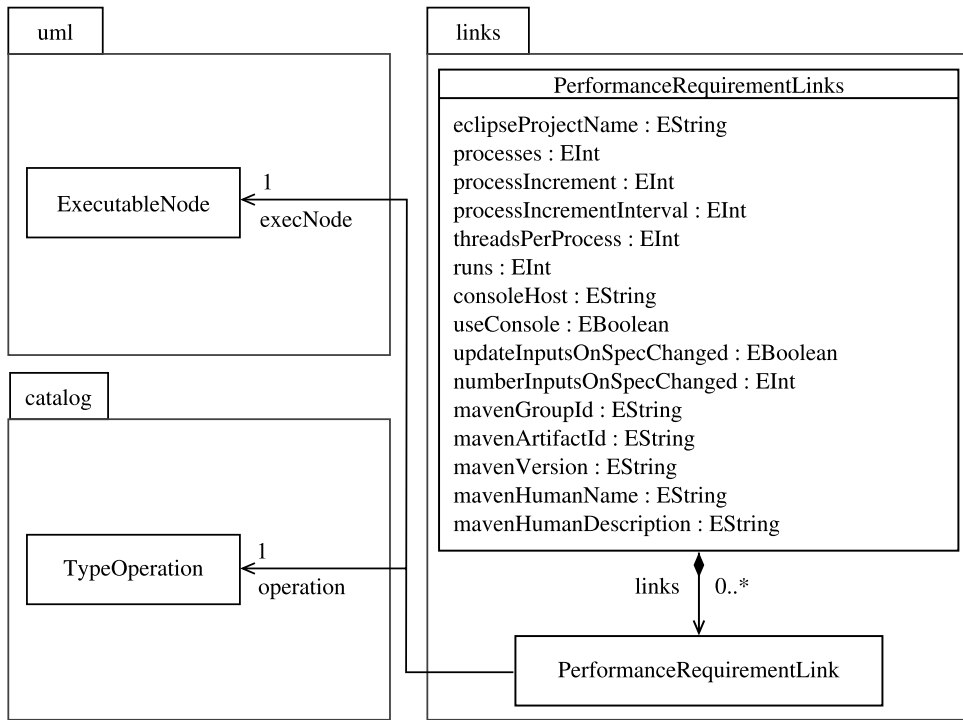


Figure 6.12. ServiceAnalyzer-MARTE weaving metamodel

6.5.3 Weaving metamodel

The weaving model needs to relate the *ExecutableNodes* in the UML activity diagram with the *TypeOperations* in the ServiceAnalyzer service catalogue. For instance, a developer might want to ensure that every invocation of the *evaluate* operation of the *Order* service finishes within a certain time while handling a certain number of requests per second.

The weaving metamodel is shown in Figure 6.12. It is quite similar to that in Figure 6.11, but the global options in the *PerformanceRequirementLinks* class have been changed to reflect the target technologies for this transformation:

- *eclipseProjectName* is the name of the Eclipse project which will be generated by the transformer. By default, it is set to “performance.tests”.
- The attributes ranging from *process* to *useConsole* are directly mapped to the configuration options of The Grinder with the same name.

process is the number of worker processes that will be used by each agent, starting from 1 and increasing by *processIncrement* every *processIncrementInterval* milliseconds (by default, by 1 every second). Each worker process will spawn as many as *threadsPerProcess* threads and repeat the tests the number of times indicated in *runs*. If *useConsole* is set to “true”, the console process at *consoleHost* will distribute work over the agents connected to it.

- The rest of the attributes can be used to customise the metadata of the Maven project that is generated by the transformer.

As for the options for the testing process itself, *updateInputsOnSpecChanged* and *numberInputsOnSpecChanged* indicate if the test inputs should be updated when the *.spec*

Listing 6.4 Apache Velocity template extracted from the ServiceAnalyzer catalog for producing the test input message

```
<w:evaluate xmlns:w="http://ws.sodmt.uca.es/">
  #foreach($V1 in $evaluate)
    <newOrder>
      #foreach($V2 in $V1)
        <articleQuantities>
          <articleID>${V2.get(0)}</articleID>
          #foreach($V3 in $V2.get(1))
            <quantity>${V3}</quantity>
          #end
        </articleQuantities>
      #end
    </newOrder>
  #end
</w:evaluate>
```

Listing 6.5 TestGenerator .spec extracted from the ServiceAnalyzer catalog describing the inputs for the template in Listing 6.4

```
typedef int (min=0, max=100) TArtID;
typedef float (min=0.01, max=2000) TPrice;
typedef list (element=TPrice, min=1, max=1) TL_float;
typedef tuple (element={TArtID, TL_float}) TArticleQtys;
typedef list (element=TArticleQtys, min=0) TOrder;
typedef list (element=TOrder, min=1, max=1) TEvaluate;
TEvaluate evaluate;
```

file describing their format changes, and how many should be generated each time.

The default options should be good enough for most users. The next sections will mention again some of them as the following steps in the generation process are introduced.

6.5.4 Test data generation

In order to run performance tests, it is necessary to provide them with test inputs so they can exercise the WS appropriately. This can be quite difficult for WS, as their inputs are complex XML documents following the advanced XML Schema type system. The present approach divides the generation of the messages into three parts to make it more manageable: a Velocity template that produces the test input message, a TestSpec document that describes how the template inputs should be generated, and the file with the template inputs themselves. Users may customise any of these three parts to suit their needs.

First, the tools extract the appropriate Velocity templates and variable declarations from the ServiceAnalyzer service catalogue to separate files. Listing 6.4 shows an Apache Velocity

Listing 6.6 Template inputs produced by TestGenerator from the .spec in Listing 6.5

```
#set($evaluate = [
  [[[85, [1530.1414]], [3, [1652.419]], [50, [550.96515]]]],
  [[[92, [1682.8262]], [45, [1593.5898]]]],
  [[[79, [72.64899]], [22, [603.8968]], [8, [1278.9677]]]]
])
```

Listing 6.7 Example `grinder.properties` file with workload configuration parameters

```
grinder.processes=5
grinder.runs=100
grinder.processIncrement=1
grinder.processIncrementInterval=1000
```

template which can produce every valid request for an order evaluation service, according to its WSDL and XML Schema declarations. As a template language, the Velocity language is kept simple, providing only the most common programming constructs, such as conditionals (`#if`), loops over a list (`#foreach`), variable assignments (`#set`) or field references (`$var.field`). Velocity templates are expanded during test execution with the variables loaded into their *contexts*. This template produces a `<newOrder>` element for each item in `$evaluate`. In turn, the template produces a `<articleQuantities>` element for each item, with the appropriate article identifier and requested quantities.

Listing 6.5 shows the TestSpec document that was extracted from the same catalogue entry. The TestSpec language is implemented by the TestGenerator tool, also available from [12]. It is a simple domain-specific language (inspired on C declarations) which allows users to define new scalar, list and tuple types based on a set of primitive types inspired by the XML Schema type system. These new types can have additional constraints, such as having minimum or maximum values or lengths, adhering to a certain regular expression or having a certain number of digits. TestGenerator can apply a *strategy* on these declarations to produce an arbitrary number of tests and store them as Velocity templates: the default strategy produces uniformly distributed random values, but it may use other random distributions or combinatorial testing techniques if manually indicated.

The Velocity files produced by TestGenerator contain the input data to be fed to the message templates. They consist of a sequence of variable assignments in which every variable receives a list of values to be used within each test. Listing 6.6 shows three test cases that were produced from the `.spec` in Listing 6.5. For instance, the first test requests 1530.14 units of article `#85`, 1652.419 units of article `#3` and 550.965 units of article `#50`. Velocity was reused to store test data since it was more flexible than a simple table or spreadsheet, as it allowed for arbitrarily nested lists.

In the wild, WSDL declarations tend to be quite lax, allowing messages with no upper bound on their length or elements containing negative integers, even though they are not accepted. In these cases, users may want to customise the service catalogue before generating the `.spec` descriptions from it. This will change the values used for all tests of the modified operations. Alternatively, users may want to modify a single `.spec` file describing the inputs of a particular test. Users may also customise the message templates with additional logic, or provide manually designed input data instead of generating random inputs.

6.5.5 Test code generation

After weaving the service catalogue model with the MARTE model and producing some input data to exercise the Web Services, the next step is generating the test specification for The Grinder.

The Grinder requires generating two different files: a `.properties` file indicating several

Listing 6.8 Example Jython script for The Grinder with the contents of the performance test to be run by each simulated client

```

class TestRunner:
    def __call__(self):
        def invoke():
            response = HTTPRequest().POST(
                "http://localhost:8080/orders",
                "..._SOAP_message_...")
            stats = grinder.statistics.getForCurrentTest()
            stats.success = (response.statusCode != 200 and stats.time < 150)
            test = Test(1, "Query_order_by_ID").wrap(invoke)
            test()

```

Operation	Passed	Failed	Bytes/s	Mean resp. length
Close Order	60	0	66,590	332.95
Evaluate Order	60	0	64,333.33	321.67

Table 6.2. Example test metrics produced by Grinder Analyzer (overall results, throughput and message sizes)

parameters of the workload to be generated, and a Jython script with the test to be run by each simulated client. Listings 6.7 and 6.8 show simplified examples for these two files. These files are automatically generated using EGL.

The `.properties` file in Listing 6.7 indicates that 5 processes should each run the test 100 times, starting with 1 process and adding one more every 1000 milliseconds. On the other hand, the test itself consists of sending an appropriate SOAP message to a specific URL and checking that the response has the OK (200) HTTP status code and that it was received within 150 milliseconds. These values are extracted from the global options in the *PerformanceRequirementLinks* object of the model. *consoleHost* and *useConsole* are also used in the `.properties` file.

The actual generated Jython script is over 180 lines long and takes advantage of several language features to avoid code repetition. In addition to running the tests themselves, it can regenerate test data if the `.spec` files have been customised by the user since the last run. Every time a test is run, a set of input values is randomly selected from the available test data. This input data is used to generate the SOAP message from the message templates, invoke the service and check the non-functional attributes of the reply. One limitation with the current version of the scripts is they can only check maximum response times, unlike the approach in Section 6.4, which can handle averages, medians and percentiles.

6.5.6 Test infrastructure and report generation

The approach in Section 6.4 was straightforward: as it simply produced Java code based on the ContiPerf library, users would simply need to add ContiPerf to their development environments and then run the tests using standard tools. However, running the tests produced by this approach would require setting up TestGenerator, The Grinder, and Apache Velocity.

For this reason, the tools implement an additional EGL transformation that produces an

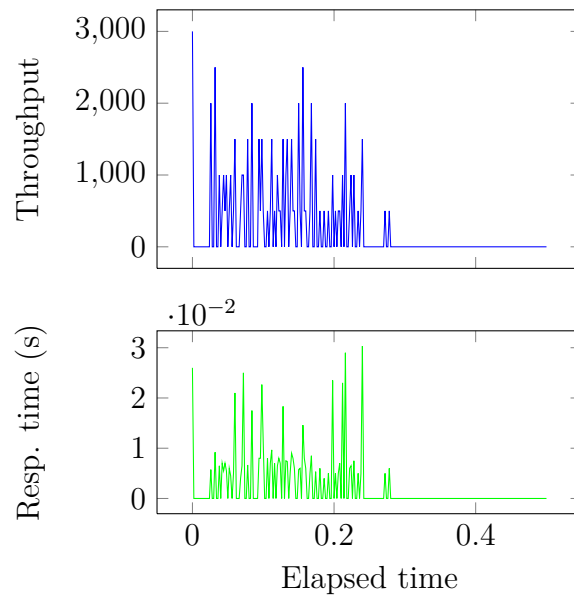


Figure 6.13. Example of an overall performance graph produced by Grinder Analyzer

Operation	Mean resp. time	Resp. time std. dev.	Mean time DNS	Mean time conn.	Mean time first byte
Close Order	14.35	15.62	0.00	0.13	13.18
Evaluate Order	8.98	6.01	0.00	0.37	5.93

Table 6.3. Example test metrics produced by Grinder Analyzer (timing information).
Times are measured in milliseconds.

Apache Maven [3] project description that automatically downloads all dependencies, runs the performance tests and produces test reports from the results. The Grinder is integrated through the open source plug-in available at [13]. Maven also enforces a standard directory layout for all the generated artefacts.

This infrastructure allows users to run the entire testing process with a single `mvn post-integration-test` command, which also invokes the Grinder Analyzer tool [5] on the raw logs to produce an HTML report including (but not limited to) the information shown in Figure 6.13 and Tables 6.2 and 6.3. On this example, the report shows that all tests passed and that the mean response time for the “Evaluate Order” service was 8.98 milliseconds. This short time is to be expected, as the tests in the example were run against local Web Services using an in-memory object-relational database.

6.6 Conclusion

This chapter has presented an overall approach for generating performance test artefacts from the abstract performance models produced by the inference algorithms in Chapter 5. To generate concrete test artefacts while keeping the abstract performance models separated from any design or implementation details, the approach links the performance model to a design or implementation model using an intermediate *weaving model*. If a design or implementation model is not available, it can be extracted from the existing code. The weaving model can be then optionally refined using a model-to-model transformation, and finally transformed into the performance test artefacts with a model-to-text transformation.

The general approach has been validated by applying it on two target technologies. Both approaches have been successfully implemented and are freely available under the open source Eclipse Public License (EPL) at [12].

The first application weaves JUnit test suites with MARTE models and converts all or some of their unit tests into performance test cases, using the ContiPerf library. The implementation model is extracted from the Java code implementing the test cases using the model discovery tool MoDisco [7], and the weaving model links the *ExecutableNodes* in the UML activity diagram to the Java tests in the MoDisco model.

The second application can generate performance test cases for any Web Service that is described using WSDL. It is independent of the language in which the Web Service has been implemented, as it is based on a special-purpose performance testing tool: The Grinder [4]. Users extract service catalogues from a set of WSDL documents and then weave the service operations in the catalogue with the MARTE models. The service catalogues also include message templates and template variable declarations, which are used to generate a set of initial template inputs. Users are able to manually customise the service catalogue, the message templates, the template variable declarations and the template inputs. In addition to the inputs, a set of automated model-to-text transformation produces the Jython code and the configuration file required by The Grinder, and a Maven project description that enables users to run the tests and produce reports with a single command (as those in Section 6.5.6).

While these applications show that the overall approach can be reused for different target technologies, they do currently share several limitations. Transformations only know the part of the system under test that is strictly needed to generate the tests. For this reason, users will need to manually customise the tests if they need to restore the

state of the system after a performance test, a memory violation or an aborted operating system process, or if they want to set up specific mockups for specific subsystems in the application. Nevertheless, the transformations could be extended with “hooks” where this kind of logic could be placed, and keeping those “hooks” from being overwritten if the tests are generated. This is already being done in the Java approach: the generated test suites use EGL protected areas that are preserved when the files are regenerated.

References

- [1] Apache Software Foundation. Apache Velocity Project homepage, November 2010. URL <http://velocity.apache.org>. Last checked: November 6th, 2013. 6.16
- [2] Apache Software Foundation. Apache JMeter, November 2013. URL <http://jakarta.apache.org/jmeter/>. Last checked: November 6th, 2013. 6.14
- [3] Apache Software Foundation. Apache Maven homepage, January 2013. URL <http://maven.apache.org>. Last checked: November 6th, 2013. 6.22
- [4] P. Aston and C. Fitzgerald. The Grinder, a Java Load Testing Framework, 2012. URL <http://grinder.sourceforge.net/>. Last checked: November 6th, 2013. 6.14, 6.22
- [5] T. Bear. Grinder Analyzer homepage, July 2012. URL <http://track.sourceforge.net/>. Last checked: November 6th, 2013. 6.22
- [6] V. Bergmann. ContiPerf 2, September 2011. URL <http://databene.org/contiperf.html>. Last checked: November 6th, 2013. 6.8
- [7] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot. MoDisco: a generic and extensible framework for model driven reverse engineering. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, pages 173–174, Antwerp, Belgium, September 2010. 6.10, 6.22
- [8] M. D. Del Fabro, J. Bézivin, and P. Valduriez. Weaving models with the eclipse AMW plugin. In *Proceedings of the 2006 Eclipse Modeling Symposium, Eclipse Summit Europe*, Esslingen, Germany, October 2006. 6.8
- [9] Eclipse Foundation. Homepage of the mdt uml2 project, June 2013. URL <http://www.eclipse.org/modeling/mdt/?project=uml2>. Last checked: November 6th, 2013. 6.6
- [10] Eclipse Foundation. Homepage of the papyrus project, June 2013. URL <http://www.eclipse.org/papyrus/>. Last checked: November 6th, 2013. 6.6
- [11] eviware.com. loadUI homepage, 2013. URL <http://www.loadui.org/>. Last checked: November 6th, 2013. 6.14
- [12] A. García-Domínguez. Homepage of the SODM+T project, April 2013. URL <https://neptuno.uca.es/redmine/projects/sodmt>. 6.8, 6.16, 6.19, 6.22

- [13] G. Iacono and F. Muñoz-Castillo. grinder-maven-plugin homepage, June 2013. URL <http://code.google.com/p/grinder-maven-plugin/>. 6.22
- [14] Java.net. JAX-WS reference implementation, November 2013. URL <http://jax-ws.java.net/>. Last checked: November 6th, 2013. 6.13
- [15] D. S. Kolovos. Epsilon ModelLink, 2012. URL <http://eclipse.org/gmt/epsilon/doc/modelink/>. Last checked: November 6th, 2013. 6.8
- [16] D. S. Kolovos, L. M. Rose, R. F. Paige, and A. García-Domínguez. The Epsilon book, 2013. URL <http://dev.eclipse.org/svnroot/modeling/org.eclipse.epsilon/trunk/doc/org.eclipse.epsilon.book/EpsilonBook.pdf>. Last checked: November 6th, 2013. 6.6, 6.8
- [17] Object Management Group. UML Profile for Schedulability, Performance, and Time (SPTP) 1.1, January 2005. URL <http://www.omg.org/spec/SPTP/1.1/>. Last checked: November 6th, 2013. 6.1
- [18] Object Management Group. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QFTP) 1.1, April 2008. URL <http://www.omg.org/spec/QFTP/1.1/>. Last checked: November 6th, 2013. 6.1
- [19] Object Management Group. UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) 1.1, June 2011. URL <http://www.omg.org/spec/MARTE/1.1/>. Last checked: November 6th, 2013. 6.1, 6.2
- [20] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, second edition, December 2008. ISBN 978-0321331885. 6.16
- [21] Web Services Interoperability Organization. Basic profile - version 1.1 (Final), August 2004. URL <http://www.ws-i.org/Profiles/BasicProfile-1.1.html>. Last checked: November 6th, 2013. 6.16
- [22] World Wide Web Consortium. Web Services Description Language (WSDL) Version 2.0 Part 0: Primer, June 2007. URL <http://www.w3.org/TR/wsdl20-primer>. Last checked: November 6th, 2013. 6.13

7

Case study

The previous chapters have revised the current state of the art and presented an extended version of the SODM methodology with improved models that can include performance annotations, from which performance requirements and tests for specific Web Services (WS) can be derived.

This chapter is dedicated to providing a complete end-to-end example of the resulting SODM+T methodology in a manufacturing context, from the high-level description of the manufacturing enterprise to the generation of the actual performance tests of one of the Web Services involved.

7.1 Overall description

7.1.1 Enterprise profile

Keraben S.A. is a manufacturing enterprise that designs, manufactures and sells various kinds of pavements and ceramic tiles, using both make-to-order and make-to-stock scheduling. Originally created as Gres de Nules-Keraben in 1974 and later renamed to Keraben in 1984, it is nowadays one of the leading companies in the ceramic tile business worldwide.

The main business goal for Keraben is achieving excellence in design and manufacturing quality through steady innovation and communication with customers while preserving the natural environment. Keraben S.A. itself is located at kilometer 44.3 of the Valencia-Barcelona highway in Nules (Castellón) and the installations span over 250 000 square meters, which enable the firm to annually produce over 11 million m^2 of ceramic tiles.

Nowadays, Keraben accepts orders from over 150 countries. Some of its products include traditional pavements and tilings made out of white and red clay, rectified pavements, technical porcelains and enamellings. Keraben continues to research and improve its technology from collection to collection. Keraben has received numerous awards in recognition of these efforts, such as the Príncipe Felipe Award for Business Excellence in the categories of Industrial Quality and Business Competitiveness, the Nova Award from the Valencia regional government for its business record, the “The Economist — Spencer Stuart” Business Ethics Award. Keraben also obtained an ISO 9000 certification on Quality Management for their ceramic tiles business, and a BS OHSAS 18001 certification on Occupational Health and Safety Management Systems.

The Keraben Group has over 1 000 employees working over several production facilities, offices, laboratories and stores in the province of Castellón. Its products span six different trademarks, each covering a particular market segment:

- Keraben: the main brand, focusing on ceramic wall and floor tiles, decorative pieces and floor tiles for high traffic areas.
- Metropol: ceramic tiles that innovate through design, shapes and colors.
- Atenea: ceramic tiles that suggest natural spaces.
- Acquabella: hydromassage and shower solutions.
- Keratrim: wall decorations and floor coverings.
- Kerafrit: design and manufacture of ceramic glazes.

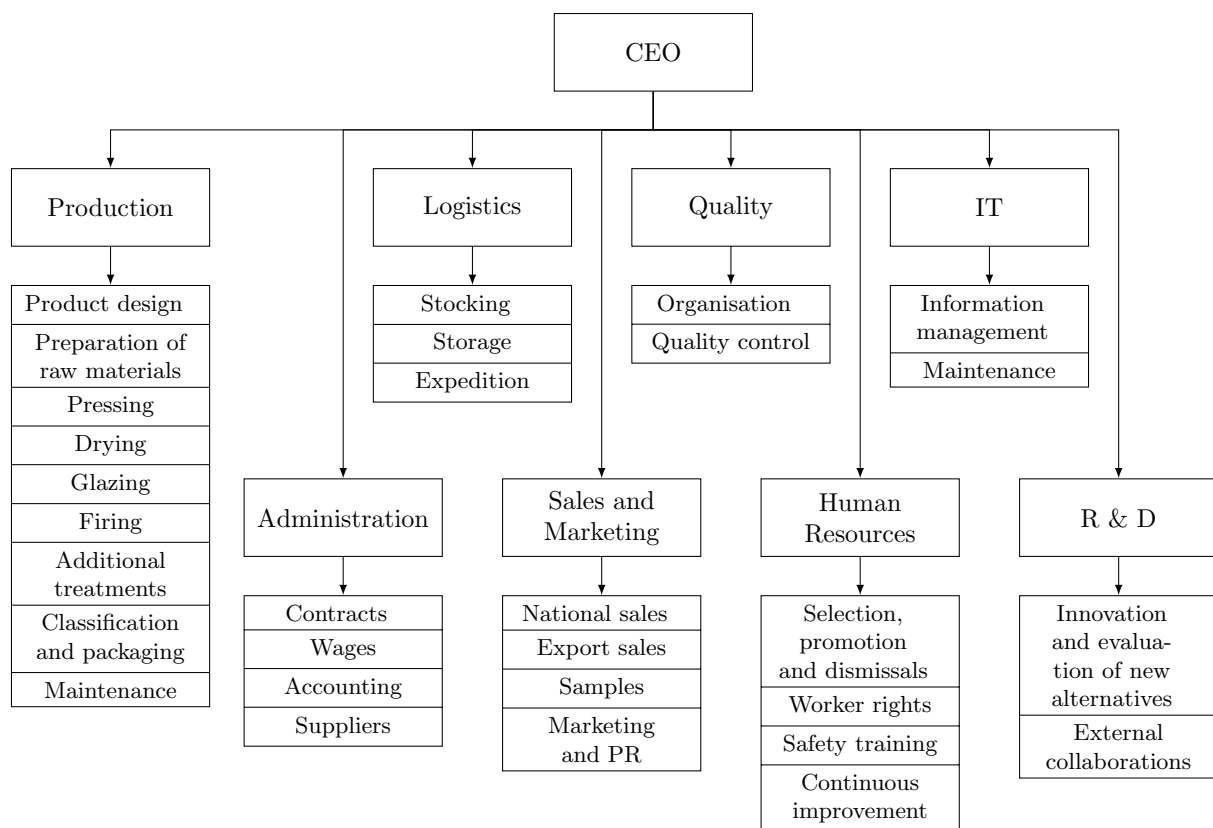


Figure 7.1. Organisational chart of Keraben

In addition, the Keraben Group has its own range of stores to sell directly to individual customers, and has sales representatives all over the world, with offices in Russia, the Czech Republic, Mexico and the United States.

Among these properties, the present case study will focus on the production facilities for the Keraben brand of ceramic products. Figure 7.1 shows the relevant organisational chart for this part of the Keraben group.

7.1.2 Manufacturing process for porcelain stoneware

This case study will focus on a particular manufacturing process within Keraben: the production of porcelain stoneware. The process starts with the extraction of clays and other raw materials from a quarry and finishes with the packaging of the tiles into pallets, ready to be distributed to the customers. Figure 7.2 summarises the steps involved:

1. Preparation of the *raw materials*, which are selected depending on the desired composition of the clay. These include white clay, quartz, feldspar and kaolin.
2. *Grinding* breaks up the raw grains, reducing their diameter from millimetres to micrometres.
3. *Atomisation* is a drying process in which hot air at 500 Celsius degrees blows over the powder suspended in fine water drops, producing a solid mass with a reduced water content.

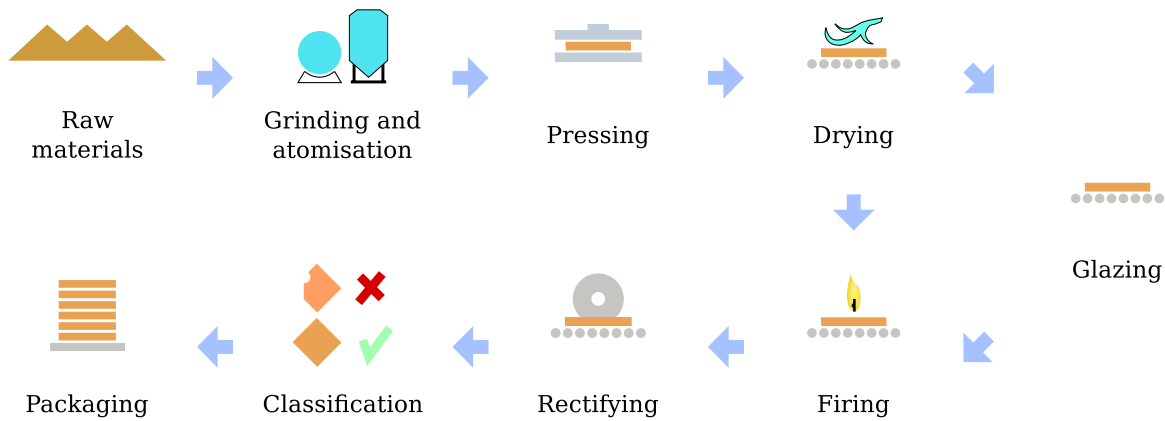


Figure 7.2. Manufacturing process for porcelain stoneware: pieces are produced both in glazed and unglazed varieties

4. There are several *pressing* processes: the most common is mechanical *dry pressing*, in which the workpieces have between 5% and 7% of water content.
5. Porcelain stoneware goes through an additional *drying* process that reduces the water content to 0.2–0.5% in order to ensure the success of the firing step and increase the mechanical resistance of the workpieces.
6. In some cases, the *glazing* step covers the workpiece with several layers of vitreous coating with thicknesses that range between 75 and 500 μm . These glazings grant various technical and aesthetic properties to the workpiece, such as impermeability, easy cleaning, colour, gloss, surface finish, chemical resistance or mechanical resistance.
7. *Firing* is a key step when manufacturing porcelain stoneware, as it requires longer times and higher temperatures than with regular glazed tiles.
8. After the tile exits the kiln, it is *rectified* in order to obtain a piece with straighter edges and precise dimensions.
9. *Classification and packaging* is the last step, in which tiles are visually and automatically inspected. Visual inspection checks for colour variations and superficial defects. Automatic inspection is used to sort the pieces by calibre.

7.1.3 Manufacturing facilities for porcelain stoneware

Keraben has eight production lines for ceramic pavements with four single layer kilns: two of these lines are dedicated to porcelain stoneware. The firm has two atomisers and six mills. Boxes with the green (raw) or fired material are transported using Automated Guided Vehicles (AGVs), with box storage areas at the entrance and exit of every kiln.

After each tile is manufactured, it is classified by quality and packaged into boxes and stacked in pallets, which are then wrapped and prepared for their expedition. Pallets wait to be expedited in a reserved area of the firm.

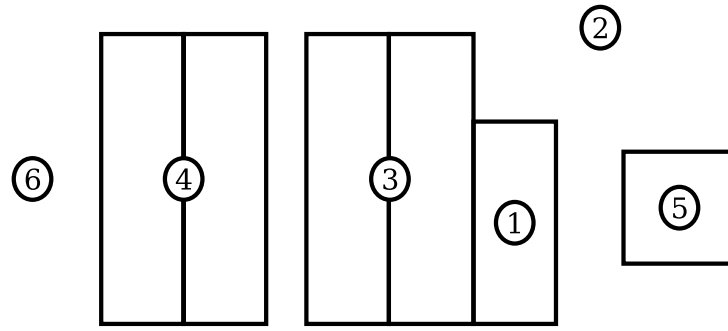


Figure 7.3. Map of the Keraben manufacturing firm: ① are the rectification lines, ② refers to the storage areas for finished products and for products to be rectified, ③ and ④ contain respectively the production lines for pavement and coverings, ⑤ is an office building and ⑥ is another storage area for the finished products.

Figure 7.3 shows a simplified plan of the firm, and Figure 7.4 zooms into the part dedicated to manufacturing porcelain stoneware tile pavements, which will be the main focus of this case study.

7.1.4 Providers

Keraben acquires raw materials, packaging supplies and furniture for their galleries and stores through several suppliers. At the moment of the present case study, the suppliers that participate in the Extended Enterprise (EE) that includes Keraben were:

- Kerafrit provides the glazings, frits and ceramic colours.
- Sibelco Hispania provides the white clay and kaolin.
- Guzmán Global provides the quartz and feldspar.
- Macer S.L. provides and maintains the pressing molds.
- Tarozzi Ibérica provides technical support for most of the machinery.
- SACMI IBÉRICA, S.A. provides technical support for the kilns.
- Cartonajes la Plana provides the packaging supplies.
- EMAT S.L. provides the ceramic showcases for the sample galleries.

7.1.5 Information and material flows

Figure 7.5 summarises the main information and material flows that take part within Keraben at an abstract level. White boxes represent the various stakeholders that provide information or material, grey boxes are the main tasks that take place within Keraben and folders are the information and material stores (normally, information systems and physical storage areas, respectively) that are used in some of the tasks.

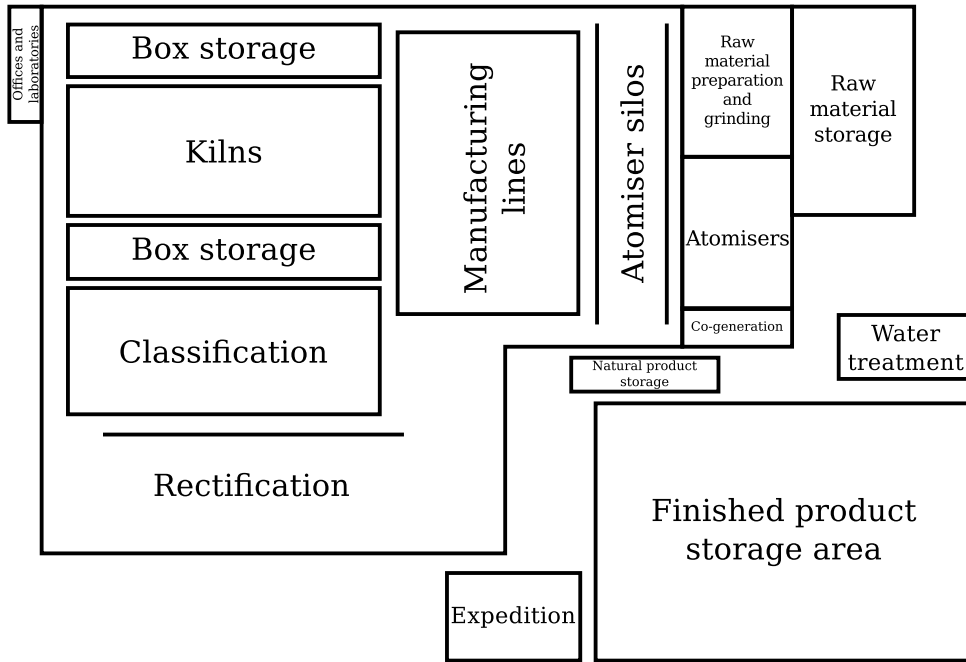


Figure 7.4. Map of the Keraben porcelain stoneware manufacturing plant ((1) and (3) from Figure 7.3).

Keraben periodically refreshes its product lines (task 1) by considering samples of the available materials from its suppliers, and internal samples produced by the design department. These product lines are collected into a catalogue: the demand for each item in the catalogue is estimated (task 2) using information from current bulk orders, recent market studies and previous sales figures. The demand is then adjusted by the management of the firm according to their experience.

The estimated demand is used in task 3 to produce a high-level production plan, which is again adjusted according to production capacity and used to produce a bill of materials for every interval in the plan. Task 4 fleshes out the production plan by using priorities and machine availability in order to generate a detailed production schedule. At the same time, task 5 ensures that the required stock is available at the beginning of every week in the production plan.

Finally, the production schedules are used to drive the machinery and transform the raw materials into finished products (task 6), which are later sold to the make-to-order and make-to-stock customers (task 7).

7.2 Computation-independent models

The previous section introduced Keraben S.A., a ceramic tile and decoration manufacturing firm, and provided a more detailed description of the manufacturing firm that produces its Keraben tiles and more specifically, the manufacturing process for its stoneware porcelain tiles.

From this section on, the extended SODM+T methodology will be used on a subset of the functionality required for the stoneware porcelain tile manufacturing firm. SODM+T inherits the top-down approach inherited from SODM (§4.3 and [4]), which is based on

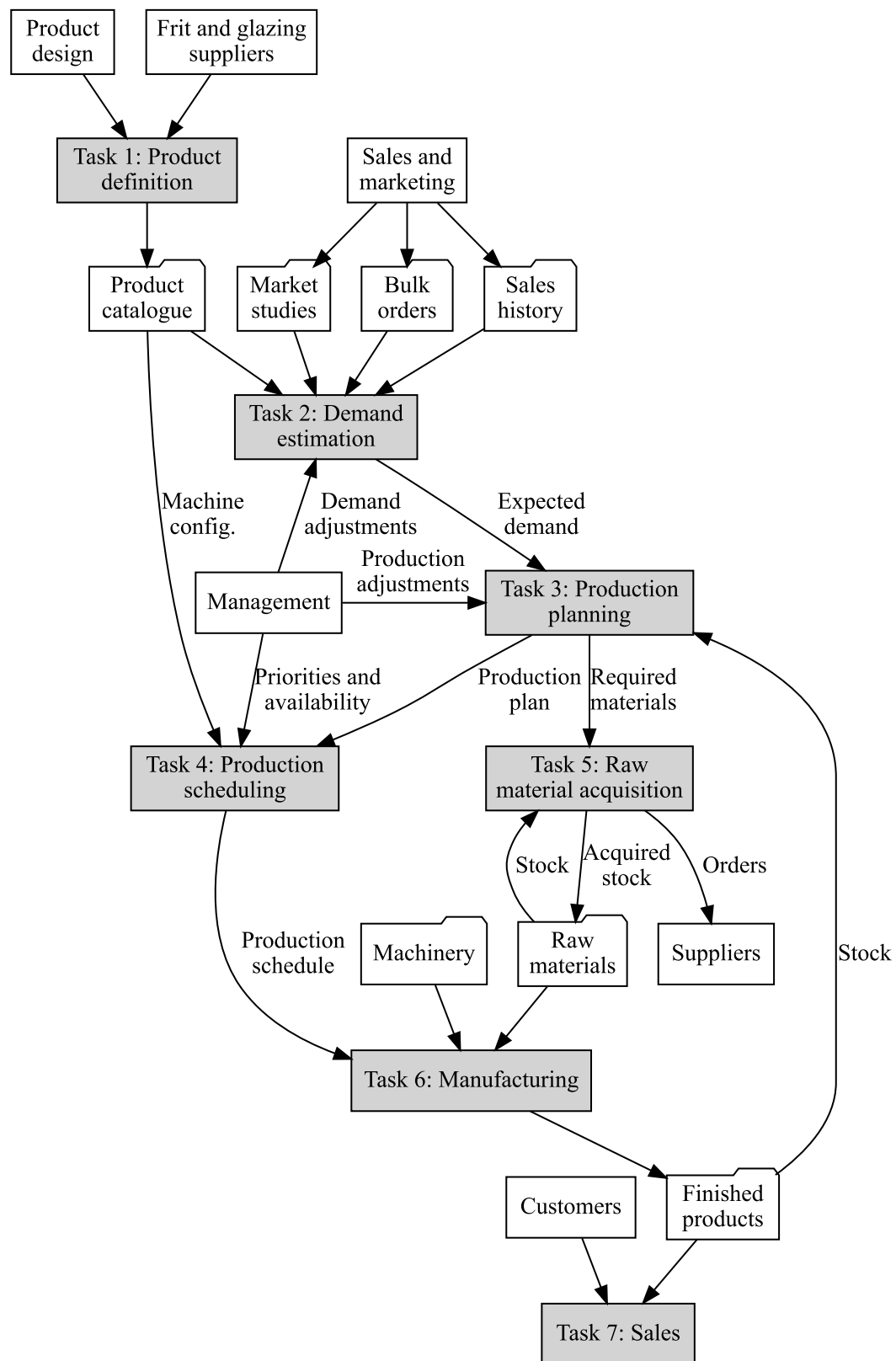


Figure 7.5. Information and material flows within Keraben: white boxes represent participants, grey boxes represent tasks and folders represent information or material stores.

the OMG MDA® approach [13]. Therefore, the first step is to develop the computation-independent models that describe the environment in which the system will operate and the goals that it must support.

7.2.1 Value models

Figure 7.6 shows the Gordijn value model for Keraben. The value model shows the value exchanges that take part in every step of the production process at Keraben. There are clearly two kinds of value exchanges in the model: while external suppliers and customers give or take value objects of some sort in exchange for money, exchanges with other units in Keraben consist mostly of passing information (reports and/or orders) between them.

The process is initiated from the start stimuli in the “Acquisition” and “Demand estimation” value activities in the “Customers” market segment and in the “Marketing” value actor, respectively. Large customers in the construction sector can place bulk or customised orders on demand, and the marketing department decides how much additional stock needs to be made to cover demand from individual homeowners. In some cases, production capacity concerns may require changes in how this demand is handled.

The “Planning” value action then talks with the logistics department and the raw material suppliers to obtain the required materials, passing control to the “Scheduling” action. “Scheduling” then refines the long-term production plan into several shorter-term schedules by taking into account the available machines and the required preventive maintenance. The “Manufacturing” action executes the schedule, repairing the machines in the event of an unexpected breakdown and transporting the finished goods as necessary. Some of the finished goods are directly packaged for the large customers, and the rest is used to stock showcases in stores. Finally, the “Sales” action sends the acquired goods to the customers.

7.2.2 Business process model

The next step in SODM+T (as defined by SODM) is creating the model for the high-level business process that needs to be supported by the system. Figure 7.7 shows a BPMN 2.0 process with the overall business model of Keraben.

The process takes into account the fact that Keraben uses both the make-to-stock (from the demand set by the Sales and Marketing department) and make-to-order manufacturing strategies. These two information sources are combined to produce a detailed demand for materials based on the definitions of the products themselves (grain types, tile composition, glazings and so on), the production rules and other consumables, such as packaging material or showcases.

At that point, the manufacturing process starts. If manufacturing completes without issues, both the customers and the sales department are informed and the products are shipped through the logistics company. Otherwise, the impact of the issue is analysed and then reported to the customers and the Sales and Marketing department. After correcting the issue (possibly communicating with the material or machine suppliers), manufacturing restarts.

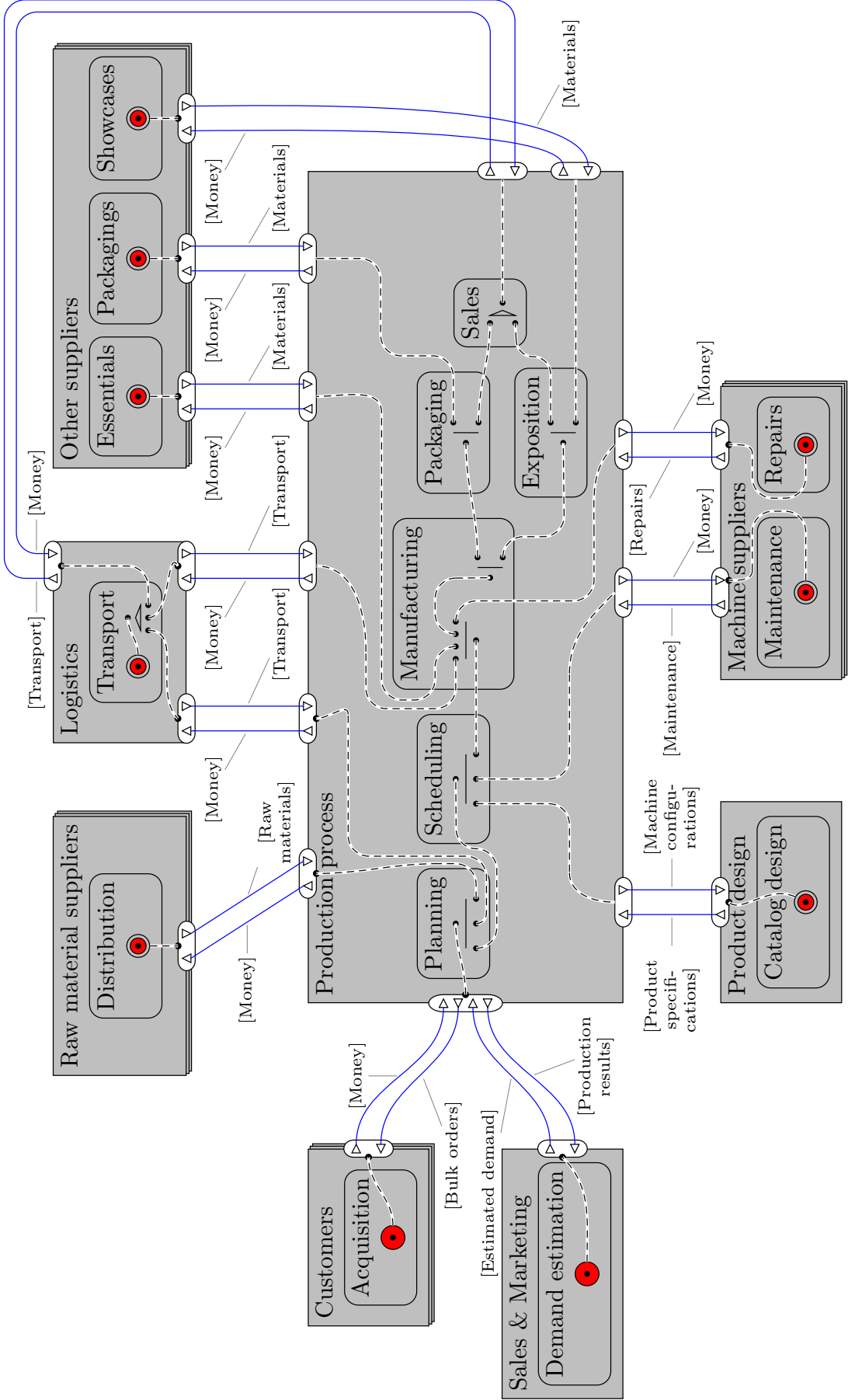


Figure 7.6. Gordijn value model for Keraben

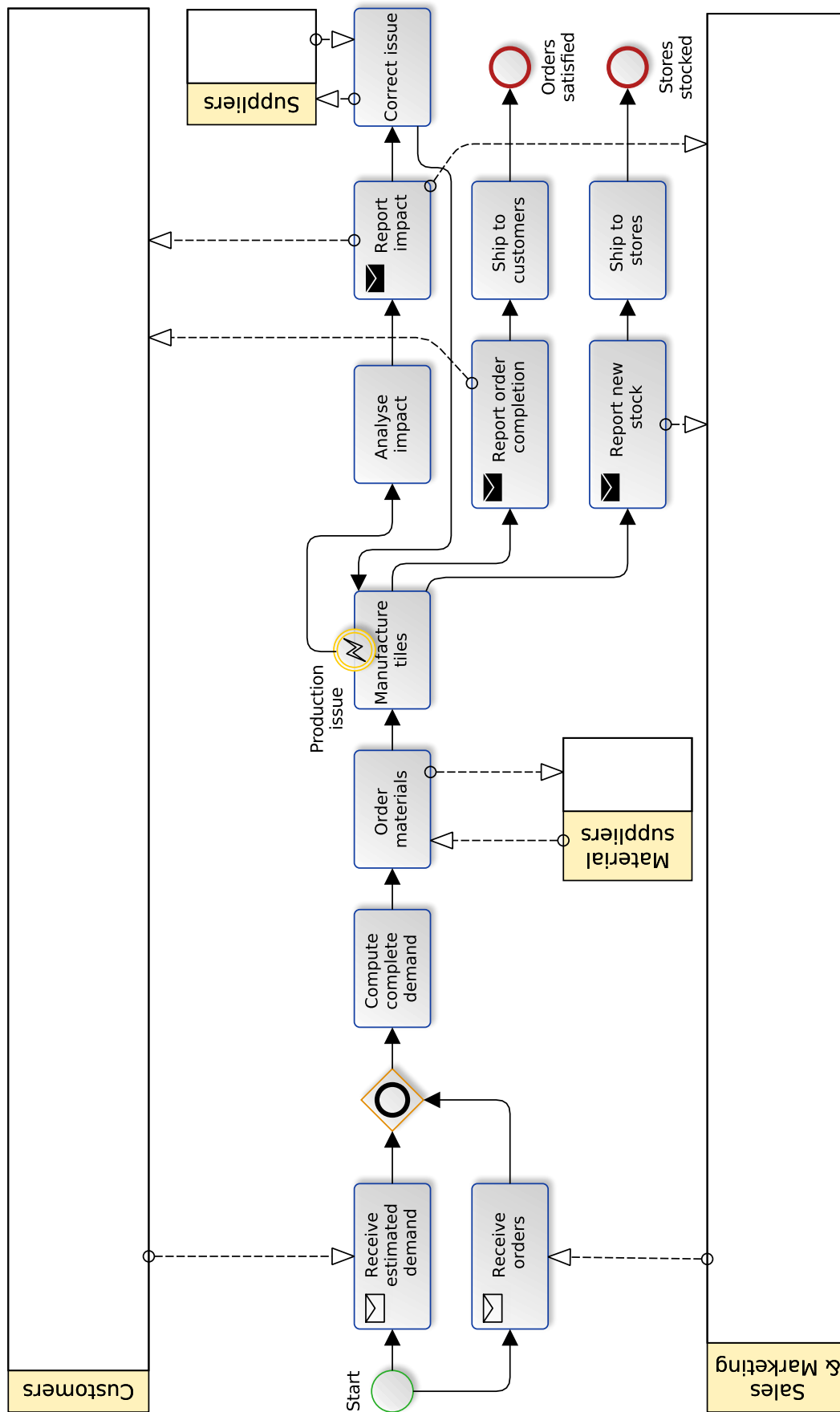


Figure 7.7. BPMN 2.0 diagram of the high-level business process for Keraben

7.2.3 Business service list

The last artefact at the CIM level is the business service list: a textual description of all the services that constitute a certain business and which satisfy a need from an end consumer.

According to SODM, the first step is finding all the end consumers by inspecting the value model in Figure 7.6 and looking for the value actors that provide the start stimuli for the value exchanges. These are the customers that buy the tiles from Keraben, and the Sales and Marketing department in Keraben itself. These two value actors represent the two manufacturing strategies in the company: make-to-order and make-to-stock, respectively. The system needs to be able to support both.

The second step is finding out the business services required by each end consumer. SODM suggests two sources for them: the value exchanges between the end consumers and the other actors, and the activities in the business process models which fulfil the needs of the end consumers.

From these two sources, it can be concluded that “Customer” requires the following business services:

- Order submission, from the “Receive orders” task in the BPMN process and the value exchange in the value model,
- Order status reporting, from the “Report order completion” task in the BPMN process, and
- Order issue notification and response from the “Report impact” task in the BPMN process, focusing on the message flows between the process and the “Customers” pool.

“Sales and Marketing” needs the following business services:

- Estimated demand submission, from the “Receive estimated demand” task in the BPMN process and the value exchange in the value model,
- Production status reporting, from the “Report order completion” task in the BPMN process and the value exchange in the value model, and
- Production issue notification and response service, from the “Report impact” task in the BPMN process, focusing on the message flows between the process and the “Sales & Marketing” pool.

The next steps will derive increasingly detailed descriptions of these business services that can be supported by a computer system.

7.3 Platform-independent models

The previous section derived a set of CIM models that described the value exchanges and the business services that the system would need to support. This section will produce the PIM models that prescribe what the system should do, without delving into specific implementation details.

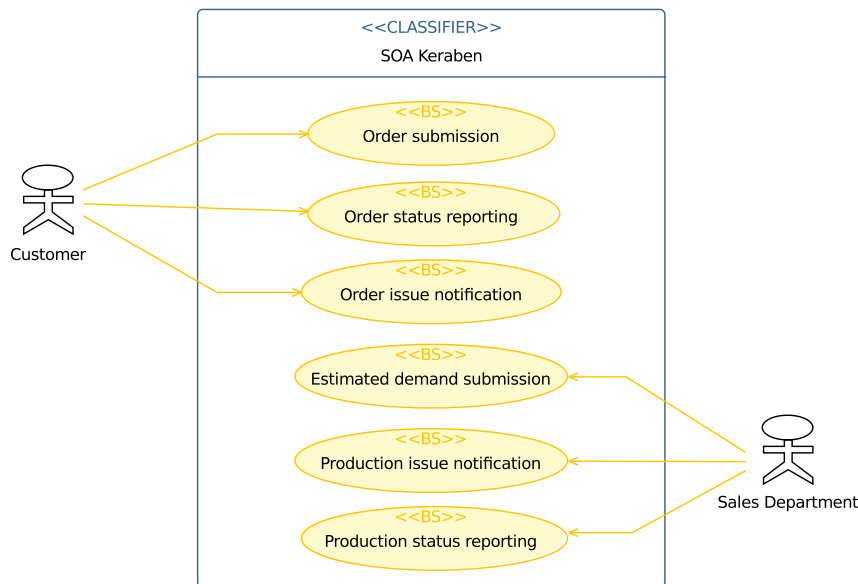


Figure 7.8. Use case model for Keraben

7.3.1 Use case model

The first step for the PIM models is to map the business services to use cases and limit the scope of the system to the desired subset. This section does not limit the scope of the system yet: the use case model in Figure 7.8 has all the business services above.

7.3.2 Extended use case models

The next step of the SODM methodology requires decomposing each use case in Figure 7.8 into its own extended use case model. In these extended use case models, high-level use cases may include simpler use cases, and a certain use case may have multiple variants.

Figure 7.9 shows the extended use case model for “Order submission”. Submitting an order requires checking it is feasible within the desired time frame and production parameters, ensuring that it can be shipped to the destination address and finalising the order after the customer has reviewed the order total and paid for it.

The “Order status reporting” extended use case model is shown in Figure 7.10. In this case, the order can be looked up by the username of the customer or by its unique ID. After finding the desired order, the customer can enquire the system about its current and previous statuses and the estimated time of arrival of the ordered goods.

The “Order issue notification” extended use case model in Figure 7.11 also looks up the order by customer username or order identifier. However, in this case the customer is informed of pending issues that may delay the order or may require changing some of the order details. Depending on the issues, the customer may decide to split the order into several smaller ones, cancel the order, change the ordered quantities of each product or change the deadline.

Figure 7.12 presents the extended use case model for “Estimated demand submission”. Periodically, staff from the Sales department specify the estimated demand of each product either month-by-month (for made-to-stock articles) or by deadlines (for made-to-order articles). These demands are then checked against the current production capabilities and

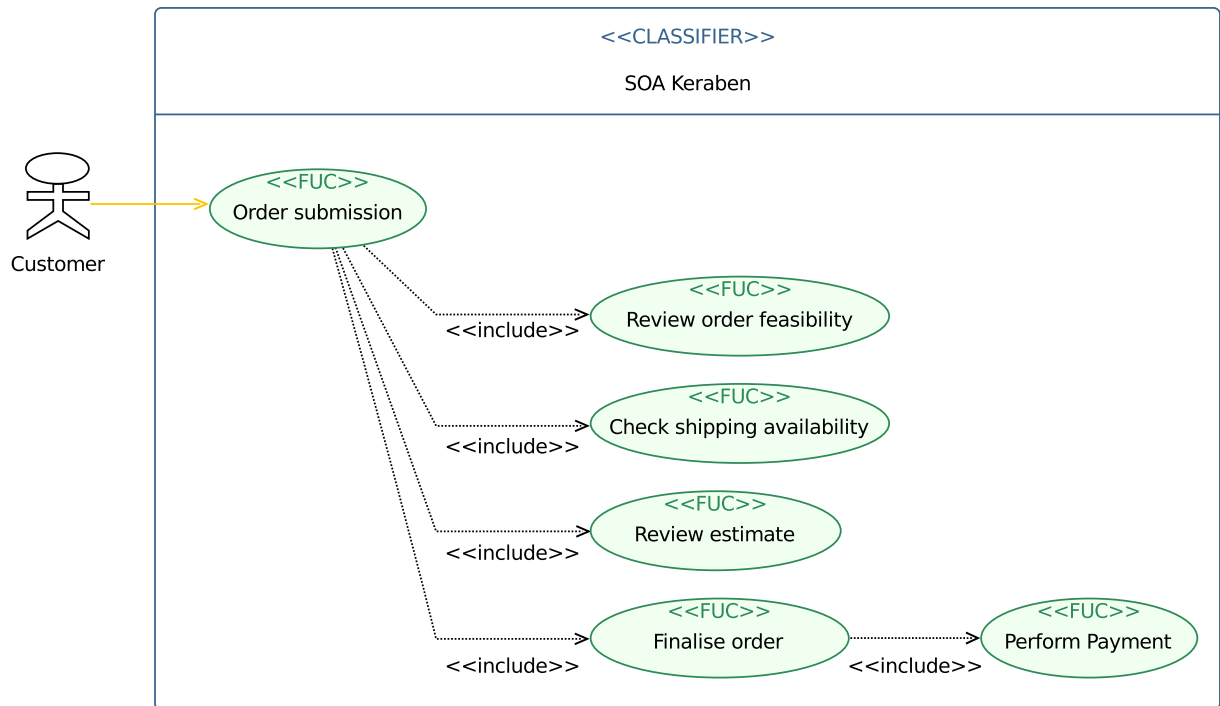


Figure 7.9. Extended use case model: “Order submission”

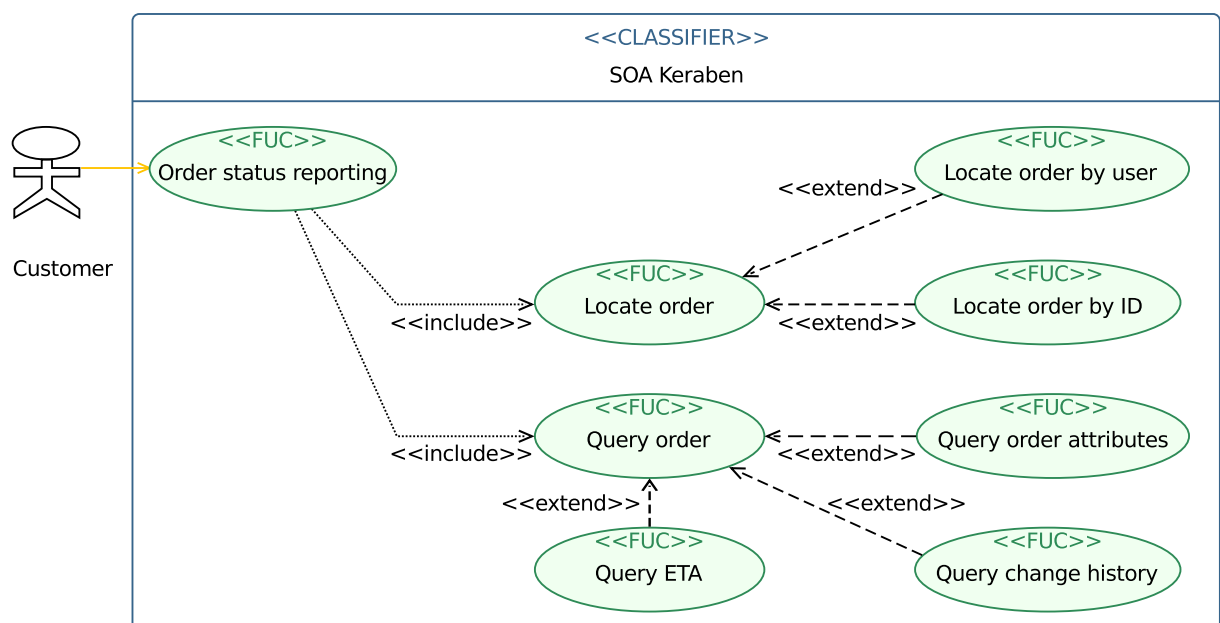


Figure 7.10. Extended use case model: “Order status reporting”

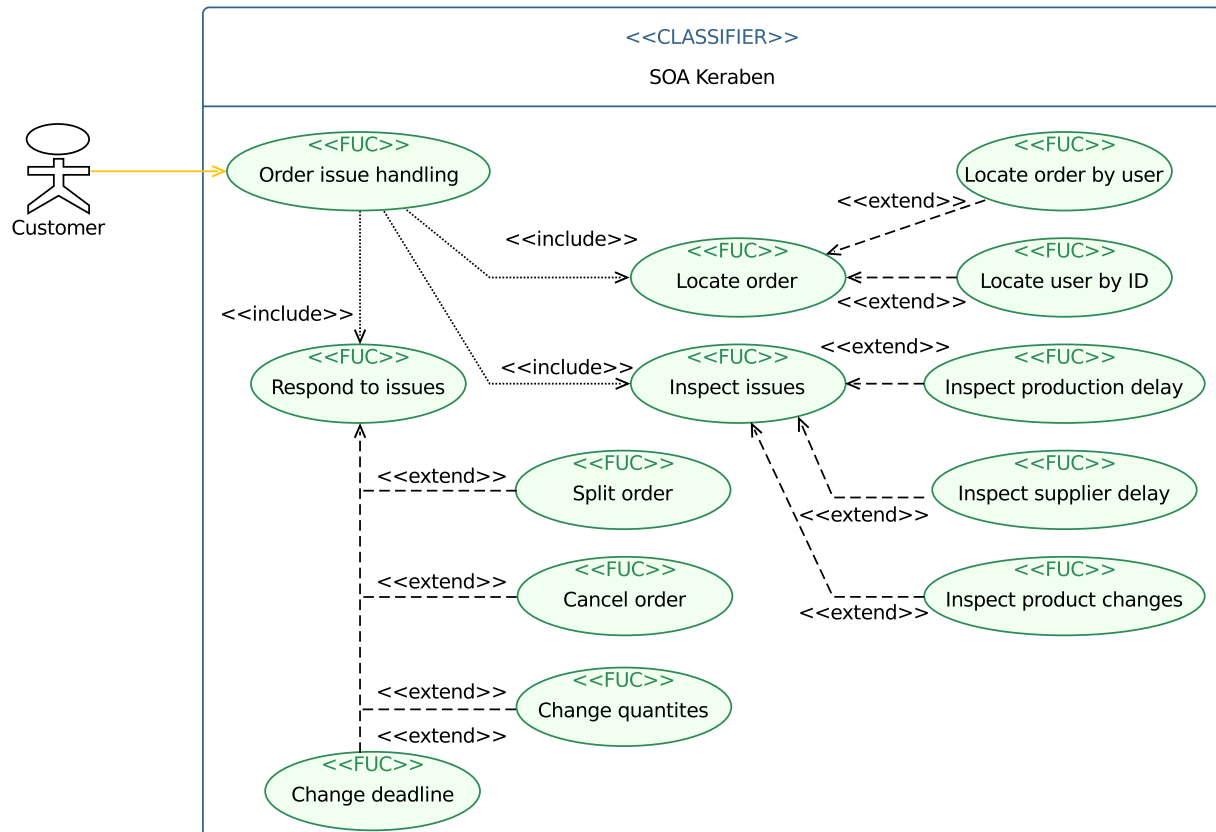


Figure 7.11. Extended use case model: "Order issue notification"

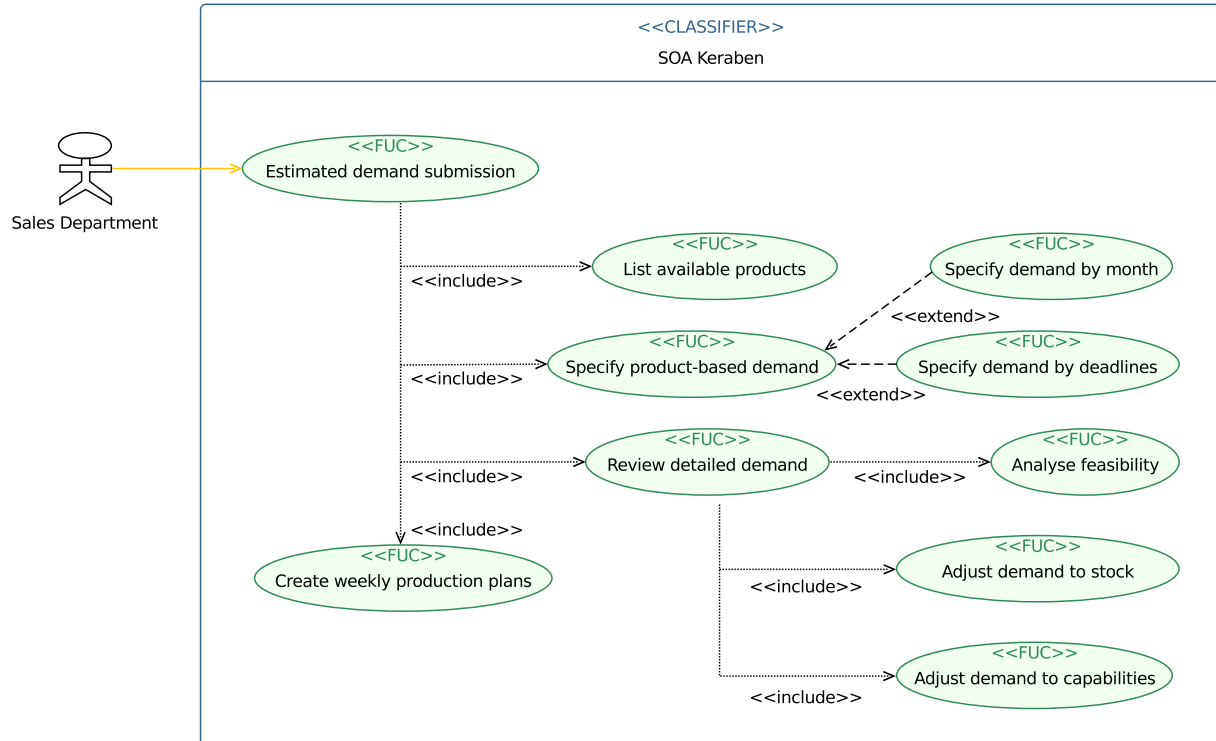


Figure 7.12. Extended use case model: "Estimated demand submission"

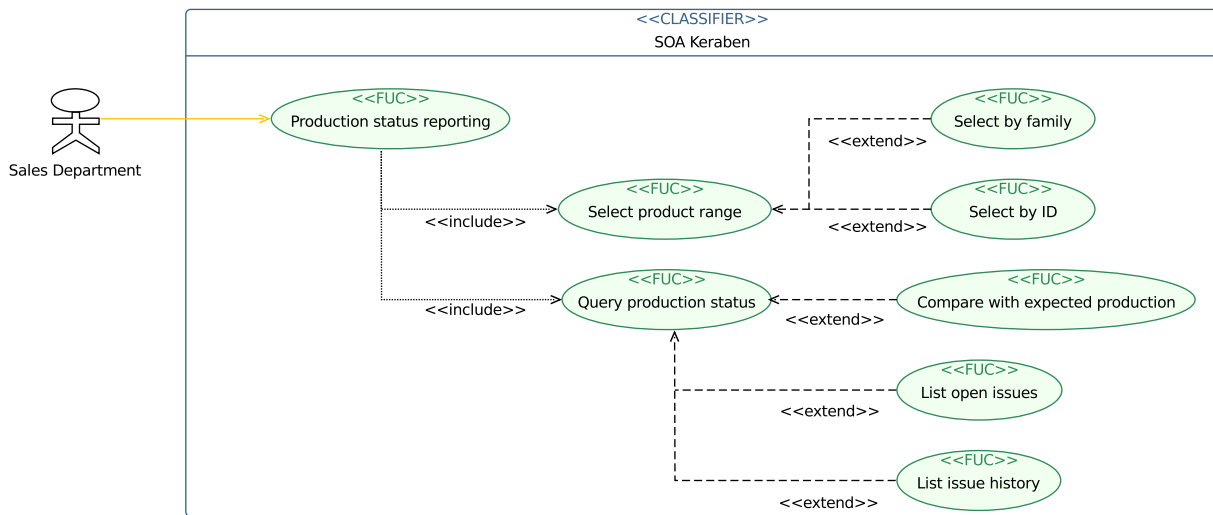


Figure 7.13. Extended use case model: “Production status reporting”

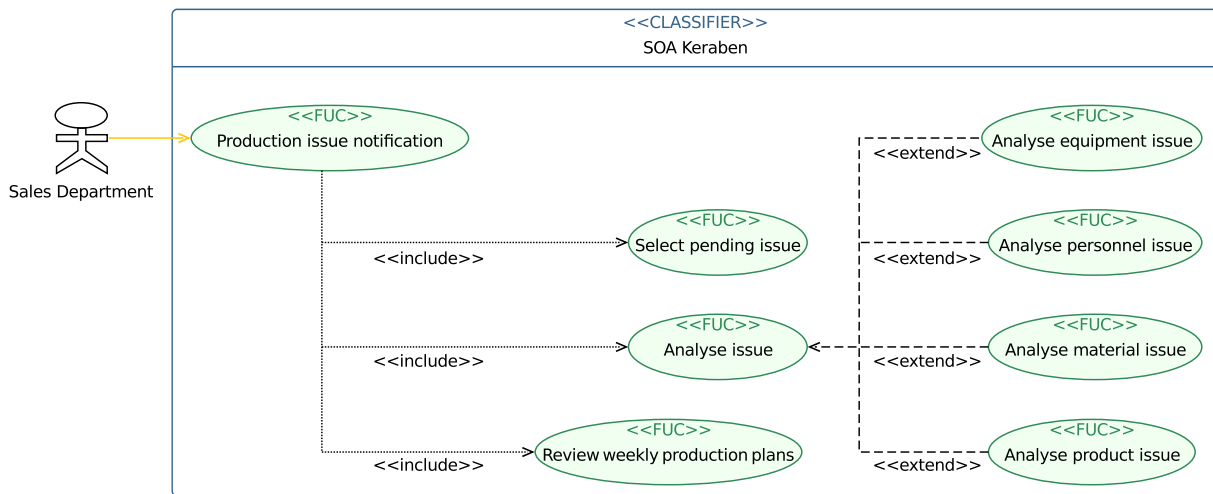


Figure 7.14. Extended use case model: “Production issue notification”

then readjusted. After the demands meet the available production capabilities, the weekly production plans are created from them.

The “Production status reporting” extended use case model in 7.13 is similar to the one for “Order status reporting”. However, instead of looking up orders and reporting on their status, it operates on the weekly production plans.

Finally, Figure 7.14 contains the extended use case model for “Production issue notification”. In this model, the Sales employee uses the system to analyse the pending manufacturing issues (e.g. equipment breakdowns, unexpected personnel leaves, material supply issues or product definitions issues) and then review the existing production plans after negotiating with affected customers.

7.3.3 Service process models

The above extended use case models divide the high-level use cases into finer steps that would be easier to implement, but do not specify the order in which they should be

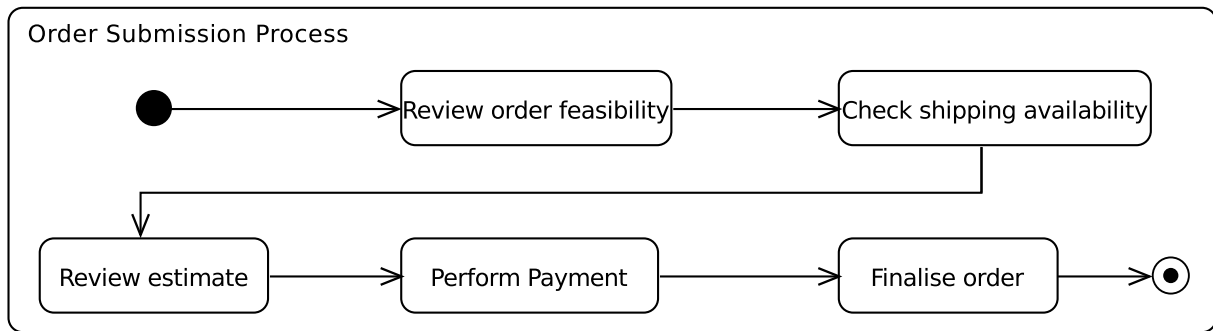


Figure 7.15. Service process model: “Order submission”

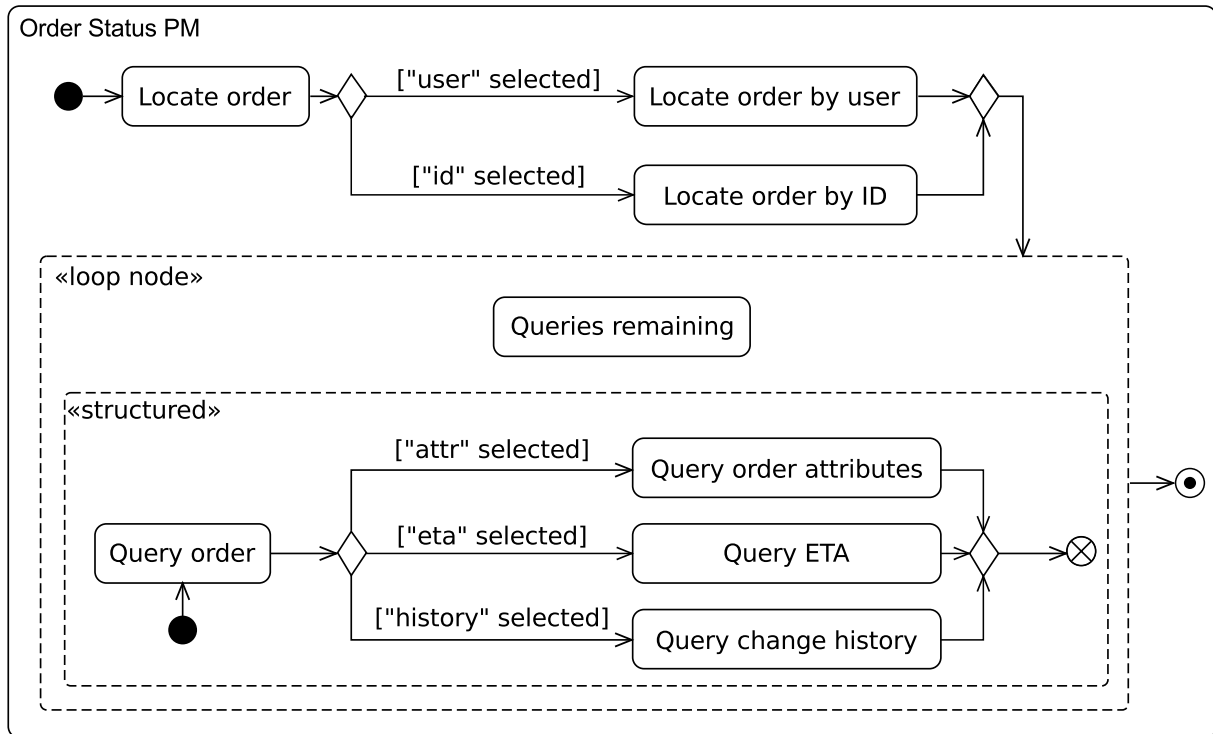


Figure 7.16. Service process model: “Order status reporting”

performed and in which situations. This is the purpose of the SODM service process models, which can be initially created following the procedure presented in Section 4.3.3.3, to be later fine-tuned by hand.

Figures 7.15 to 7.17 show the service models for three of the six extended use case models in Section 7.3.2. The other three models were excluded from the present case study due to their similarity to the considered models. While the performance inference algorithms from Chapter 5 could be applied at this stage, it was decided to postpone their usage to the extended service composition models: these would include all the information required to derive performance test cases from them using the approach in Chapter 6.

The service process model for “Order submission” in Figure 7.15 is a simple ordering of the elements in the extended use case model of Figure 7.9.

In the case of “Order status reporting”, the mapping from the extended use case model in Figure 7.10 to the service process model in Figure 7.16 is slightly more contrived. The

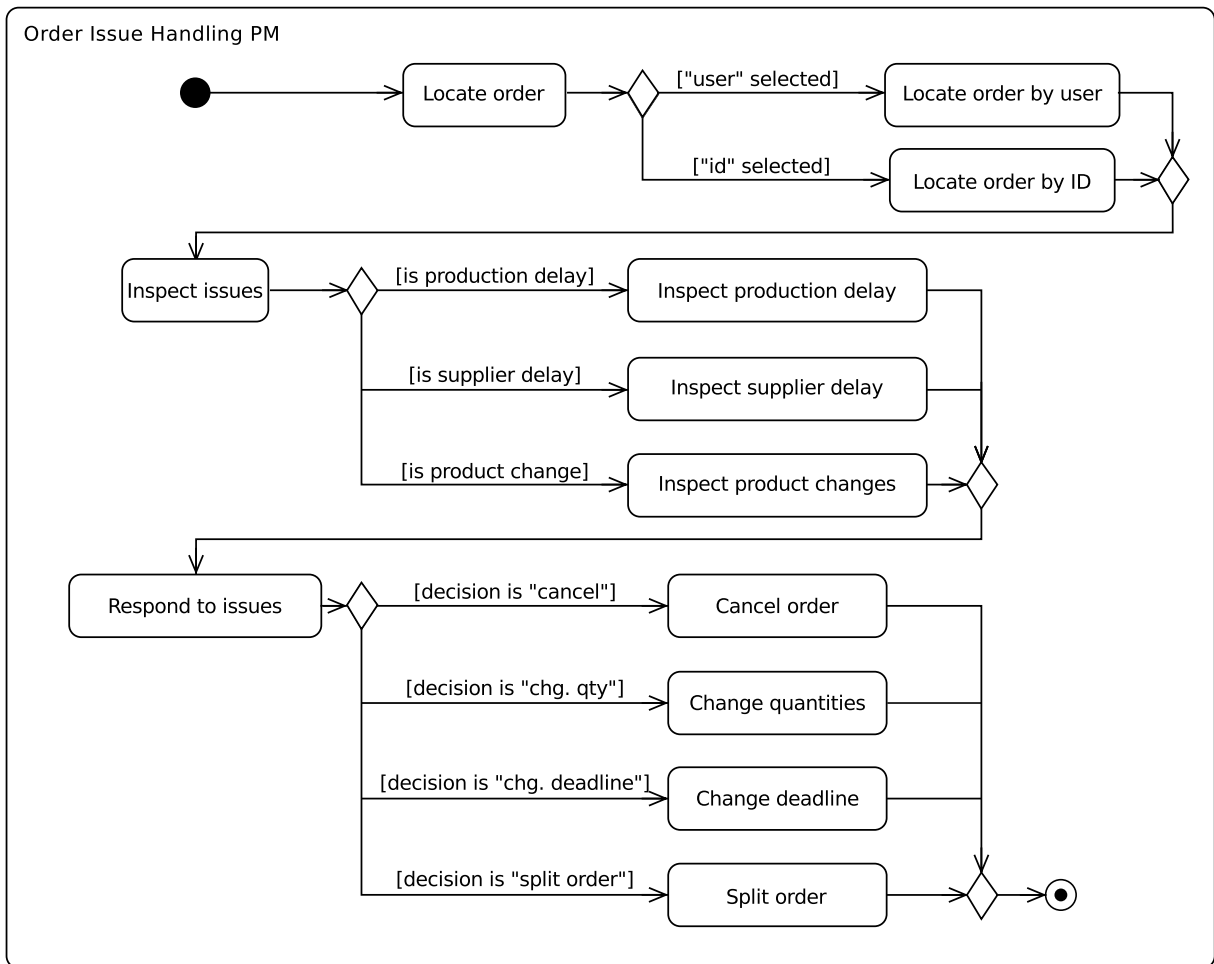


Figure 7.17. Service process model: “Order issue notification”

“Locate order” action decides whether the order should be looked up by username or ID, and then a loop answers to the queries from the customer. This loop was introduced manually after performing the initial mapping.

The “Order issue notification” service process model in Figure 7.17 is also a straightforward mapping of the original extended use case model, in which the action corresponding to each use case extension is picked through a UML decision node.

7.3.4 Service composition models

The last set of platform-independent models are the SODM service composition models, which describe not only *what* should be done and in which order, but also *who* should do it. Additionally, some actions may be further subdivided to make them easier to implement or to distribute responsibilities among the actors of the process (*holons* the present case).

Due to space restrictions and the similarity of the models that would be produced, the present case study has only produced the service composition model (Figure 7.18) for the “Order submission” service process (Figure 7.15).

In order to simplify the layout of the model and make it fit into a single page, an alternative notation from the traditional swimlane-based drawings was used to represent the UML activity partitions in the model. In this notation, the activity partitions are

not explicitly drawn: instead, the names of the partitions to which each action belongs are embedded into their names. For instance, if action “A” should be performed within partition “P”, the decorated action would use “(P) A” as its label. This notation is also allowed by the UML specification [14] (§12.3.10, Figure 12.58).

The actions are distributed among the following actors, which take the role of the highest-level holons or manufacturing agents in the composition:

- The customer interacts with the business-to-business system that manages the orders and provides an accessible web interface.
- The business-to-business (B2B) system interacts with the other automated actors in the composition on behalf of the user and informs the user of the status of the ordering process and its end result.
- The manufacturing execution system (MES) provides the live plant status information required to produce a reasonably accurate ETA for the order.
- The logistics system (Logistics) is external to the manufacturing firm, belonging to one or more shipping companies contracted by Keraben. Interaction with these companies is assumed to be automated through Web Services.
- The Finances department is informed of the amount that should be charged to the customer and takes care of the payment itself.

Most of the actions in the original process model have been divided in two: one for the side of the customer and one for the side of the relevant automated system. In order to illustrate the ability of the algorithms to deal with nested actions, “Review order feasibility” was subdivided using a UML structured activity node instead of being decomposed into multiple separate action nodes.

7.4 Platform-specific models

Throughout the previous sections, the methodology has descended from a high-level description of the enterprise under study and its business processes to specific composition models detailing what should be done in each service composition and who should do it.

The next layer in the OMG Model-Driven Architecture[®] (MDA[®]) approach consists of the Platform Specific Models (PSMs) which extend the previous models with the details required for a specific methodology. In addition, these models will contain the performance annotations that will be used to derive their early performance requirements.

7.4.1 Extended service composition models

At this stage, extended service composition models are built by simply annotating the actions that should become Web Services with the «WS» stereotype. In the case of “Order submission”, the following actions were annotated:

- (B2B) Check product availability
- (MES) Estimate production dates

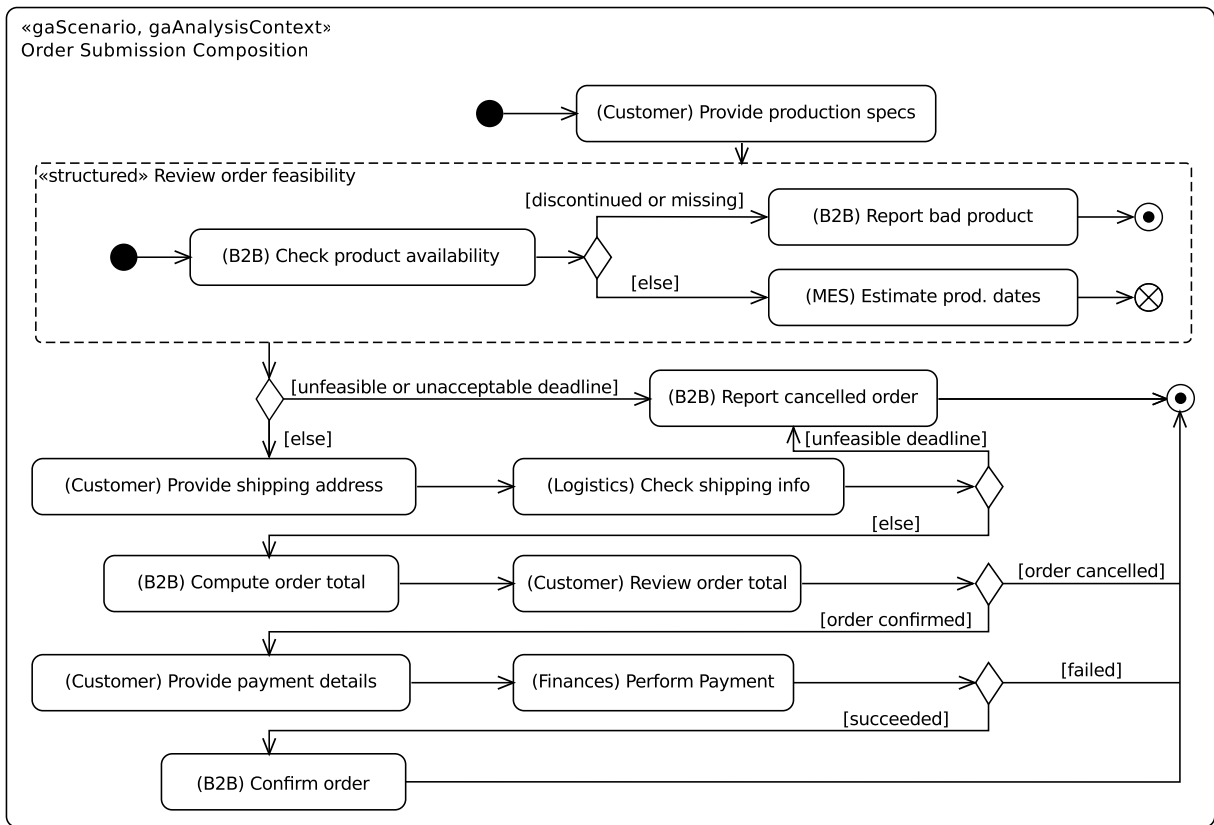


Figure 7.18. Service composition model: “Order submission”

- (Logistics) Check shipping information
- (Finances) Perform payment

Additionally, the model received the following performance annotations:

- The entire process should never take longer than 5 seconds while handling 5 requests per second.
- “Customer” nodes have 0 minimum time and 0 weight (these are actions taken by the user: not by the system).
- “Review order feasibility” has 0 minimum time and weight 5, as generating a schedule can be much more expensive than just checking an item in an inventory.
- “Estimate production dates” originally had weight 3, but after some performance testing (described later) it was revised to have minimum time 0.4s and weight 2.
- The rest of the nodes have minimum time 0 and weight 1, which are the default recommended values.
- The outgoing edges for the decision nodes have been annotated according to these estimations:
 - 10% of all orders refer to a missing or discontinued product.

- 20% of the remaining orders have unfeasible or unacceptable manufacturing deadlines.
- 20% of the remaining orders have unfeasible or unacceptable shipping deadlines.
- 10% of the remaining orders are cancelled after reviewing the order total.
- 5% of the remaining orders cannot be processed due to problems during payment.

As mentioned above, the experience obtained in this case study suggests that the performance inference algorithms should be applied on the service composition models and/or the extended service composition models, but not on the process models. This is because of two reasons. The first reason is that the process models will need to be further decomposed in any case and the inferred performance requirements will not be directly usable for testing. The second reason is that process models would not contain information about who performs the action, and so it would be difficult to estimate if the inferred value would be feasible or not.

After applying the performance inference algorithms, the following performance requirements (rounded to 2 decimal places) were detected for each non-Customer activity:

- “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.

7.4.2 Web Service interface models

The final step in the original SODM methodology was creating the WS interface models for each of the actions that was tagged with «WS» in the extended service composition models. These models were UML class diagrams annotated with stereotypes from a custom profile inspired on the Web Services Description Language (WSDL) specification (§4.3.4).

Following this approach, the SODM WS profile was implemented according to the Eclipse UML implementation [5] using the Papyrus model editors [6]. The resulting profile is shown in Figure 4.5. While implementing the profile, several changes were made from the original version in Figure 4.5:

- The restrictions for the association ends in *AssociationOperationMessage*, *TypeSchema* and *AssociationPartElement* were modelled using explicit executable OCL constraints instead of implicit associations (shown as blue dog-eared annotations).
- *Endpoint* was revised to include a *location* attribute instead of a *name*, following more closely its practical usage in WSDL documents.

- Some types were renamed to follow common style conventions more closely: *Association-OperationMessage* was renamed to *OperationMessageAssociation* and *Association-PartElement* was renamed to *PartElementAssociation*.
- *PartElementAssociation* was simplified, removing its two child classes *PartElement* and *PartType*. Current best practices in WS development recommend using elements and not types, for the sake of simplicity and interoperability.

Using this profile, the WS interface model for “Estimate production dates” shown in Figure 7.20 was created. In this simple model, the definition *MES* would contain all the WS that would be provided by the manufacturing execution system. In particular, it has a *SchedulerService* with a binding to an interface with a single operation (“Estimate-ProductionDates”). This operation takes the reason for the query, the ID of the ISA-95 product production rule (§7.5.1), the requested quantity and its unit of measurement. The operation can reply either with a tentative scheduled (ID, earliest start date and latest end date), or with an error message.

While the model was simple to produce after implementing the UML profile, its usefulness as a separate model is unclear. The model operates at almost the same level of abstraction as WSDL and lacks the specific tooling that most WSDL editors have. Therefore, it can be concluded this particular kind of model could be directly replaced by WSDL documents.

7.5 Implementation

At this point, all the SODM models have been created, and some of these models have been annotated with performance requirements. Before these requirements can be used to generate performance tests, it is necessary to implement the system itself.

This section shows how the relevant part of the manufacturing execution system holon (or manufacturing agent) was designed and implemented from scratch. The holon was divided into three large parts:

- A *data model* describing the concepts it would internally work with, implemented as a *persistence layer* using Spring Roo [9] and Hibernate [11]. The data model is an adaptation of the ISA-95 data model [10]: several changes were required to make it amenable for the simple scheduler implemented within the “Estimate production dates” WS.
- A *web interface* which would allow users to manipulate its internal data model, e.g. to model the manufacturing processes within Keraben with it and observe the results produced by the WS below. The web interface was implemented using the above persistence layer, Spring Roo and the Spring MVC framework [8].
- A set of WS providing functionality to other holons using machine-oriented and interoperable interfaces. In particular, the current implementation focused on the “Estimate production dates” WS that was described above and was implemented using the Apache CXF framework [1].

The following subsections provide additional details about each of these components of the MES holon.

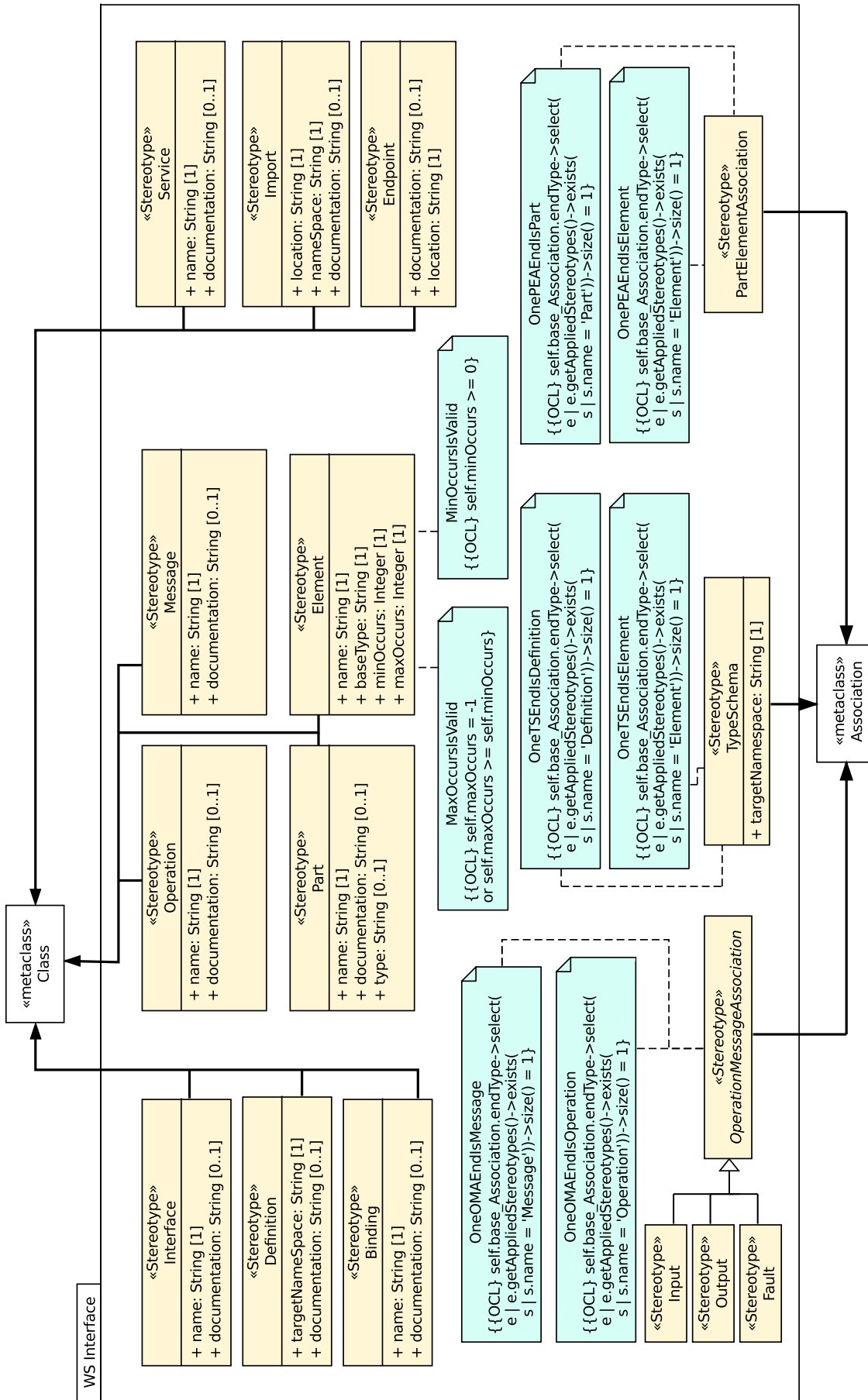


Figure 7.19. Revised Web Service interface metamodel after implementation with the Papyrus tool. Filled arrows represent “extension” of existing UML concepts or *metaclasses*. The dog-eared annotation nodes (shaded in yellow) specify additional OCL constraints on valid stereotype usage. Stereotypes are shaded in blue.

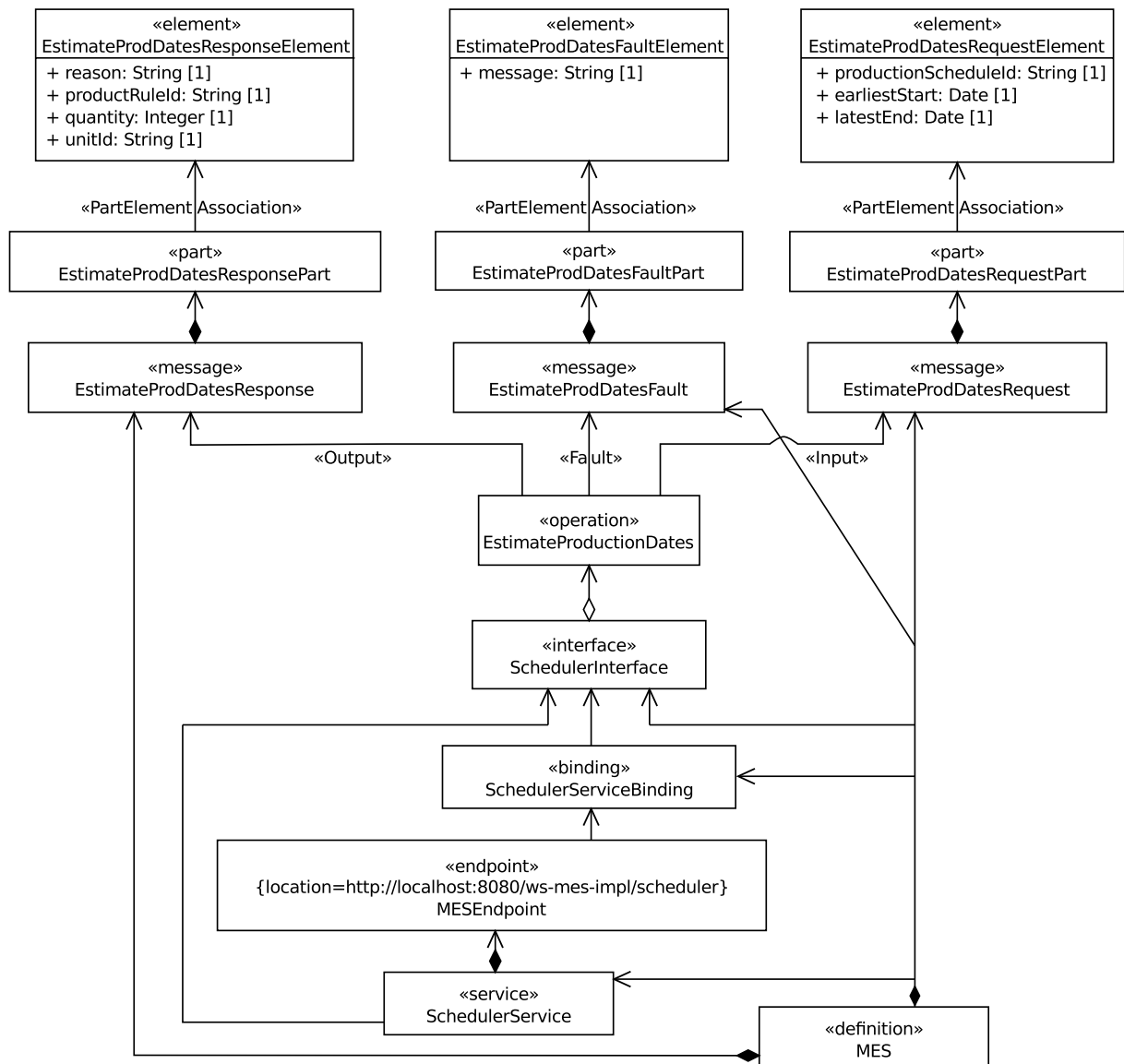


Figure 7.20. Web Service interface model for “Estimate production dates”

7.5.1 Persistence layer: adaptation of the ISA-95 object model

In order to implement the “Estimate production dates” WS it was necessary to write a simple scheduler, and the scheduler itself would need a description of the current resources and processes available in the enterprise. Consequently, an appropriate data model had to be developed.

The persistence layer is based on the object model in ISA-95 part 2, with some changes attempting to reduce the large amount of duplication present in the specification. It is divided into a main package (*domain*), and several subpackages with each of the conceptual areas within ISA-95:

- The main *domain* package (shown in Figure 7.21) provides several common basic concepts that are reused throughout the data model. Most of these concepts are not explicitly represented in ISA-95, being implicitly repeated throughout the entire specification instead.

For instance, the idea of a *Unit* is left in ISA-95 as a single string (“meters”), without regards for the complex issues involving unit-based arithmetic, conversions and comparisons. In the adapted data model, *Units* may be based on others (e.g. “decimeters” is based on “meters” with a conversion factor equal to 0.1) and may have a fractional part (“meters/second” has “second” as its fractional part), which is useful for multiplications. While users are free to define their own *Units*, the system needs to treat some basic units specially: these are the *SystemUnitTypes*. *Quantity-WithUnit* implements a full set of arithmetic, conversion and comparison operators for numeric values annotated with units, while ensuring that illegal operations are reported (e.g. comparing litres with meters).

In addition to better unit handling, the *domain* package contains *Property*, a base class for all the property classes within the data model, and several enumerations used throughout several classes (*ExecutionDependencyType* and *CapabilityType*). Priority levels are also modelled as a separate entity, so users can define their own.

- Three subpackages describe the three kinds of resources available in a manufacturing firm, according to ISA-95: *people*, *equipment* and *material*. These packages closely follow the originals.

The *people* package (Figure 7.22) provides the concept of a *Person* which may belong to zero or more *PersonClasses*. Both *Person* and *PersonClass* objects may have properties of their own. Some of the properties in a person class may be certified for a particular person through a *QualificationTestSpec* and a positive *QualificationTestResult*.

The *equip* package (Figure 7.23) follows a similar pattern: a piece of *Equipment* may belong to zero or more *EquipmentClasses*, and both types of objects may have their own properties. Some of the properties in an *EquipmentClass* may be tested for a certain piece of *Equipment* through a *EquipmentCapabilityTestSpec* and a positive *EquipmentCapabilityTestResult*.

Additionally, *Equipment* may belong to a certain *EquipmentType* in the ISA-95 equipment hierarchy (e.g. site, area or process cell). *Equipment* may also undergo maintenance: a *MaintenanceWorkOrder* contains a set of *MaintenanceRequests*

submitted for certain machines and a set of *MaintenanceResponses* with the work performed at a certain time.

The *material* package (Figure 7.24) is defined around the concept of a *Material-Definition*, which belongs to one or more *MaterialClasses* and may be available throughout several *MaterialLots*, which are arbitrarily subdivided into *Material-Sublots*. Classes, definitions and lots may have their own properties: definition or class properties may be tested on particular lots through *QATestResult* and *QATestSpec*.

- Two subpackages are dedicated to representing the available manufacturing processes (*psegment*) and the products that can be manufactured (*product*).

The *psegment* package (Figure 7.25) defines the concept of a *ProcessSegment*: a step of a manufacturing process which may contain lower level *ProcessSegments*, may depend on other *ProcessSegments* and may have certain *ProcessSegmentParameters*. The personnel, equipment and material requirements for the *ProcessSegments* are included as *PersonnelSegmentSpecs*, *EquipmentSegmentSpecProperty*s and *Material-SegmentSpecs*, respectively. These three kinds of *SegmentSpecs* may have their own properties as well.

The *product* package (Figure 7.26) builds upon *psegment* by describing *Product-ProductionRules* as collections of *ProductSegments*. *ProductSegments* map to *Process-Segments* and also allow for nesting, parameters and inter-segment dependencies. They also allow for extending the original *ProcessSegment* with additional personnel, equipment and material requirements through *PersonnelSpecs*, *EquipmentSpecs* and *MaterialSpecs* and their properties.

- The last three subpackages are used for scheduling: *capability* models the available and committed production capacities, *schedule* models the production currently planed, and *perform* represents the achieved production results.

The *capability* package (Figure 7.27) has been reorganised, adding a common *Capability* parent class for *ProductionCapability* and *ChildCapability*. *ChildCapability* itself is a new parent class for the three kinds of capabilities (*PersonnelCapability*, *EquipmentCapability* and *MaterialCapability*) that a certain *ProductionCapability* may contain.

The *schedule* package (Figure 7.28) is largely unchanged, however: a *Production-Schedule* contains a set of *ProductionRequests*, which in turn may contain *Segment-Requirements* for a certain *ProductSegment*. *SegmentRequirements* may include parameters or detailed personnel, equipment and material requirements through the *ProductionParameter*, *PersonnelRequirement*, *EquipmentRequirement* and *Material-Requirement*, respectively.

The *perform* package (Figure 7.29) largely mirrors the *schedule* package: the *ProductionPerformance* reports the results from a certain *ProductionSchedule*, which are divided into *ProductionResponses* for each *ProductionRequest* and *Segment-Responses* for each *SegmentRequirement*. *SegmentRequirements* detail actual production information and usage of resources through the *ProductionDate*, *PersonnelActual*, *EquipmentActual* and *MaterialActual*, respectively.

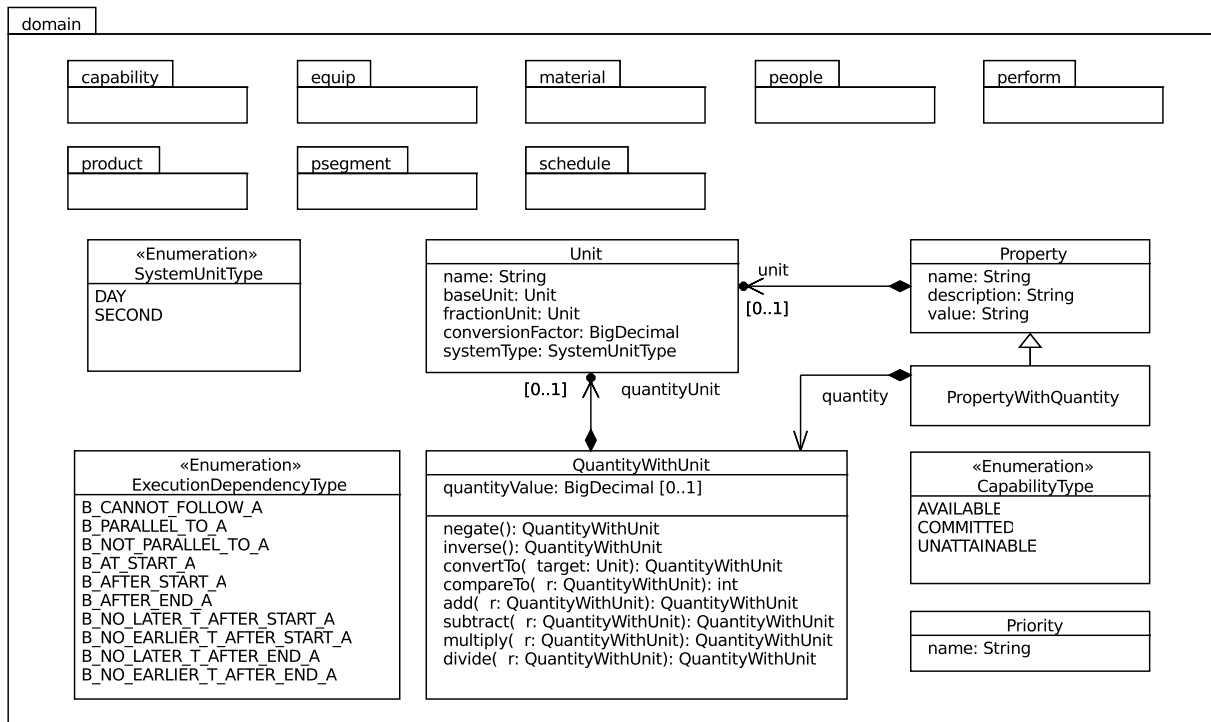


Figure 7.21. UML class diagram for the *domain* base package

While these packages cover a large part of the concepts required in a data model of a manufacturing firm, there are some important omissions. One of the most important ones is the lack of any description of the supply chain: for instance, suppliers are not represented anywhere in the data model. Other issues became apparent when trying to use this ISA-95 inspired data model for scheduling: there is no explicit distinction between make-to-stock and make-to-order production strategies, and setup operations need to be modelled as explicit process segments with their own fixed duration instead of being part of the process segment they are related to.

Other issues have already been discussed: there is a large amount of duplication throughout the original specification, and units are not well modelled. Nevertheless, ISA-95 has shown to be a good starting point, even though it would need considerable changes to use it to implement a full-fledged MES.

The presented data model has been implemented as a set of Java classes using the tools from the Spring Roo project, which generate code targeting the Hibernate framework. Using these Java classes, two kinds of databases were automatically generated: in-memory HSQLDB databases for internal testing [15], and a traditional PostgreSQL [16] database for deployment and performance testing. The resulting databases are reasonably complex: for instance, the PostgreSQL database has 109 interrelated tables.

7.5.2 Web interface: specification of rectification processes with ISA-95

Having implemented the data model as a persistence layer, the next step for implementing the “Estimate production dates” was specifying the Keraben tile rectification resources, processes and production capacities with it.

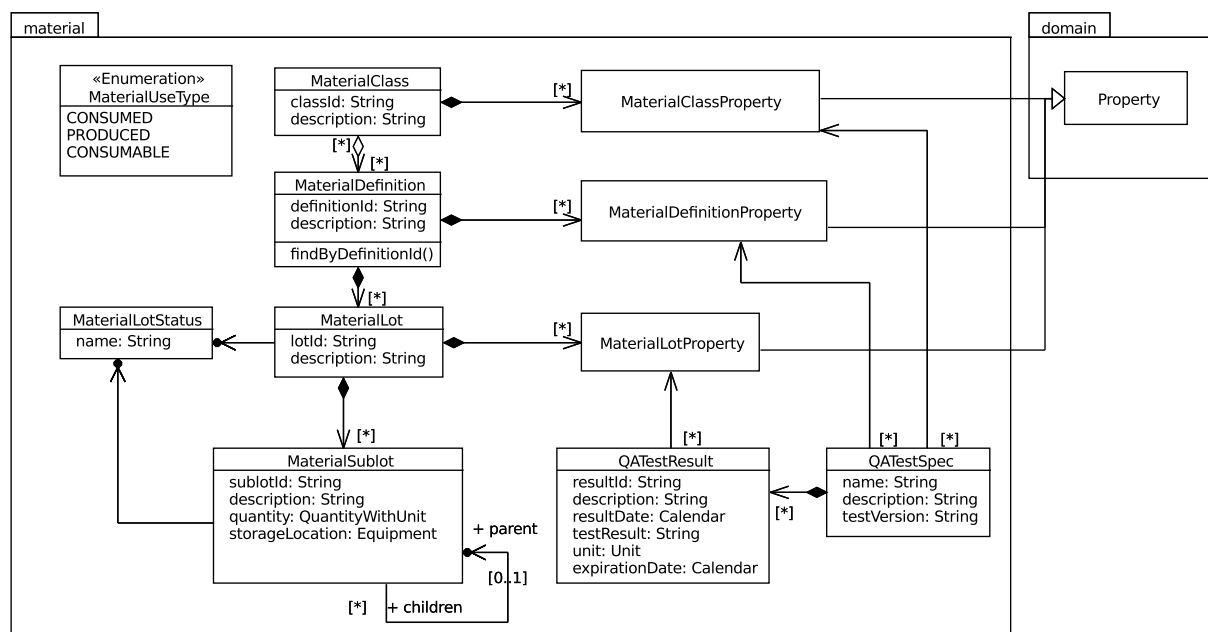


Figure 7.24. UML class diagram for the *material* subpackage

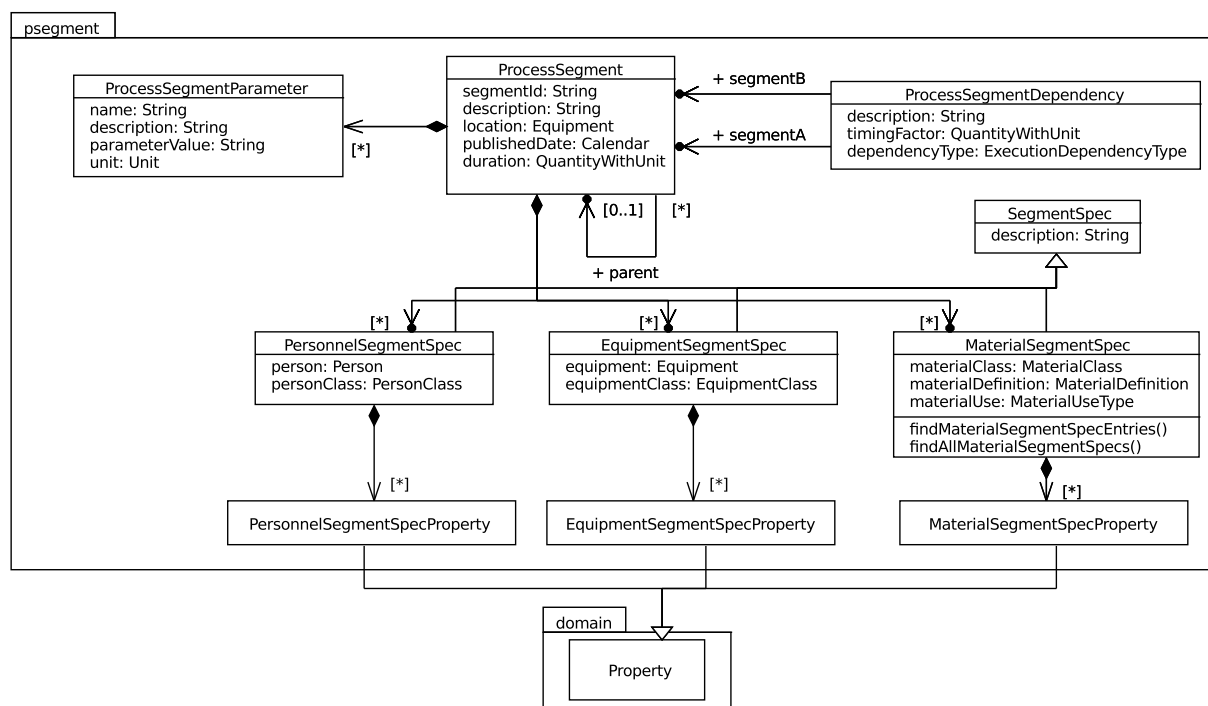


Figure 7.25. UML class diagram for the *psegment* subpackage

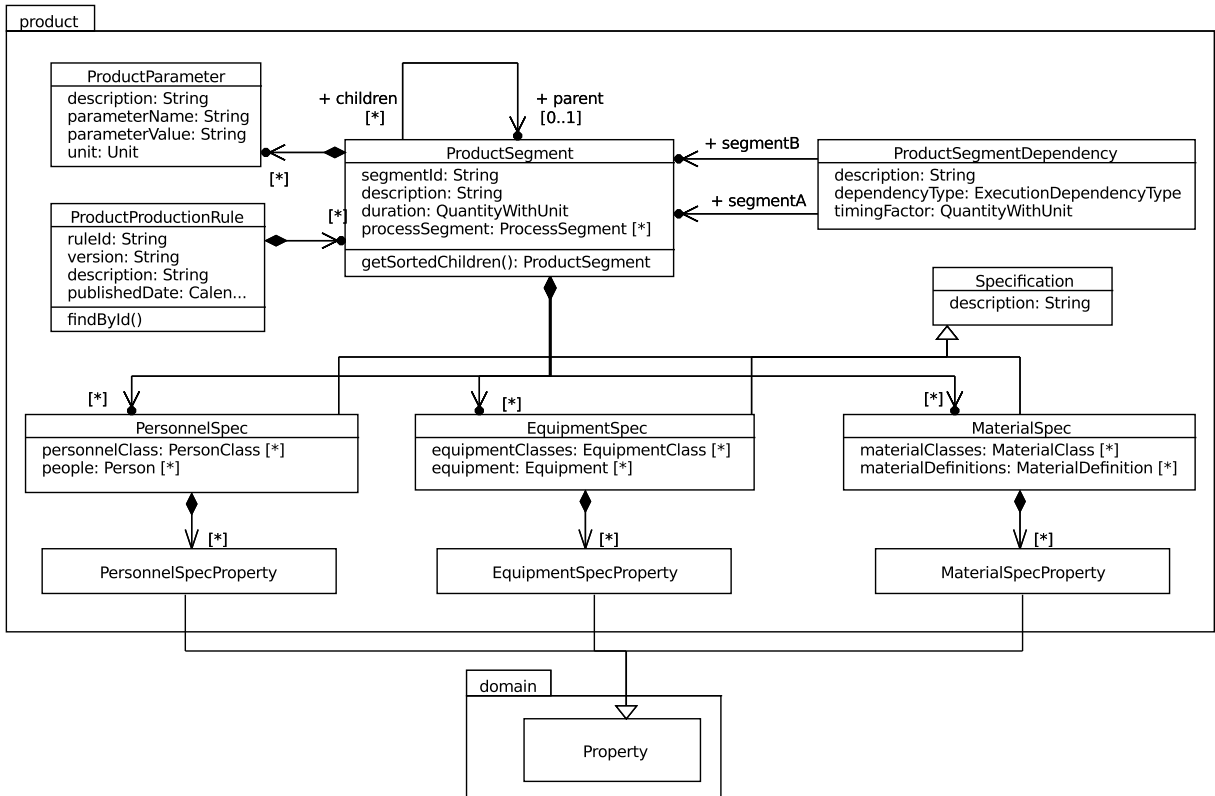


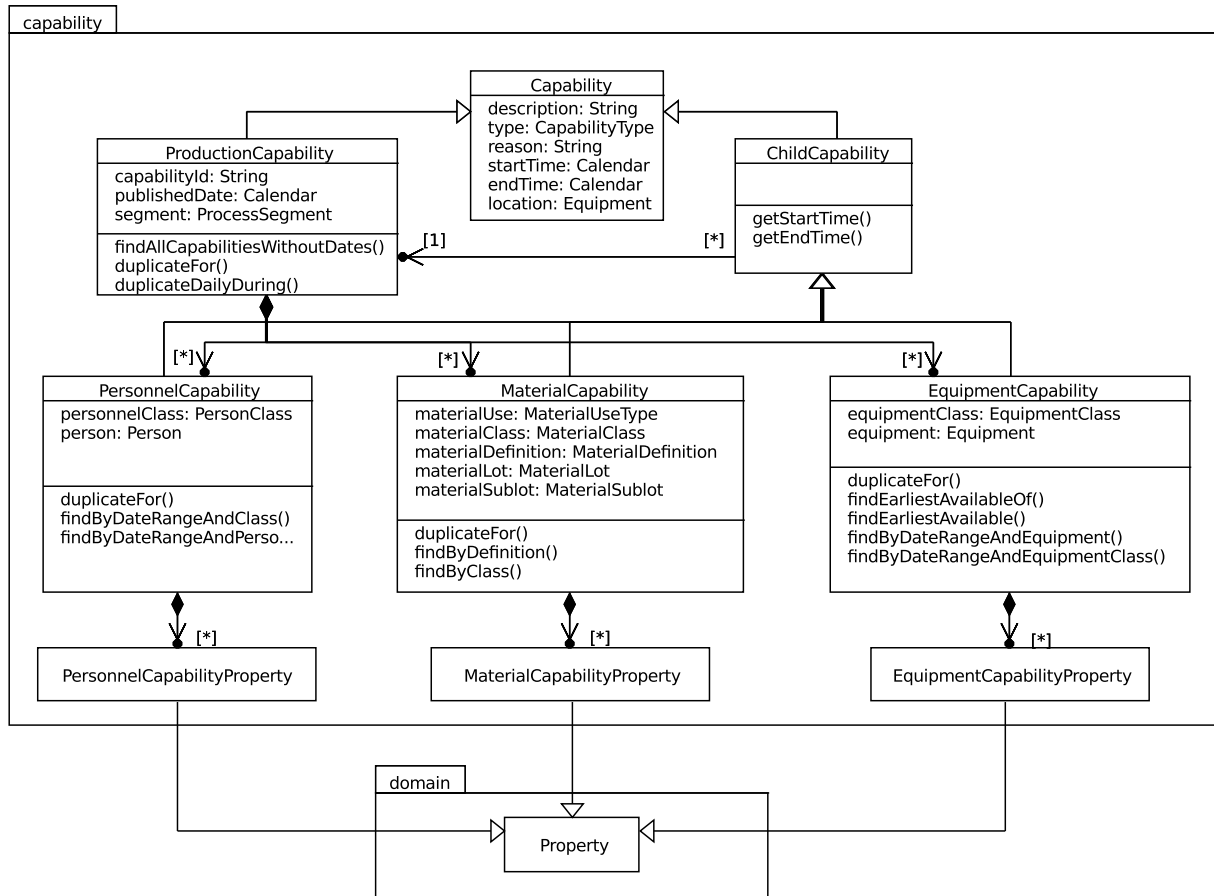
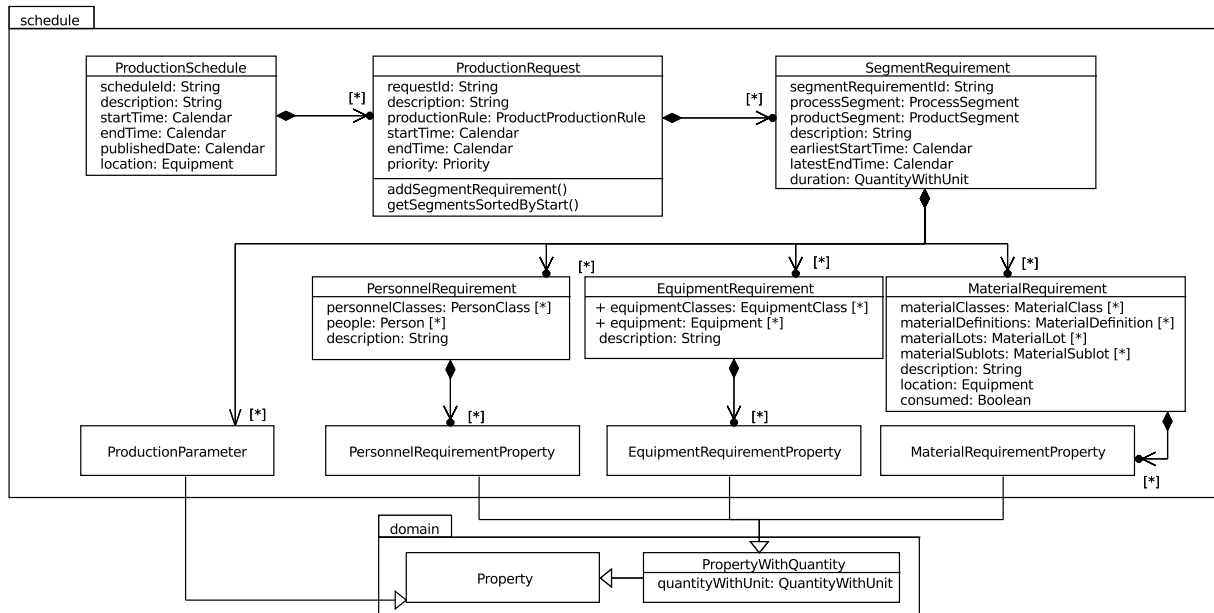
Figure 7.26. UML class diagram for the *product* subpackage

To this effect, a web interface (shown in Figure 7.30) was built for manipulating the entities of the data model. A first version of the web interface was originally generated from the persistence layer itself, using Spring Roo and targeting the Spring MVC framework [8]. This first version was then manually improved in regards to usability by grouping related concepts together, simplifying data entry and improving the way in which information was simplified.

For instance, the default navigational menu was replaced with a set of HTML5 collapsible menus with persistent local state. The web interface was also extended with the ability to define production capacity templates and instantiate them for a particular time period: this way, the current daily capacity can be defined as a *ProductionCapability* with no time period and then be quickly duplicated for each day of the present week. This was used to generate the estimated production capabilities for every working day until December 2013.

According to the above data model, the Keraben plant consists of the following resources:

- 2 person classes (“Operators” and “Plant Managers”). There are 4 operators, only identified by a number (“Operator 1” to “Operator 4”).
- 1 equipment class for each kind of machine in the rectification process and the production lines themselves: “EQC-Feeders”, “EQC-Cutters”, “EQC-HRectifiers”, “EQC-VRectifiers”, “EQC-Dryers”, “EQC-Inspectors”, “EQC-BoxLoaders” and “EQC-RectifyingLines”. There are 2 matching pieces of equipment for each class, representing the two rectification lines in the firm.
- Additional equipment classes model the rest of the ISA-95 equipment hierarchy,

Figure 7.27. UML class diagram for the *capabilities* subpackageFigure 7.28. UML class diagram for the *schedule* subpackage

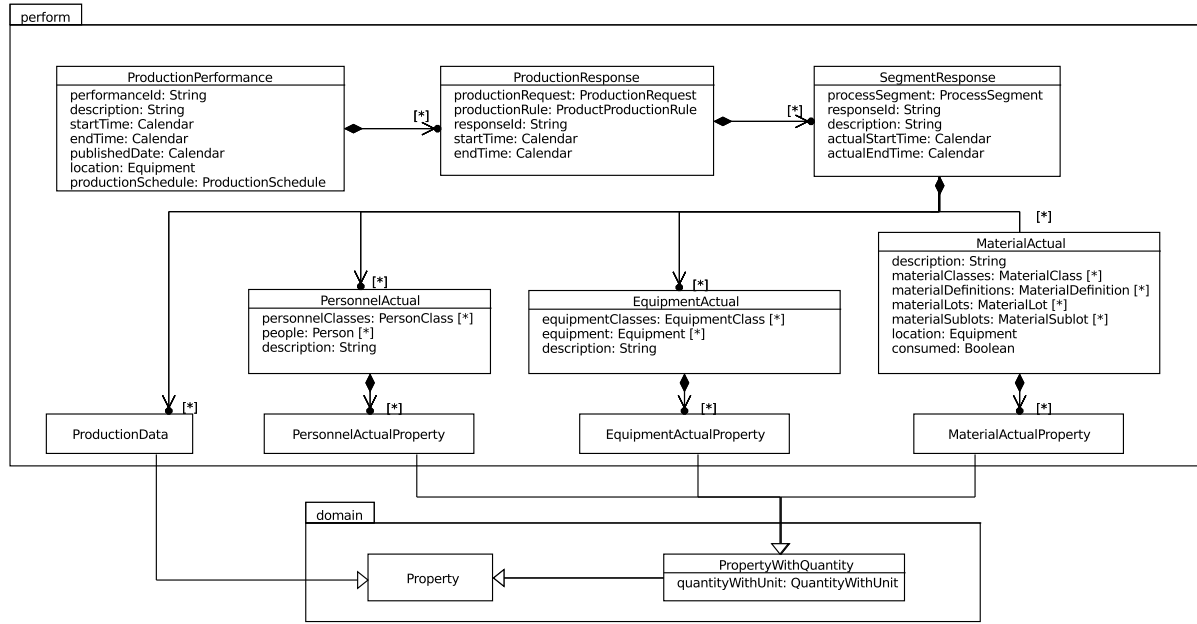


Figure 7.29. UML class diagram for the *perform* subpackage

several *Equipment* instances have been defined for the Keraben enterprise group (“EQ-Group”), the Keraben firm (“EQ-FirmTiles”), the production area (“EQ-AreaProd”) and the storage areas for raw and finished products (“EQ-AreaStorageRaw” and “EQ-AreaStorageFinished”).

- 3 material classes: consumables (grinding wheels and replacement belts), source materials (medium-, large- and custom-sized raw tiles) and rectified tiles (medium-, large- and custom-sized raw tiles rectified on 2 or 4 sides).
- 2 material lots with 300 medium raw tiles and 300 large raw tiles, respectively. Custom tiles are produced on demand by cutting large raw tiles.

The rectification processes have been modelled as follows:

- 14 process segments for each step of the rectification process, including line setup times (with fixed times) and considering different variable times for each tile size:
 - “SG-R-Setup-LM”, “SG-R-Setup-C”: line setup for large/medium tiles and custom tiles.
 - “SG-R-Feeder-L”, “SG-R-Feeder-M”: piece feeding for large or medium tiles, respectively.
 - “SG-R-Cutter”: piece cutting for producing custom tile sizes.
 - “SG-R-HRectifying-L”, “SG-R-HRectifying-M” and “SG-R-HRectifying-C”: rectification of the horizontal edges.
 - “SG-R-VRectifying-L”, “SG-R-VRectifying-M” and “SG-R-VRectifying-C”: rectification of the vertical edges.
 - “SG-R-Drying”, “SG-R-Inspection” and “SG-R-BoxLoading”: the remaining steps of drying the pieces, inspecting them and loading them into boxes.

ISA-95 Model Administration Panel

PEOPLE ▼

EQUIPMENT ▼

MATERIAL ▼

PRODUCTION CAPABILITIES ▼

PROCESS SEGMENT ▲

Process Segments

Personnel Segment Specs

Personnel Segment Spec Properties

Equipment Segment Specs

Equipment Segment Spec Properties

Material Segment Specs

Material Segment Spec Properties

Process Segment Dependencies

Process Segment Parameters

PRODUCT PRODUCTION RULE ▼

PRODUCTION SCHEDULE ▼

PRODUCTION PERFORMANCE ▼

TYPES ▼

▼ List all Process Segments

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

List results per page: [5](#) [10](#) [15](#) [20](#) [25](#) | Page 1 of 2

[Home](#) | Language: | Theme: [standard](#) | [alt](#)

Figure 7.30. Screenshot of the web administration panel

Each process segment has segment specifications requiring an operator and an appropriate machine (or the entire line for the setup tasks). “SG-R-Feeder-L” and “SG-R-Feeder-M” have material specifications requiring the raw tiles, and the rectifying segments take the raw tiles (after cutting for the custom-sized ones) and produce rectified tiles (first on 2 edges, and then on 4 edges).

For the sake of simplicity, each process segment takes batches of 30 tiles. Process segment dependencies indicate the appropriate order: feeding should come first, then cutting (if required), then horizontal and vertical rectification, and finally drying, inspection and packaging.

- 6 product production rules for each tile size (large, medium and custom) and finish (rectified on 2 edges or 4 edges). Each production rule contains a root product segment with no process segment that ensures that entire production lines are reserved during the manufacturing of a single batch of tiles, and one child product segment one product segment for every process segment required to produce it.

For instance, “PPR-RC2” (the rule for producing custom-sized tiles rectified on 2 edges) has “PPR-RC2-Parent” as its root product segment. “PPR-RC-Parent” in turn contains “PPR-RC2-Setup”, “PPR-RC2-Cut”, “PPR-RC2-HRectifying” and so on.

Finally, the available production capability was modelled by defining a template of the current production capacity, in which each production line could produce 1000 pieces a day and operators could work part-time (4 hours a day).

7.5.3 Web service: provision of a scheduler WS

So far, the rectifying processes have been described using the implemented data model. The final step was implementing the “Estimate production dates” WS itself. For the purpose of this case study, this WS was kept as a simple scheduler that would use the stored production capabilities to produce a *ProductionSchedule* doing the required work as soon as possible while balancing the workload over both production lines.

The WS uses the same persistence layer as the web interface and offers its services through a SOAP WS (§3.1.2.4). Most of the low-level communication details are abstracted away by the Apache CXF framework [1], which generates the required code from the WSDL document that was produced from the WS interface model in Figure 7.20 (page 7.20). After some initial manual testing using the soapUI tool [7], the WS underwent functional testing using the facilities in the Spring framework.

7.6 Performance test generation and execution

Having defined the performance requirements and implemented the “Estimate production dates” WS backed by an appropriate data model and a formalisation of the Keraben processes, the performance tests can now be generated using the approach described in Chapter 6 (Figure 6.10 in page 6.10).

First, a message catalogue was automatically derived from the WSDL document for the “Estimate production dates” WS by using the . The catalogue is shown in Listing 7.1. In particular, the required information for generating input messages for the “generateSchedule” operation (line 5) of the “SchedulerImplService” (line 4) is contained in lines 6–20. Lines 7–11 indicate the variables that must be populated to produce a single message, and lines 12–19 are the template that will be used to produce the message itself.

The “generateSchedule” operation in line 5 is then weaved with the the “Estimate production dates” action in the extended use case model based on Figure 7.18, using the Epsilon ModeLink editor [12]. The model-to-text transformations described in Section 6.5 generated almost all the required code, test data, configuration files and launch scripts required for performance testing with The Grinder [2] tool and the Grinder Analyzer [3] Maven plugin.

However, it was necessary to manually customise the TestSpec document describing the test data to be generated so it would use existing product production rules, meaningful quantities and valid units. The resulting TestSpec document is shown in Listing 7.2, and illustrates how the generated code can be still customised to fit a particular domain.

After running the performance tests, the Grinder Analyzer showed that the WS could handle about 30 transactions per second while replying under 0.5s, which is well within the estimated time limit of 2.28s while handling 5 requests per second. For the purposes of this case study, tests were run on a server running at the same machine, which stayed otherwise idle during the test: regular users would be advised to mimic their production environment as much as possible for the test.

Listing 7.1 Message catalogue generated from the WSDL document of the “Estimate production dates” Web Service

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <mes:services xmlns:mes="http://serviceAnalyzer/messageCatalog">
3   <mes:service name="SchedulerImplService" uri="http://impl.scheduler.isa95study.sodmt.uca.es/">
4     <mes:port address="http://localhost:8080/ws-mes-impl/scheduler" name="SchedulerImplPort">
5       <mes:operation name="generateSchedule">
6         <mes:input>
7           <mes:decls>
8             <mes:typedef min="1" name="TQuantity" type="int"/>
9             <mes:typedef element="string,␣string,␣TQuantity,␣string" name="TGenerateSchedule" type="tuple"/>
10            <mes:variable name="generateSchedule" type="TGenerateSchedule"/>
11          </mes:decls>
12          <mes:template><![CDATA[
13            <s:generateSchedule xmlns:s="http://www.uca.es/sodmt/isa95study/scheduler">
14              <reason>$generateSchedule.get(0)</reason>
15              <productRuleId>$generateSchedule.get(1)</productRuleId>
16              <quantity>$generateSchedule.get(2)</quantity>
17              <unitId>$generateSchedule.get(3)</unitId>
18            </s:generateSchedule>
19          ]]></mes:template>
20        </mes:input>
21        <mes:output>
22          <mes:decls>
23            <mes:typedef element="string,␣dateTime,␣dateTime" name="TGenerateScheduleResponse" type="tuple"/>
24            <mes:variable name="generateScheduleResponse" type="TGenerateScheduleResponse"/>
25          </mes:decls>
26          <mes:template><![CDATA[
27            <s:generateScheduleResponse xmlns:s="http://www.uca.es/sodmt/isa95study/scheduler">
28              <productionScheduleId>$generateScheduleResponse.get(0)</productionScheduleId>
29              <earliestStart>$generateScheduleResponse.get(1)</earliestStart>
30              <latestEnd>$generateScheduleResponse.get(2)</latestEnd>
31            </s:generateScheduleResponse>
32          ]]></mes:template>
33        </mes:output>
34        <mes:fault name="GenerateScheduleFault_Exception">
35          <mes:decls>
36            <mes:variable name="generateScheduleFault" type="string"/>
37          </mes:decls>
38          <mes:template><![CDATA[
39            <s:generateScheduleFault xmlns:s="http://www.uca.es/sodmt/isa95study/scheduler">
40              <message>$generateScheduleFault</message>
41            </s:generateScheduleFault>
42          ]]></mes:template>
43        </mes:fault>
44      </mes:operation>
45    </mes:port>
46  </mes:service>
47 </mes:services>

```

Listing 7.2 Customised TestSpec specification of the input data for testing “Estimate production dates”.

```

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

typedef string (values={
  "PPR-RC2", "PPR-RC4", "PPR-RM2", "PPR-RM4", "PPR-RL2", "PPR-RL4"
}) ProductProductionRule;

TGenerateSchedule generateSchedule;

```

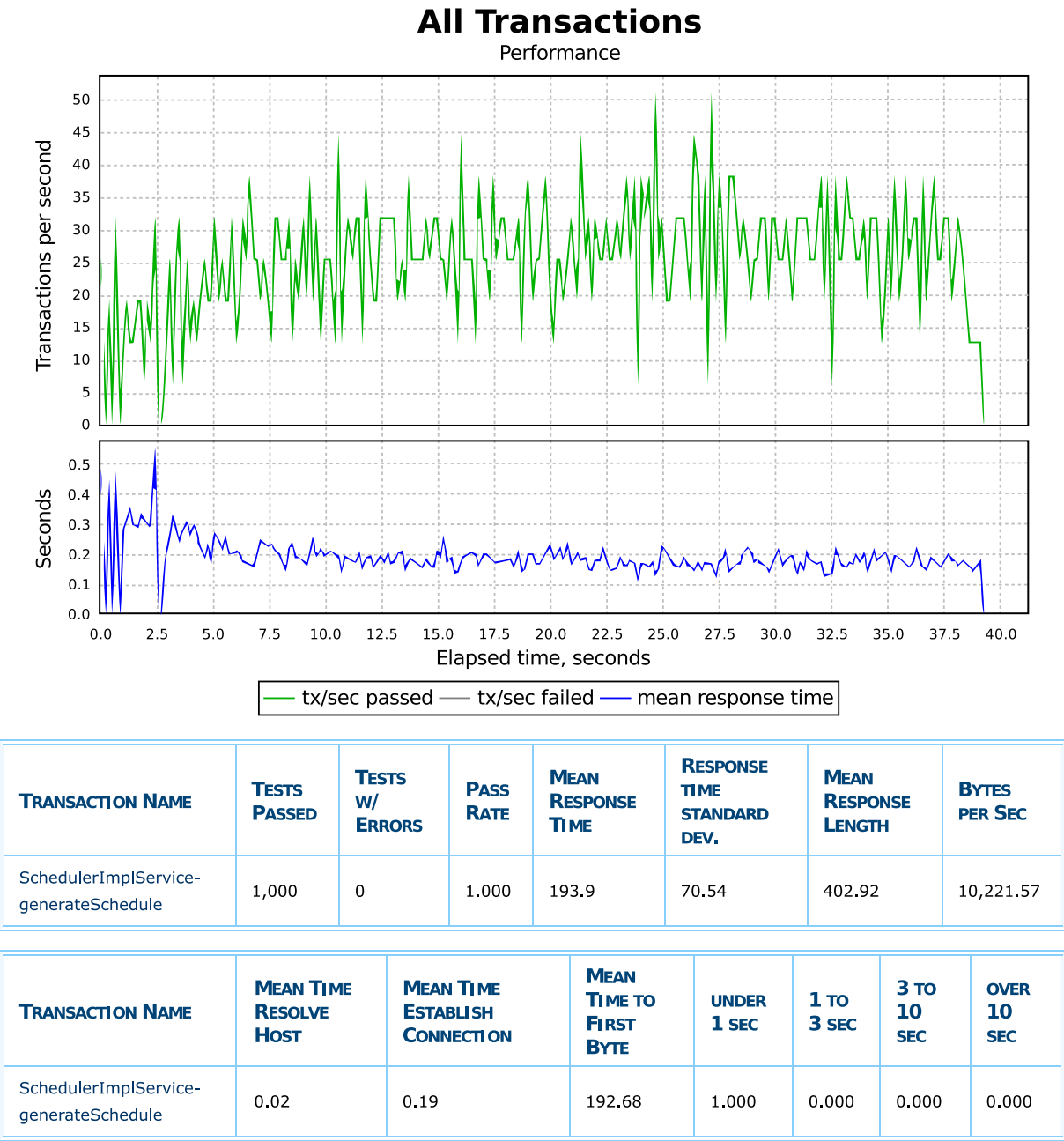


Figure 7.31. Performance testing results for “Estimate production dates”

7.7 Conclusion

The present chapter has shown a complete example of how the SODM+T methodology would be used in a manufacturing context, focusing on a specific manufacturing holon (the manufacturing execution system in a certain firm) and a certain service within it (the scheduler used to estimate production dates).

The Keraben S.A. manufacturing enterprise was selected as a considerable amount of information on its business practices was collected during a collaboration with Lledó Prades Martell from the Universitat Jaume I. Starting with a high-level description of the enterprise, later sections provided additional details on the manufacturing process for porcelain stoneware, the available facilities, providers and information and material flows.

This high-level information was used to derive the first layer of the SODM+T methodology, comprised by the Computation Independent Models (CIMs). These models included a Gordijn value model with the value exchanges between the Keraben firm, its collaborators in the supply chain and its customers, and a BPMN diagram with the overall business model followed by the firm. From this high-level business model, a subset of the business services dedicated to customer order and production schedule handling was selected.

The next layer of SODM+T was dedicated to the Platform Independent Models (PIMs). While the use case model could be easily extracted from the previous business service list, the extended use case models required adding manually new insights. The use cases in these models were then arranged into service process models, focusing on the order handling service processes for the sake of simplicity. Finally, the “Order submission” service process was fully fleshed out as a service composition model by distributing the work involved among the relevant actors (holons).

The PIM layer was followed with the Platform Specific Models (PSMs), in which some of the actions in the “Order submission” service composition model were tagged with the «WS» stereotype to indicate that they were to be implemented as Web Services, producing an extended service composition model. This model was tagged with performance annotations from which specific performance requirements were derived for each action in the model.

Before using the inferred performance requirements to generate performance tests, it was necessary to implement the modelled part of the system: the “Estimate production dates” WS of the MES holon in the Keraben firm. This required adapting the ISA-95 object model to current object-oriented programming best practices and implementing it as a reusable persistence layer, using the Spring Roo toolset and the Hibernate object-relational mapping framework. This persistence layer was then reused to develop a web interface for validating the data model and formalising the rectification processes within Keraben, and for implementing the “Estimate production dates” WS itself.

Implementing the ISA-95 object model using current tools and using it to describe a manufacturing process revealed several issues in the ISA-95 specification. Some of these issues could be attributed to the lack of a reference implementation for the object model itself: developing a simple web interface such as the one in this case study could have been useful to find some of them. The main issue is the high degree of duplication present throughout the specification: for instance, instead of having a central “Property” concept, the same attributes are repeated on every kind of property. In other cases, there is a lack of attention to detail on aspects that are important for data quality and consistency, such as proper handling of units, more restrictive specifications for identifiers (e.g. for process

segments, pieces of equipment and so on) or how to model certain situations (make-to-stock versus make-to-order). In summary, this confirms that future specifications of the sort should follow a user-centred approach in which the document is developed in parallel with a simplistic reference implementation that helps find inconsistencies or unclear points.

The case study also highlighted several points in which SODM+T should be changed. The WS interface model inherited from SODM did not produce the expected productivity benefits when compared to directly developing the WSDL document, as it operated at the same level of abstraction: therefore, future applications of SODM+T would be advised to skip them. While the performance inference algorithms could have been applied as early as when the service process models were first developed, their performance requirements would not have helped much, since these models do not indicate who should perform each task. For this reason, the performance inference step is best delayed to the very last step before implementation: extended service composition models. These minor issues do not detract from the overall success of this case study: SODM+T showed that it could guide the user from high-level descriptions to implementation and then assist the user in producing the concrete performance tests themselves.

References

- [1] Apache Software Foundation. Apache CXF, November 2013. URL <https://cxf.apache.org/>. Last checked: November 6th, 2013. 7.20, 7.32
- [2] P. Aston and C. Fitzgerald. The Grinder, a Java Load Testing Framework, 2012. URL <http://grinder.sourceforge.net/>. Last checked: November 6th, 2013. 7.32
- [3] T. Bear. Grinder Analyzer homepage, July 2012. URL <http://track.sourceforge.net/>. Last checked: November 6th, 2013. 7.32
- [4] M. V. de Castro. *Aproximación MDA para el desarrollo orientado a servicios de sistemas de información web: del modelo de negocio al modelo de composición de servicios web*. PhD thesis, Universidad Rey Juan Carlos, March 2007. 7.5
- [5] Eclipse Foundation. Homepage of the mdt uml2 project, June 2013. URL <http://www.eclipse.org/modeling/mdt/?project=uml2>. Last checked: November 6th, 2013. 7.19
- [6] Eclipse Foundation. Homepage of the papyrus project, June 2013. URL <http://www.eclipse.org/papyrus/>. Last checked: November 6th, 2013. 7.19
- [7] eviware.com. soapUI home page, 2012. URL <http://www.soapui.org/>. 7.32
- [8] GoPivotal. Spring Framework homepage, November 2013. URL <http://projects.spring.io/spring-framework/>. Last checked: November 6th, 2013. 7.20, 7.28
- [9] GoPivotal. Spring Roo homepage, August 2013. URL <http://projects.spring.io/spring-roo/>. Last checked: November 6th, 2013. 7.20
- [10] International Electrotechnical Commission. IEC/FDIS 62264-2:2004 – enterprise-control system integration – part 2: Model object attributes, 2004. 7.20

- [11] JBoss Community. Hibernate homepage, November 2013. URL <http://hibernate.org/>. Last checked: November 6th, 2013. 7.20
- [12] D. S. Kolovos. Epsilon ModeLink, 2012. URL <http://eclipse.org/gmt/epsilon/doc/modelink/>. Last checked: November 6th, 2013. 7.32
- [13] Object Management Group. MDA Guide version 1.0.1, June 2003. URL <http://www.omg.org/cgi-bin/doc?omg/03-06-01>. Last checked: November 6th, 2013. 7.7
- [14] Object Management Group. Unified Modeling Language (UML) 2.4.1, August 2011. URL <http://www.omg.org/spec/UML/2.4.1/>. Last checked: November 6th, 2013. 7.17
- [15] The hsql Development Group. HSQLDB homepage, January 2013. URL <http://hsqldb.org/>. Last checked: November 6th, 2013. 7.25
- [16] The PostgreSQL Global Development Group. PostgreSQL homepage, November 2013. URL <http://www.postgresql.org/>. Last checked: November 6th, 2013. 7.25

Conclusions and future work

The present chapter closes the Thesis by listing the obtained results in Section 8.2, outlining the future lines of work branching from this Thesis in Section 8.2 and listing the publications produced during the present Thesis in Section 8.3.

8.1 Obtained results

Throughout the years, market forces have required more advanced products at ever decreasing costs, forcing manufacturing enterprises to specialise in their core strengths and collaborating with others for those activities that are outside their value-adding activities. This has resulted in an accelerating trend towards increasingly distributed manufacturing systems, where networks of manufacturing agents collaborate in a variety of configurations, which can range from the rigid hierarchic model to the highly flexible but hard to manage heterarchical model.

One of these configurations is known as a *holarchy*: a network of *holons* or entities that are both wholes (*hol-*) and parts (*-on*). A holon can be part of several higher-level holons and also be formed by several lower-level holons. Holonic manufacturing has received considerable attention in the area of manufacturing control while implemented as multi-agent systems, but not so much in the area of enterprise integration, where the emergent behaviour may make it difficult to offer the necessary behaviour and performance guarantees that inter-enterprise integration requires. In addition, intelligent agent platforms usually have issues of their own: they may have a high learning curve for existing IT workers, they are hard to integrate with legacy technologies and the widely available options may be immature in comparison with other technologies.

In the last few years, these issues have prompted researchers to integrate service-based technologies into intelligent agents. Services are pieces of software that can be reused from anywhere in the organisation to perform a certain task. Their implementations tend to be based on the same standards as the Web (making them Web Services) and use off-the-shelf technology that is easier to integrate with existing systems. Nevertheless, the issues with their emergent behaviour in some contexts still stand.

The present Thesis has proposed relaxing the tight link commonly seen between holons and agents and instead starting from the view of holons as services, which may need varying degrees of intelligence to perform their role: from business holons that need to be predictable and repeatable and may use a combination or *composition* of services, to machine control holons that may exhibit emergent behaviour by using intelligent agents with more advanced communication primitives. The Thesis has focused on the business and high-level manufacturing management holons which constitute the interface between levels 3 and 4 of the ISA-95 enterprise integration architecture. The holarchy formed by these business holons underlying each high-level manufacturing agent can be seen as a Service Oriented Architecture: a view of the organisation as a well-organised catalogue of services, normally implemented as Web Services.

While a SOA is a good fit for implementing a holarchy of business holons, developing one is not an easy task. For this reason, a multitude of methodologies have been suggested prior to the beginning of this Thesis. The survey presented in this Thesis studied a considerable range of methodologies, from the high-level conceptual methodology by Erl to the comprehensive IBM SOMA methodology. Since many Spanish manufacturing firms are small or medium enterprises with fewer resources dedicated to internal IT, other more

lightweight methodologies such as BPSOM or SODM were considered. In particular, the SODM methodology was found to have a good balance between cost and capabilities, and its model-driven approach based on constructing increasingly detailed representations of the SOA ensured a certain degree of formalisation and automation.

Nevertheless, a common shortcoming in all these methodologies was found: none of them explicitly assisted users in developing test cases for the services themselves. For that reason, the present Thesis suggested two basic approaches for extending SODM so it could help potential users test their services: adding high-level performance requirements that would trickle from the high-level service compositions to the individual WS, and specifying service contracts which would describe the expected behaviour of the services. These two extension paths would assist non-functional and functional testing of the services, respectively.

This thesis has developed the side dedicated to supporting non-functional testing, resulting in an extended SOA methodology called SODM+T (SODM with Testing) that can be used to analyse and design the holarchies of business holons in distributed manufacturing systems. SODM+T reimplements most of SODM on top of the state-of-the-art Epsilon toolkit and extends the SODM service process and service composition models with local and global performance annotations on the expected load and response times of each action and the whole workflow, respectively. The global performance annotations are mandatory, while the local performance annotations that deal with the modeller's partial knowledge of the computational cost of each action in the model: this partial knowledge may include minimum times or a relative weight. Repetitive activities can be modelled using local annotations indicating the estimated number of times that an action and its contents will be executed.

These partially annotated extended models are then processed by two algorithms that infer the expected load and required response time for each action in the activity, helping developers study the implications and feasibility of the performance requirements at an early stage of development of the SOA, before concrete services have been implemented. The load or *throughput* inference algorithm is rather simple: essentially, it propagates the load received through the start of the process or composition to the rest of the actions by computing a certain expression in each action, while ensuring that actions are not traversed before their predecessors. In contrast, the response time or *time limit* inference algorithm has gone through many revisions, resulting in three high-level alternative implementations:

- A formulation based on linear programming (LP) using the `glpsol` solver from the GNU Linear Programming Kit. The annotated models are transformed automatically into a problem written in the GNU MathProg language, which is solved by `glpsol`: the results are fed back to the model.

This formulation is easy to understand and verify, but suffers from performance issues due to the exponential growth of the number of paths in a model as more nodes are added. It is also complicated by the need for invoking an external tool and interpreting its results.

- A simpler formulation based on an *ad hoc* graph-based algorithm that enumerates all the paths from the initial node to the final nodes and traverses the paths from the most to the least strict ones.

The first versions used fixed point techniques and had no support for minimum times,

weights or expected iteration counts. Later versions solved these issues and were considerably simplified, while still producing equivalent results to the LP formulation. Nevertheless, the basic performance issues still remained, as the overall ideas had not changed.

- A more advanced *incremental* graph-based algorithm that automatically discarded uninteresting subpaths as soon as possible, by defining a novel partial relation order on subpaths. This partial relation order was devised after several attempts using graph transformations and model aggregating and is justified in Appendix A.

Interestingly, this version does not have the performance problems of the previous versions. While the worst case could still have exponential cost, the observed average case is much more manageable and has below-exponential cost. Additional studies show that performance is better with stricter time limits, but not exceedingly so.

After performing theoretical and empirical studies on the above algorithms, the next step was to standardise the *ad hoc* notation that was being used. The throughput inference algorithm and the advanced incremental graph-based algorithm were ported from the SODM service process and service composition models to standard UML activity models developed with the Eclipse Papyrus editor, annotated with a selected subset of the Generic Quantitative Analysis Modelling (GQAM) subpackage of the OMG Modelling and Analysis of Real-Time and Embedded Systems (MARTE) profile. During this process, the algorithms were further improved by adding support for UML structured and loop nodes.

So far, the MARTE-based models could only be used to design the performance requirements at an early stage. The next step in this Thesis was developing a general approach that used model extraction and model weaving to link the final design or implementation of the service with the original performance requirements, add the required technology-specific information and produce the first version of a set of performance test artefacts. These test artefacts could be then used to repeatedly check if a service met the requirements that were originally set. In particular, this general approach was implemented for two different target technologies and applied on a simple case study:

- Repurposing regular tests based on the popular JUnit testing framework for Java-first Web Services, using the facilities provided by the ContiPerf library.
- Generating new tests for the The Grinder performance testing framework for contract-first Web Services. In contrast with the previous implementation, this implementation can be used with Web Services written in any language, as long as they are described using a Web Services Description Language (WSDL) document. The Grinder was used instead of JMeter or PerfUnit due to its code-based approach, which was more amenable to the model-driven approach followed in the Thesis.

In addition to the The Grinder configuration files, the tool generates TestGenerator descriptions to generate the input data, Velocity templates for producing the input messages from the input data and Jython code to kickstart the use of The Grinder. Each of these intermediate results are plain text documents that can be manually customised to make them better suited to the needs of the tester. TestGenerator itself is a new domain-specific language that was implemented during this Thesis to help generate the required test data: due to size constraints, only a short description of the language has been included.

After extending SODM with the above algorithms and test generation facilities, a larger case study was conducted in a manufacturing context by applying the SODM+T methodology from start to end on a part of a ceramic tile manufacturing firm. The case study pointed to some places in which the inherited elements from SODM or the extensions themselves could be improved: for instance, the SODM WS interface models were not found to have a positive effect on productivity, and the performance inference algorithms were delayed until the final extended service composition models were produced. Nevertheless, the overall results were satisfactory: SODM+T managed to guide the modeller from a high-level description of the firm to concrete performance tests for some of the Web Services. The case study also revealed a few issues in the quality of the object model of the ISA-95 specification: missing conceptual areas (e.g. suppliers), a high degree of duplication of information and lack of attention to some issues that may highly impact data quality.

In summary, SODM+T users can now design the performance requirements of their SOA at an early stage without worrying about the performance of the inference algorithms and using readily available tools. After these models are ready and the services have been implemented by other means, the models can be reused to generate performance testing artefacts for Java-first or contract-first Web Services, further increasing their value.

It is important to note that in addition to the contributions directly related to the present Thesis, the Epsilon model management framework used to implement SODM+T was also improved in many large and small ways as issues and opportunities emerged. After participating in the Epsilon community since 2009, the author of the present Thesis was invited to become an official Epsilon developer in 2011 and has continued sending contributions ever since (more than 380 commits to date), such as better tooling for unit testing and debugging Epsilon-based model management tasks, improving the Eugenia graphical editor generator, updating the ModeLink model weaving tools and refining some aspects of model validation and in-place transformations in Epsilon. These contributions have not only benefited SODM+T, but also the increasingly large community of users behind Epsilon.

8.2 Future work

The author intends to continue the work presented in this Thesis in several areas. Some of this work is already underway at an early stage. Some of these areas are:

- Extending the case study in this Thesis to cover a larger subset of the described enterprise and plant management activities. Due to space limitations, the current case study only covered a specific WS within a certain holon. Extending the case study to multiple holons and multiple WS should be easier now with the insights gained from implementing the ISA-95 object model.
- Defining an improved notation for describing manufacturing processes that would simplify the data entry into a ISA-95 based system such as the one implemented for the case study in this Thesis. Manually introducing each step and relating it to the others was found to be a cumbersome and error-prone activity which would be highly amenable to computer-based modelling.

- Providing additional guidelines for defining the minimum time limits and weights used in the performance annotations. While some general rules have been proposed in the relevant chapters, a step-by-step methodology would greatly simplify the process for potential users.
- Adding a higher level of autonomy and intelligence to some of the holons, by integrating Enterprise Service Buses, Complex Event Processing and/or intelligent agent technologies into the service-based holons that require it. Again, this is the opposite approach to what is usually done by the artificial intelligence community: grafting service-oriented capabilities to the agents.

In particular, a quick survey of the field shows ASEME to be a strong contender for a lightweight intelligent agent methodology that could potentially be combined with SODM+T.

- Supporting functional testing in addition to non-functional testing, as was originally suggested when SODM was selected as a base technology. These functional tests required descriptions of the intended behaviour of the services: originally, WSCoL was selected as a good candidate notation.

During a recent 3-month stay, the WSCoL language was updated to a recent Web Service composition engine and rebased on top of the standard W3C XQuery 1.0 language to solve some important issues in its implementation. With a proper notation for service contracts, the next step is using the contracts to derive or evaluate test cases through several testing techniques in the literature.

8.3 Publications

The publications derived from the work in the present Thesis are listed below, from most recent to least recent. Some of the publications are under review and are noted as such.

In addition to the publications, the Eclipse-based tools developed during the present Thesis have been published as open source under the Eclipse Public License (EPL), and are available from the SODM+T website at <http://neptuno.uca.es/~sodmt>.

8.3.1 Journal articles

- Under review: A. García-Domínguez, I. Medina-Bulo and M. Marcos Bárcena, “Early inference of performance requirements in Web Service composition models”, *Information and Software Technology* (JCR 2012: 1.522), ISSN 0950-5849.
- Under review: Dimitrios S. Kolovos, A. García-Domínguez, Louis M. Rose and Richard F. Paige, “Towards Automated Generation of Graphical Model Editors from Annotated Metamodels”, *Software and Systems Modeling* (JCR 2012: 1.250), ISSN 1619-1374.
- L. Prades, F. Romero, A. Estruch, A. García-Domínguez and J. Serrano, “Defining a Methodology to Design and Implement Business Process Models in BPMN

According to the Standard ANSI/ISA-95 in a Manufacturing Enterprise”, *Procedia Engineering*, no. 63, p. 115-122, September 2013, ISSN 1877-7058. DOI: 10.1016/j.proeng.2013.08.283.

- A. García-Domínguez, M. Marcos-Bárcena, I. Medina-Bulo and L. Prades-Martell, “Towards an Integrated SOA-based Architecture for Interoperable and Responsive Manufacturing Systems”, *Procedia Engineering*, no. 63, p. 123–132, September 2013, ISSN 1877-7058. DOI: 10.1016/j.proeng.2013.08.268.
- A. García-Domínguez, I. Medina-Bulo and M. Marcos Bárcena, “Performance Test Case Generation for Java and WSDL-based Web Services from MARTE”, *International Journal on Advances in Internet Technology*, vol. 5, no. 3–4, p. 173–185, December 2012, ISSN 1942-2652.

This is a considerably extended version of the early work presented in ICIW 2012, after implementing the general approach outlined in the article on top of two different technologies.

- 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, p. 1–6, February 2012, ISSN 1013-9826.

This is an improved version of the MESIC 2011 paper, which was selected among the papers from the conference.

- 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* 204, p. 62–68, ISSN 0211-2124.

This is an extended and revised version of the JSWEB 2009 paper, which was selected among the papers in the conference to be part of a special number on Web Services and SOA.

8.3.2 Conference papers

ICIW 2012 A. García-Domínguez, I. Medina-Bulo and M. Marcos-Bárcena, “An Approach for Performance Test Artefact Generation for Multiple Technologies from MARTE-Annotated Workflows”, *Proceedings of the 7th International Conference on Internet and Web Applications and Services*, p. 221–226, June 2012, Stuttgart, Germany. ISBN: 978-1-61208-200-4.

TTC 2011 Louis M. Rose, A. García-Domínguez, James R. Williams, Dimitrios S. Kolovos, Richard F. Paige and Fiona A. C. Polack, “Saying Hello World with Epsilon - A Solution to the 2011 Instructive Case”, *Electronic Proceedings in Theoretical Computer Science*, vol. 74, p. 332–339, November 2011, Zurich, Switzerland. DOI: 10.4204/EPTCS.74.27.

MoDELS 2011 A. García-Domínguez, Dimitrios S. Kolovos, Louis M. Rose, Richard F. Paige and I. Medina-Bulo, “EUnit: a Unit Testing Framework for Model Management Tasks”, *Proceedings of the ACM/IEEE 14th International Conference on Model*

Driven Engineering Languages and Systems, vol. 6981/2011, p. 395–409, October 2011, Wellington, New Zealand. DOI: 10.1007/978-3-642-24485-8_29.

MESIC 2011 A. García-Domínguez, M. Marcos-Bárcena and I. Medina-Bulo, “A Comparison of BPMN 2.0 with Other Notations for Manufacturing Processes”, Proceedings of the 4th Manufacturing Engineering Society International Conference, vol. 1431, p. 593–600, September 2011, Cádiz, Spain. DOI: 10.1063/1.4707613.

ICSOFT 2011 A. García-Domínguez, I. Medina-Bulo and M. Marcos-Bárcena, “Model-driven Design of Performance Requirements with UML and MARTE”, Proceedings of the 6th International Conference on Software and Data Technologies, vol. 2, p. 54–63, July 2011, Seville, Spain. ISBN 978-989-8425-77-5.

QSIC 2011 A. García-Domínguez, I. Medina-Bulo and M. Marcos-Bárcena, “Model-Driven Design of Performance Requirements”, Proceedings of the 11th International Conference on Quality Software, p. 76–85, July 2011, Madrid, Spain. DOI: 10.1109/QSIC.2011.16.

IPROMS 2010 A. García-Domínguez, I. Medina-Bulo and M. Marcos-Bárcena, “Inference and propagation of performance constraints from abstract to concrete business workflows”, Proceedings of the Sixth Virtual International Conference of the EU FP6 I*PROMS Network of Excellence, November 2010.

JISBD 2010 A. García-Domínguez, I. Medina-Bulo and M. Marcos-Bárcena, “SODM+T: Inferencia de restricciones de rendimiento”, Actas de las XV Jornadas de Ingeniería del Software y Bases de Datos, p. 103–106, September 2010, Valencia, Spain. ISBN 978-84-92812-51-6.

MoSE 2010 A. García-Domínguez, I. Medina-Bulo and M. Marcos-Bárcena. “Inference of performance constraints in Web Service composition models”, CEUR Workshop Proceedings of the 2nd International Workshop on Model-Driven Service Engineering, vol. 608, p. 55–66, June 2010, Málaga, Spain. ISSN 1613-0073.

MESIC 2009 I. Medina-Bulo, A. García-Domínguez, F. Aguayo, L. Sevilla and M. Marcos-Bárcena. “Proposal of a Methodology for Implementing a Service-Oriented Architecture in Distributed Manufacturing Systems”, Proceedings of the 3rd Manufacturing Engineering Society International Conference, AIP Conference Proceedings, vol. 1181, no. 622, p. 622–632, November 2009, Alcoy, Spain. ISBN 978-0-7354-0722-0.

JSWEB 2009 A. García-Domínguez, I. Medina-Bulo, M. Marcos-Bárcena. “Hacia la Integración de Técnicas de Pruebas en Metodologías Dirigidas por Modelos para SOA”, Actas de las V Jornadas en Servicios Web y SOA, p. 167–180, October 2009, Madrid, Spain. ISBN 987-84-692-6832-2.

8.3.3 Book chapters

- A. Jiménez-Rielo, D. Granada and A. García-Domínguez, Eugenia, in “Desarrollo de software dirigido por modelos: conceptos, métodos y herramientas”, Ra-Ma, 2013. ISBN 978-84-9964-215-4.

- 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, in “Software and Data Technologies”, Communications in Computer and Information Science, Springer Berlin Heidelberg, no. 303, p. 136–150, January 2013. ISBN 978-3-642-36176-0.

A

Related proofs

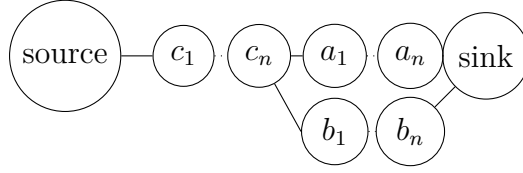


Figure A.1. General situation when comparing two paths $p_a = \{a_1, \dots, a_n, \text{sink}\}$ and $p_b = \{b_1, \dots, b_n, \text{sink}\}$: the ancestors $p_c = \{\text{source}, c_1, \dots, c_n\}$ are not known *a priori*

This appendix is dedicated to collecting some of the long proofs that are used throughout the text. The proofs were moved here in order to simplify the reading of the original chapters, while preserving these important details.

A.1 Path ordering simplification

If all paths between the source and the sinks need to complete in $L > 0$ seconds, Section 5.4.2 defined the set of valid node and path constraints as:

$$C(L) = \{(m, w) \mid m \in [0, L], w \in \mathbb{R}_0^+\} \quad (\text{A.1})$$

The elements of $C(L)$ must be ordered to know which constraints are always stricter than others.

Roughly speaking, $c(p_a)$ is less or just as strict as $c(p_b)$ if the slack assigned per unit of weight in any path of the form $p_c + p_a$ is greater or equal than that in $p_c + p_b$, for any time limit $t \leq L$ and any path p_c formed by common ancestors from p_a and p_b , ensuring that $w(p_c + p_a), w(p_c + p_b) > 0$. Figure A.1 illustrates the idea.

It would be very inefficient to actually test for all $p_c + p_a$ and $p_c + p_b$ in the graph. Instead, any possible $c(p_c) = (x, y) \in C(L)$ for which $c(p_c + p_a), c(p_c + p_b) \in C(L)$ will be considered. $\preceq_{s(L)}$ could be defined as:

$$\begin{aligned} (a, b) &\preceq_{s(L)} (c, d) \\ \Leftrightarrow &\forall t \in [0, L] \forall x \in [0, L] \forall y \in \mathbb{R}_0^+ \\ &b + y, d + y > 0 \wedge t - (a + x), t - (c + x) \geq 0 \\ \Rightarrow &\frac{t - (a + x)}{b + y} \geq \frac{t - (c + x)}{d + y} \end{aligned} \quad (\text{A.2})$$

However, the definition in (A.2) is too abstract to be useful in an actual algorithm. A simpler predicate must be derived from it:

$$\begin{aligned} &\forall t \in [0, L] \forall x \in [0, L] \forall y \in \mathbb{R}_0^+ \\ &b + y, d + y > 0 \wedge t - (a + x), t - (c + x) \geq 0 \\ \Rightarrow &\frac{t - (a + x)}{b + y} \geq \frac{t - (c + x)}{d + y} \\ \Leftrightarrow &\forall t \in [0, L] \forall x \in [0, L] \forall y \in \mathbb{R}_0^+ \\ &b + y, d + y > 0 \wedge t - x \geq a, t - x \geq c \\ \Rightarrow &\frac{t - (a + x)}{b + y} \geq \frac{t - (c + x)}{d + y} \end{aligned}$$

$$\begin{aligned}
&\Leftrightarrow \forall t \in [0, L] \forall x \in [0, L] \forall y \in \mathbb{R}_0^+ \\
&\quad b + y, d + y > 0 \wedge t - x \geq a, t - x \geq c \\
&\quad \Rightarrow (t - a - x)(d + y) \geq (t - (c + x))(b + y) \\
&\Leftrightarrow \forall t \in [0, L] \forall x \in [0, L] \forall y \in \mathbb{R}_0^+ \\
&\quad b + y, d + y > 0 \wedge t - x \geq a, t - x \geq c \\
&\quad \Rightarrow dt - ad - dx + yt - ay - xy \geq bt - bc - bx + yt - cy - xy \\
&\Leftrightarrow \forall t \in [0, L] \forall x \in [0, L] \forall y \in \mathbb{R}_0^+ \\
&\quad b + y, d + y > 0 \wedge t - x \geq a, t - x \geq c \\
&\quad \Rightarrow dt - ad - dx - ay \geq bt - bc - bx - cy \\
&\Leftrightarrow \forall t \in [0, L] \forall x \in [0, L] \forall y \in \mathbb{R}_0^+ \\
&\quad b + y, d + y > 0 \wedge t - x \geq \max\{a, c\} \\
&\quad \Rightarrow dt - bt - dx + bx \geq ad - bc + ay - cy \\
&\Leftrightarrow \forall t \in [0, L] \forall x \in [0, L] \forall y \in \mathbb{R}_0^+ \\
&\quad b + y, d + y > 0 \wedge t - x \geq \max\{a, c\} \\
&\quad \Rightarrow (d - b)t - (d - b)x \geq ad - bc + (a - c)y \\
&\Leftrightarrow \forall t \in [0, L] \forall x \in [0, L] \forall y \in \mathbb{R}_0^+ \\
&\quad b + y, d + y > 0 \wedge t - x \geq \max\{a, c\} \\
&\quad \Rightarrow (d - b)(t - x) \geq ad - bc + (a - c)y
\end{aligned}$$

t and x have been moved to one side of the inequation and y to the other. Since \geq for \mathbb{R} is transitive, it is only necessary to find the conditions required so

$$\begin{aligned}
&\min\{(d - b)(t - x) \mid 0 \leq t, x \leq L \wedge t - x \geq c\} \\
&\geq \max\{ad - bc + (a - c)y \mid y \in \mathbb{R}_0^+\}
\end{aligned}$$

In order for $ad - bc + (a - c)y$ to have an upper bound, it must be that $a \leq c$, as y can grow indefinitely. In that case, the maximum is equal to $ad - bc$, for $y = 0$.

Computing $\min\{(d - b)(t - x)\}$ requires considering the sign of $d - b$ and the new assumption, $a \leq c$, which is equivalent to $\max\{a, c\} = c$:

$$\min\{(d - b)(t - x)\} = \begin{cases} (d - b) \min\{t - x\} = (d - b)c & d - b \geq 0 \\ (d - b) \max\{t - x\} = (d - b)L & \text{otherwise.} \end{cases}$$

Collecting the previous results, the relation $\preceq_{s(L)}$ for this case is defined as:

$$\begin{aligned}
(a, b) \preceq_{s(L)} (c, d) &\Leftrightarrow a \leq c \wedge (d \geq b \wedge (d - b)c \geq ad - bc \\
&\quad \vee d < b \wedge (d - b)L \geq ad - bc)
\end{aligned} \tag{A.3}$$

Equation (A.3) can be simplified further when $d \geq b$: $(d - b)c \geq ad - bc$ is equivalent to $dc - bc \geq ad - bc$, $cd \geq ad$, $c \geq a$ and finally $a \leq c$, which is checked in a previous term. Therefore, this term can be removed. Cleaning up some other terms to improve readability results in the following definition, equivalent to the previous one:

$$(a, b) \preceq_{s(L)} (c, d) \Leftrightarrow a \leq c \wedge (b \leq d \vee b > d \wedge (b - d)L \leq bc - ad) \tag{A.4}$$

The predicate in (A.4) implies the predicate in (A.2), by construction. However, this means that all pairs (L, x) and (L, y) are considered just as strict regardless of their weights, which is unintuitive and may hinder early validation of the constraints. Equation (A.4) will be revised to forbid that $(L, b) \preceq_{s(L)} (L, c)$ when $b > c$:

$$(a, b) \preceq_{s(L)} (c, d) \Leftrightarrow a \leq c \wedge (b \leq d \vee a < L \wedge b > d \wedge (b - d)L \leq bc - ad) \quad (\text{A.5})$$

The definition in (A.5) can be simplified one step further, as $b > d$ is simply the negation of the $b \leq d$:

$$(a, b) \preceq_{s(L)} (c, d) \Leftrightarrow a \leq c \wedge (b \leq d \vee a < L \wedge (b - d)L \leq bc - ad) \quad (\text{A.6})$$

The version in A.6 is the one used in the final algorithm.

A.2 Path ordering as a partial order

In the previous section, the original definition of $\preceq_{s(L)}$ was simplified. This section will prove that $\preceq_{s(L)}$ is a partial order on $C(L)$.

Lemma 1. $\preceq_{s(L)}$ is reflexive in $C(L)$.

Proof. Let $(a, b) \in C(L)$. It is trivially true that $a \leq a$ and $b \leq b$. Therefore, $a \leq a \wedge b \leq b$ holds and, by definition of $\preceq_{s(L)}$, $(a, b) \preceq_{s(L)} (a, b)$. \square

Lemma 2. $\preceq_{s(L)}$ is antisymmetric in $C(L)$.

Proof.

$$\begin{aligned} & (a, b) \preceq_{s(L)} (c, d) \wedge (c, d) \preceq_{s(L)} (a, b) \\ \Rightarrow & a \leq c \wedge (b \leq d \vee a < L \wedge b > d \wedge (b - d)L \leq bc - ad) \\ & \wedge c \leq a \wedge (d \leq b \vee c < L \wedge d > b \wedge (d - b)L \leq ad - bc) \\ \Rightarrow & a = c \wedge (b \leq d \vee a < L \wedge b > d \wedge (b - d)L \leq bc - ad) \\ & \wedge (d \leq b \vee c < L \wedge d > b \wedge (d - b)L \leq ad - bc) \\ \Rightarrow & a = c \wedge (b \leq d \vee a < L \wedge b > d \wedge (b - d)L \leq ba - ad) \\ & \wedge (d \leq b \vee a < L \wedge d > b \wedge (d - b)L \leq ad - ba) \\ \Rightarrow & a = c \wedge (b \leq d \vee a < L \wedge b > d \wedge (b - d)L \leq (b - d)a) \\ & \wedge (d \leq b \vee a < L \wedge d > b \wedge (d - b)L \leq (d - b)a) \\ \Rightarrow & a = c \wedge (b \leq d \vee a < L \wedge b > d \wedge L \leq a) \\ & \wedge (d \leq b \vee a < L \wedge d > b \wedge L \leq a) \\ \Rightarrow & a = c \wedge (b \leq d \vee \perp) \wedge (d \leq b \vee \perp) \\ \Rightarrow & a = c \wedge b \leq d \wedge d \leq b \\ \Rightarrow & a = c \wedge b = d \\ \Rightarrow & (a, b) = (c, d) \end{aligned}$$

\square

Lemma 3. $\preceq_{s(L)}$ is transitive in $C(L)$.

Proof. Expanding $\preceq_{s(L)}$ results in a rather long formula:

$$\begin{aligned}
& (a, b) \preceq_{s(L)} (c, d) \wedge (c, d) \preceq_{s(L)} (e, f) \\
& \Rightarrow a \leq c \wedge (b \leq d \vee a < L \wedge b > d \wedge (b - d)L \leq bc - ad) \\
& \quad \wedge c \leq e \wedge (d \leq f \vee c < L \wedge d > f \wedge (d - f)L \leq de - cf) \\
& \Rightarrow a \leq c \leq e \\
& \quad \wedge (b \leq d \wedge d \leq f \\
& \quad \vee b \leq d \wedge c < L \wedge d > f \wedge (d - f)L \leq de - cf \\
& \quad \vee a < L \wedge b > d \wedge (b - d)L \leq bc - ad \wedge d \leq f \\
& \quad \vee a < L \wedge b > d \wedge (b - d)L \leq bc - ad \\
& \quad \wedge c < L \wedge d > f \wedge (d - f)L \leq de - cf)
\end{aligned}$$

It can already be concluded that $a \leq e$, using the transitivity of \leq in \mathbb{R} . However, the rest of the formula is rather complicated. This predicate will now be converted into Disjunctive Normal Form and then each of its 4 conjunctive clauses will be proved to imply $(a, b) \preceq_{s(L)} (c, d)$ separately.

1. The first conjunctive clause is

$$a \leq c \leq e \wedge b \leq d \wedge d \leq f$$

Since \leq in \mathbb{R} is transitive, $b \leq f$. It is already known that $a \leq e$, so it can be concluded that $(a, b) \preceq_{s(L)} (e, f)$ in this case.

2. The second conjunctive clause is

$$a \leq c \leq e \wedge b \leq d \wedge c < L \wedge d > f \wedge (d - f)L \leq de - cf$$

This conjunctive clause relates b with d and d with f , but not b with f . To solve this issue, the trivially true term $b \leq f \vee b > f$ will be added. Converting the resulting expression into Disjunctive Normal Form produces 2 conjunctive clauses:

- a) The first one includes $b \leq f$, immediately implying that $(a, b) \preceq_{s(L)} (e, f)$, since it is already known that $a \leq e$.
- b) The second one includes $b > f$. As $c < L$ and $a \leq c$, concluding that $a < L$. Only $(f - b)L \leq be - af$ needs to be proved, following these steps:

$$\begin{aligned}
& a \leq c \wedge b \leq d \wedge (e, f) \in C(L) \wedge (d - f)L \leq de - cf \\
& \Rightarrow a \leq c \wedge b \leq d \wedge e \leq L \wedge (d - f)L \leq de - cf \\
& \Rightarrow a \leq c \wedge b \leq d \wedge e - L \leq 0 \wedge (d - f)L \leq de - cf \\
& \Rightarrow a \leq c \wedge b(e - L) \geq d(e - L) \wedge (d - f)L \leq de - cf \\
& \Rightarrow a \leq c \wedge b(e - L) \geq d(e - L) \wedge -fL \leq d(e - L) - cf \\
& \Rightarrow a \leq c \wedge -fL \leq b(e - L) - cf \\
& \Rightarrow a \leq c \wedge bL - fL \leq be - cf \\
& \Rightarrow a \leq c \wedge (b - f)L \leq be - cf \\
& \Rightarrow (b - f)L \leq be - af
\end{aligned}$$

3. The third conjunctive clause is

$$a \leq c \leq e \wedge a < L \wedge b > d \wedge (b - d)L \leq bc - ad \wedge d \leq f$$

This has the same problem as in the previous clause. It can be solved it in the same way, by adding the trivially true term $b \leq f \vee b > f$, converting into Disjunctive Normal Form and studying each conjunctive clause:

- a) Again, the first one includes $b \leq f$, implying that $(a, b) \preceq_{s(L)} (e, f)$.
- b) The second one includes $a < L$ and $b > f$. It is already known that $a \leq e$, so it is only necessary to prove that $(b - f)L \leq be - af$. The process is as follows:

$$\begin{aligned}
& c \leq e \wedge a < L \wedge d \leq f \wedge (b - d)L \leq bc - ad \\
& \Rightarrow c \leq e \wedge L - a > 0 \wedge d \leq f \wedge (b - d)L \leq bc - ad \\
& \Rightarrow c \leq e \wedge d(L - a) \leq f(L - a) \wedge (b - d)L \leq bc - ad \\
& \Rightarrow c \leq e \wedge d(L - a) \leq f(L - a) \wedge bL - dL \leq bc - ad \\
& \Rightarrow c \leq e \wedge d(L - a) \leq f(L - a) \wedge bL \leq bc + (L - a)d \\
& \Rightarrow c \leq e \wedge bL \leq bc + (L - a)f \\
& \Rightarrow bL \leq be + (L - a)f \\
& \Rightarrow bL - fL \leq be - af \\
& \Rightarrow (b - f)L \leq be - af
\end{aligned}$$

4. The fourth conjunctive clause is

$$\begin{aligned}
& a < L \wedge b > d \wedge (b - d)L \leq bc - ad \\
& \wedge c < L \wedge d > f \wedge (d - f)L \leq de - cf
\end{aligned}$$

It is already known that $a < L$ and since $b > d \wedge d > f$, it can be concluded that $b > f$. $(b - f)L \leq be - af$ has to be proved. The steps required are as follows:

$$\begin{aligned}
& a, c < L \wedge b, d > f \wedge (e, f) \in C(L) \\
& \wedge (b - d)L \leq bc - ad \wedge (d - f)L \leq de - cf \\
& \Rightarrow L - a, L - c > 0 \wedge b, d > f \wedge (e, f) \in C(L) \\
& \wedge (b - d)L \leq bc - ad \wedge (d - f)L \leq de - cf \\
& \Rightarrow L - a, L - c > 0 \wedge b, d > f \wedge f \geq 0 \\
& \wedge (b - d)L \leq bc - ad \wedge (d - f)L \leq de - cf \\
& \Rightarrow L - a, L - c > 0 \wedge b, d > 0 \\
& \wedge (b - d)L \leq bc - ad \wedge (d - f)L \leq de - cf \\
& \Rightarrow L - a, L - c > 0 \wedge b, d > 0 \\
& \wedge bL - dL \leq bc - ad \wedge dL - fL \leq de - cf \\
& \Rightarrow L - a, L - c > 0 \wedge b, d > 0 \\
& \wedge bL - bc \leq dL - ad \wedge dL - de \leq fL - cf \\
& \Rightarrow L - a, L - c > 0 \wedge b, d > 0 \\
& \wedge b(L - c) \leq d(L - a) \wedge d(L - e) \leq f(L - c) \\
& \Rightarrow L - a, L - c > 0 \wedge b, d > 0 \\
& \wedge (L - c)/(L - a) \leq d/b \wedge (L - e)/(L - c) \leq f/d
\end{aligned}$$

$$\begin{aligned}
&\Rightarrow \frac{(L-c)(L-e)}{(L-a)(L-c)} \leq \frac{df}{bd} \\
&\Rightarrow \frac{L-e}{L-a} \leq \frac{f}{b} \\
&\Rightarrow (L-e)b \leq (L-a)f \\
&\Rightarrow bL - fL \leq be - af \\
&\Rightarrow (b-f)L \leq be - af
\end{aligned}$$

□

Theorem 1. *The minimum element in $C(L)$ according to $\preceq_{s(L)}$ is $(0, 0)$.*

Proof. From the definition of $C(L)$, it is known that for every $(a, b) \in C(L)$, $0 \leq a \wedge 0 \leq b$ is true, and therefore $(0, 0) \preceq_{s(L)} (a, b)$. It will now be proved that $(a, b) \in C(L) - \{(0, 0)\} \wedge (c, d) = (0, 0) \Rightarrow (a, b) \not\preceq_{s(L)} (c, d)$:

- If $a > 0$, $a \not\leq c$ and so $(a, b) \not\preceq_{s(L)} (c, d)$.
- If $b > 0$, $b \not\leq d$. It is true that $b > d$, but simplifying $(b-d)L \leq bc - ad$ in this case results in $bL \leq 0$, which is a contradiction, as $b, L > 0$.

Since $(0, 0) \preceq_{s(L)} (a, b)$ for all $(a, b) \in C(L)$ and $(a, b) \not\preceq_{s(L)} (0, 0)$ for all $(a, b) \in C(L) - \{(0, 0)\}$, it follows that $(0, 0)$ is the minimum element of $\preceq_{s(L)}$ in $C(L)$. □

Theorem 2. *$\preceq_{s(L)}$ defines a partial order on $C(L)$.*

Proof. From Lemmas 1, 2 and 3, it can be concluded that $\preceq_{s(L)}$ is reflexive, antisymmetric and transitive in $C(L)$. □

B

The Epsilon EUnit testing
framework

EUnit is an unit testing framework specifically developed during this work to test model management tasks, based on the Epsilon Object Language (EOL) and the Epsilon Ant workflow tasks. It provides assertions for comparing models, files and directories. Tests can be reused with different sets of models and input data, and differences between the expected and actual models can be graphically visualised. It is available as open source from the official Epsilon website¹.

This chapter discusses the motivation behind EUnit, describes how tests are organized and written and shows two examples of how a model-to-model transformation can be tested with EUnit. This is followed by a discussion of how EUnit can be extended to support other modelling and model management technologies. Finally, two cases studies are shown on its usage for performing regression testing for the EuGENia graphical model editor generator and for performing unit testing for the SODM with Testing (SODM+T) performance inference algorithms of Chapter 5.

B.1 Motivation

Model-driven approaches are being adopted in a wide range of demanding environments, such as finance, health care or telecommunications [9]. In this context, validation and verification is identified as one of the many challenges of Model Driven Software Engineering (MDSE) [21].

MDSE in practice involves creating models, and thereafter *managing* them, via various tasks, such as model transformation, validation and merging. The validation and verification of each type of model management task has its own specific challenges. Kolovos et al. list testing concerns for Model to Model (M2M) and Model to Text (M2T) transformations, model validations, model comparisons and model compositions in [13]. Baudry et al. identify three main issues when testing model transformations [2]: the complexity of the input and output models, the immaturity of the model management environments and the large number of different transformation languages and techniques.

B.1.1 Common issues

While each type of model management task does have specific complexity, some of the concerns raised by Baudry can be generalized to apply to all model management tasks:

- There is usually a large number of models to be handled. Some may be created by hand, some may be generated using hand-written programs, and some may be generated automatically following certain coverage criteria.
- A single model or set of models may be used in several tasks. For instance, a model may be validated before performing an in-place transformation to assist the user, and later on it may be transformed to another model or merged with a different model. This requires having at least one test for each valid combination of models and sets of tasks.
- Test oracles are more complex than in traditional unit testing [16]: instead of checking scalar values or simple lists, entire graphs of model objects or file trees may have to

¹<http://eclipse.org/epsilon>

be compared. In some cases, complex properties in the generated artifacts may have to be checked.

- Models and model management tasks may use a wide range of technologies. Models may be based on Ecore [20], XML files or Java object graphs, among many others. At the same time, tasks may use technologies from different platforms, such as Epsilon, openArchitectureWare (oAW) [10] or the ATLAS Model Management Architecture (AMMA) [5]. Many of these technologies offer high-level tools for running and debugging the different tasks using several models. However, users wishing to do automated unit testing need to learn low-level implementation details about their modelling and model management technologies. This increases the initial cost of testing these tasks and hampers the adoption of new technologies.
- Existing testing tools tend to focus on the testing technique itself, and lack integration with external systems. Some tools provide graphical user interfaces, but most do not generate reports which can be consumed by a continuous integration server, for instance.

B.1.2 Testing with JUnit

The previous issues are easier to understand with a concrete example. This section shows how a simple transformation between two Eclipse Modeling Framework (EMF) models in the Epsilon Transformation Language (ETL) using JUnit 4 [3] would be normally tested, and points out several issues due to JUnit's limitations as a general-purpose unit testing framework for Java programs.

For the sake of brevity, only an outline of the JUnit test suite is included. All JUnit test suites are defined as Java classes. This test suite has three methods:

1. The test setup method (marked with the `@Before` JUnit annotation) loads the required models by creating and configuring instances of *EmfModel*. After that, it prepares the transformation by creating and configuring an instance of *EtlModule*, adding the input and output models to its model repository.
2. The test case itself (marked with `@Test`) runs the ETL transformation and uses the generic comparison algorithm implemented by EMF Compare to perform the model comparison.
3. The test teardown method (marked with `@After`) disposes of the models.

Several issues can be identified in each part of the test suite. First, test setup is tightly bound to the technologies used: it depends on the Application Programming Interface (API) of the *EmfModel* and *EtlModule* classes, which are both part of Epsilon. Later refactorings in these classes may break existing tests.

The test case can only be used for a single combination of input and output models. Testing several combinations requires either repeating the same code and therefore making the suite less maintainable, or using parametric testing, which may be wasteful if not all tests need the same combinations of models.

Model comparison requires the user to manually select a model comparison engine and integrate it with the test. For comparing EMF models, EMF Compare is easy to use and

readily available. However, generic model comparison engines may not be available for some modelling technologies, or may be harder to integrate.

Finally, instead of comparing the obtained and expected models, several properties could have been checked in the obtained model. However, querying models through Java code can be quite verbose.

B.1.3 Selected approach

Several approaches could be followed to address these issues. A first instinct would be to extend JUnit and reuse all the tooling available for it. A custom test runner would simplify setup and teardown, and modelling platforms would integrate their technologies into it. Since Java is very verbose when querying models, the custom runner should run tests in a higher-level language, such as EOL. However, JUnit is very tightly coupled to Java, and this would impose limits on the attainable level of integration. For instance, errors in the model management tasks or the EOL tests could not be reported from their original source, but rather from the Java code which invoked them. Another problem with this approach is that new integration code would need to be written for each of the existing platforms.

Alternatively, a new language exclusively dedicated to testing could be added to the Epsilon family. Being based on EOL, model querying would be very concise, and with a test runner written from scratch, test execution would be very flexible. However, this would still require all platforms to write new code to integrate with it, and this code would be tightly coupled to Epsilon.

As a middle ground, EOL could be decorated to guide its execution through a new test runner, while reusing the Apache Ant [1] tasks already provided by several of the existing platforms, such as AMMA or Epsilon. Like Make, Ant is a tool focused on automating the execution of processes such as program builds. Unlike Make, Ant defines processes using XML *buildfiles* with sets of interrelated *targets*. Each target contains in turn a sequence of *tasks*. Many Ant tasks and Ant-based tools already exist, and it is easy to create a new Ant task.

Among these three approaches, EUnit follows the last one. Ant tasks take care of model setup and management, and tests are written in EOL and executed by a new test runner, written from the ground up.

B.2 Test organisation

EUnit has a rich data model: test suites are organized as trees of tests, and each test is divided into many parts which can be extended by the user. This section is dedicated to describing how test suites and tests are organized. The next section indicates how they are written.

B.2.1 Test suites

EUnit test suites are organized as trees: inner nodes group related test cases and define *data* bindings. Leaf nodes define *model* bindings and run the test cases.

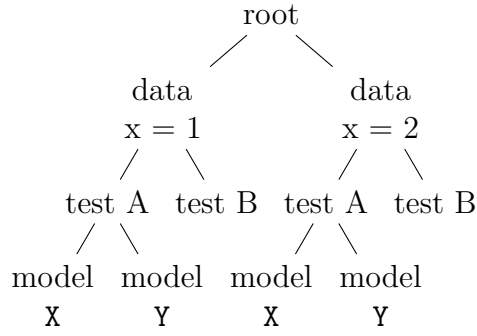


Figure B.1. Example of an EUnit test tree

Data bindings repeat all test cases with different values in one or more variables. They can implement parametric testing, as in JUnit 4. EUnit can nest several data bindings, running all test cases once for each combination.

Model bindings are specific to EUnit: they allow developers to repeat a single test case with different subsets of models. Data and model bindings can be combined. One interesting approach is to set the names of the models to be used in the model binding from the data binding, as a quick way to try several test cases with the same subsets of models.

Figure B.1 shows an example of an EUnit test tree: nodes with data bindings are marked with **data**, and nodes with model bindings are marked with **model**. EUnit will perform a preorder traversal of this tree, running the following tests:

1. *A* with $x = 1$ and model X.
2. *A* with $x = 1$ and model Y.
3. *B* with $x = 1$ and both models.
4. *A* with $x = 2$ and model X.
5. *A* with $x = 2$ and model Y.
6. *B* with $x = 2$ and both models.

Optionally, EUnit can filter tests by name, running only *A* or *B*. Similarly to JUnit, EUnit logs start and finish times for each node in the tree, so the most expensive test cases can be quickly detected. However, EUnit logs CPU time² in addition to the usual wallclock time.

Parametric testing is not to be confused with *theories* [18]: both repeat a test case with different values, but results are reported quite differently. While parametric testing produces separate test cases with independent results, theories produce aggregated tests which only pass if the original test case passes for every data point. Figure B.2 illustrates these differences. EUnit does not support theories yet: however, they can be approximated with data bindings.

²CPU time only measures the time elapsed in the thread used by EUnit, and is more stable with varying system load in single-threaded programs.

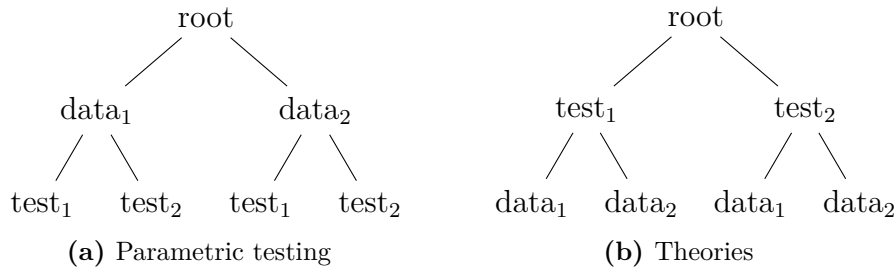


Figure B.2. Comparison between parametric testing and theories

B.2.2 Test cases

The execution of a test case is divided into the following steps:

1. Apply the data bindings of its ancestors.
2. Run the model setup sections defined by the user.
3. Apply the model bindings of this node.
4. Run the regular setup sections defined by the user.
5. Run the test case itself.
6. Run the teardown sections defined by the user.
7. Tear down the data bindings and models for this test.

An important difference between JUnit and EUnit is that setup is split into two parts: model setup and regular setup. This split allows users to add code before and after model bindings are applied. Normally, the model setup sections will load all the models needed by the test suite, and the regular setup sections will further prepare the models selected by the model binding. Explicit teardown sections are usually not needed, as models are disposed automatically by EUnit. EUnit includes them for consistency with the xUnit frameworks.

Due to its focus on model management, model setup in EUnit is very flexible. Developers can combine several ways to set up models, such as model references, individual Apache Ant [1] tasks, Apache Ant targets or Human-Usable Textual Notation (HUTN) [17] fragments.

A test case may produce one among several results. **SUCCESS** is obtained if all assertions passed and no exceptions were thrown. **FAILURE** is obtained if an assertion failed. **ERROR** is obtained if an unexpected exception was thrown while running the test. Finally, tests may be **SKIPPED** by the user.

B.3 Test specification

The previous section described how test suites and test cases are organized. This section will show how to write them.

As discussed before, after evaluating several approaches, it was decided to combine the expressive power of EOL and the extensibility of Apache Ant. For this reason, EUnit test suites are split into two files: an Ant buildfile and an EOL script with some special-purpose annotations. The next subsections describe the contents of these two files and revisit the previous example with EUnit.

B.3.1 Ant buildfile

EUnit uses standard Ant buildfiles: running EUnit is as simple as using its Ant task. Users may run EUnit more than once in a single Ant launch: the graphical user interface will automatically aggregate the results of all test suites.

B.3.1.1 EUnit invocations

An example invocation of the EUnit Ant task using the most common features is shown in Listing B.1. Users will normally only use some of these features at a time, though. Optional attributes have been listed between brackets. Some nested elements can be repeated 0+ times (*) or 0-1 times (?). Valid alternatives for an attribute are separated with |.

Listing B.1 Format of an invocation of the EUnit Ant task

```
<epsilon.eunit src="..."
  [failOnErrors="..."]
  [package="..."]
  [toDir="..."]
  [report="yes|no"]>
  (<model      ref="OldName"  [as="NewName"] />)*
  (<uses       ref="x"        [as="y"]   />)*
  (<exports    ref="z"        [as="w"]   />)*
  (<parameter  name="myparam" value="myvalue" />)*
  (<modelTasks><!-- Zero or more Ant tasks --></modelTasks>)?
</epsilon.eunit>
```

The EUnit Ant task is based on the Epsilon abstract executable module task (see [15]), inheriting some useful features. The attribute *src* points to the path of the EOL file, and the optional attribute *failOnErrors* can be set to **false** to prevent EUnit from aborting the Ant launch if a test case fails. EUnit also inherits support for importing and exporting global variables through the *<uses>* and *<exports>* elements: the original name is set in *ref*, and the optional *as* attribute allows for using a different name. For receiving parameters as name-value pairs, the *<parameter>* element can be used.

Model references (using the *<model>* nested element) are also inherited from the Epsilon abstract executable module task. These allow model management tasks to refer by name to models previously loaded in the Ant buildfile. However, EUnit implicitly reloads the models after each test case. This ensures that test cases are isolated from each other.

The EUnit Ant task adds several new features to customize the test result reports and perform more advanced model setup. By default, EUnit generates reports in the XML format of the Ant *<junit>* task. This format is also used by many other tools, such as the TestNG unit testing framework [4], the Jenkins continuous integration server [12] or the JUnit Eclipse plug-ins. To suppress these reports, *report* must be set to *no*.

By default, the XML report is generated in the same directory as the Ant buildfile, but it can be changed with the *toDir* attribute. Test names in JUnit are formed by its Java

package, class and method: EUnit uses the filename of the EOL script as the class and the name of the EOL operation as the method. By default, the package is set to the string “default”: users are encouraged to customize it with the *package* attribute.

The optional `<modelTasks>` nested element contains a sequence of Ant tasks which will be run after reloading the model references and before running the model setup sections in the EOL file. This allows users to run workflows more advanced than simply reloading model references, such as the one in Listing B.5.

B.3.1.2 Helper targets

Ant buildfiles for EUnit may include *helper targets*. These targets can be invoked from anywhere in the EOL script using `runTarget("targetName")`. Helper targets are quite versatile: called from an EOL model setup section, they allow for reusing model loading fragments between different EUnit test suites. They can also be used to invoke the model management tasks under test. Listing B.5 shows a helper target for an ETL transformation, and listing B.9 shows a helper target for an ATL transformation.

B.3.2 EOL script

The Epsilon Object Language script is the second half of the EUnit test suite. EOL annotations are used to tag some of the operations as data binding definitions (`@data` or `@Data`), additional model setup sections (`@model/@Model`), test setup and teardown sections (`@setup/@Before` and `@teardown/@After`) and test cases (`@test/@Test`). Suite setup and teardown sections can also be defined with the `@suitesetup` and `@BeforeClass` and with the `@suiteteardown` and `@AfterClass` annotations: these operations will be run before and after all tests, respectively.

B.3.2.1 Data bindings

Data bindings repeat all test cases with different values in some variables. To define a data binding, users must define an operation which returns a sequence of elements and is marked with `@data variable`. All test cases will be repeated once for each element of the returned sequence, setting the specified variable to the corresponding element. Listing B.2 shows two nested data bindings and a test case which will be run four times: with $x=1$ and $y=5$, $x=1$ and $y=6$, $x=2$ and $y=5$ and finally $x=2$ and $y=6$. The example shows how x and y could be used by the setup section to generate an input model for the test. This can be useful if the intent of the test is ensuring that a certain property holds in a class of models, rather than a single model.

Alternatively, if both x and y were to use the same sets of values, two `@data` annotations could be added to the same operation. For instance, Listing B.3 shows 4 test cases could be specified: $x=1$ and $y=1$, $x=1$ and $y=2$, $x=2$ and $y=1$ and $x=2$ and $y=2$.

B.3.2.2 Model bindings

Model bindings repeat a test case with different subsets of models. They can be defined by annotating a test case with `$with map` or `$onlyWith map` one or more times, where `map` is an EOL expression that produces a *Map*. For each key-value pair $key = value$, EUnit will rename the model named *value* to *key*. The difference between `$with` and `$onlyWith`

Listing B.2 Example of a 2-level data binding

```
@data x
operation firstLevel() { return 1.to(2); }

@data y
operation secondLevel() { return 5.to(6); }

@setup
operation generateModel() { —* generate model using x and y * }

@test
operation mytest() { —* test with the generated model * }
```

Listing B.3 Example of reusing the same operation for several data bindings

```
@data x
@data y
operation levels() { return 1.to(2); }

@setup
operation generateModel() { —* generate model using x and y * }

@test
operation mytest() { —* test with the generated model * }
```

is how they handle the models not mentioned in the *Map*: **\$with** will preserve them as is, and **\$onlyWith** will make them unavailable during the test. **\$onlyWith** is useful for tightly restricting the set of available models in a test and for avoiding ambiguous type references when handling multiple models using the same metamodel.

Listing B.4 shows two tests which will be each run twice. The first test uses **\$with**, which preserves models not mentioned in the *Map*: the first time, model **A** will be the default model and model **B** will be the **Other** model, and the second time, model **B** will be the default model and model **A** will be the **Other** model. The second test uses two **\$onlyWith** annotations: on the first run, **A** will be available as **Model** and **B** will not be available, and on the second run, only **B** will be available as **Model** and **A** will be unavailable.

Listing B.4 Examples of model bindings

```
$with Map { "" = "A", "Other" = "B" }
$with Map { "" = "B", "Other" = "A" }
@test
operation mytest() {
  —* use the default and Other models, while keeping the rest as is *
}

$onlyWith Map { "Model" = "A" }
$onlyWith Map { "Model" = "B" }
@test
operation mytest2() {
  — first time: A as 'Model', B is unavailable
  — second time: B as 'Model', A is unavailable
}
```

Signature	Description
<code>runTarget(name : String)</code>	Runs the specified target of the Ant buildfile which invoked EUnit.
<code>exportVariable(name : String)</code>	Exports the specified variable, to be used by another executable module.
<code>useVariable(name : String)</code>	Imports the specified variable, which should have been previously exported by another executable module.
<code>loadHutn(name : String, hutn : String)</code>	Loads an EMF model with the specified name, by parsing the second argument as an HUTN [17] fragment.
<code>antProject : org.apache.tools.ant.Project</code>	Global variable which refers to the Ant project being executed. This can be used to create and run Ant tasks from inside the EOL code.

Table B.1. Extra operations and variables in EUnit

B.3.2.3 Additional variables and built-in operations

EUnit provides several variables and operations which are useful for testing. These are listed in Table B.1.

B.3.2.4 Assertions

EUnit implements some common assertions for equality and inequality, with special versions for comparing floating-point numbers. EUnit also supports a limited form of exception testing with `assertError`, which checks that the expression inside it throws an exception. Custom assertions can be defined by the user with the `fail` operation, which fails a test with a custom message. The available assertions are shown in Table B.2. Table B.3 lists the available option keys which can be used with the model equality assertions, by comparator.

More importantly, EUnit implements specific assertions for comparing models, files and trees of files. Model comparison is not implemented by the assertions themselves: it is an optional service implemented by some EMC drivers. Currently, EMF models will automatically use EMF Compare as their comparison engine. The rest of the EMC drivers do not support comparison yet. The main advantage of having an abstraction layer implement model comparison as a service is that the test case definition is decoupled from the concrete model comparison engine used.

Model, file and directory comparisons take a snapshot of their operands before comparing them, so EUnit can show the differences right at the moment when the comparison was performed. This is especially important when some of the models are generated on the fly by the EUnit test suite, or when a test case for code generation may overwrite the results of the previous one.

Figure B.3 shows a screenshot of the EUnit graphical user interface. On the left, an Eclipse view shows the results of several EUnit test suites. It can be seen that the `load-models-with-hutn` suite failed. The Compare button to the right of “Failure Trace” can be pressed to show the differences between the expected and obtained models, as shown

on the right side. EUnit implements a pluggable architecture where *difference viewers* are automatically selected based on the types of the operands. There are difference viewers for EMF models, file trees and a fallback viewer which converts both operands to strings.

Table B.2. Assertions in EUnit

Signature	Description
<code>assertEqualDirectories(expectedPath : String, obtainedPath : String)</code>	Fails the test if the contents of the directory in <i>obtainedFile</i> differ from those of the directory in <i>expectedPath</i> . Directory comparisons are performed on recursive snapshots of both directories.
<code>assertEqualFiles(expectedPath : String, obtainedPath : String)</code>	Fails the test if the contents of the file in <i>obtainedPath</i> differ from those of the file in <i>expectedPath</i> . File comparisons are performed on snapshots of both files.
<code>assertEqualModels([msg : String, expectedModel : String, obtainedModel : String [, options : Map])</code>	Fails the test with the optional message <i>msg</i> if the model named <i>obtainedModel</i> is not equal to the model named <i>expectedModel</i> . Model comparisons are performed on snapshots of the resource sets of both models. During EMF comparisons, XMI identifiers are ignored. Additional comparator-specific options can be specified through <i>options</i> .
<code>assertEquals([msg : String, expected : Any, obtained : Any)</code>	Fails the test with the optional message <i>msg</i> if the values of <i>expected</i> and <i>obtained</i> are not equal.
<code>assertEquals([msg : String, expected : Real, obtained : Real, ulps : Integer)</code>	Fails the test with the optional message <i>msg</i> if the values of <i>expected</i> and <i>obtained</i> differ in more than <i>ulps</i> units of least precision. See this site for details.
<code>assertError(expr : Any)</code>	Fails the test if no exception is thrown during the evaluation of <i>expr</i> .
<code>assertFalse([msg : String, cond : Boolean)</code>	Fails the test with the optional message <i>msg</i> if <i>cond</i> is true . It is a negated version of <code>assertTrue</code> .
<code>assertLineWithMatch([msg : String, path : String, regexp : String)</code>	Fails the test with the optional message <i>msg</i> if the file at <i>path</i> does not have a line containing a substring matching the regular expression <i>regexp</i> ³ .

Continues on next page

³See *java.util.regex.Pattern* for details about the accepted syntax for regular expressions.

Continued from previous page

Signature	Description
<code>assertMatchingLine([msg : String,] path : String, regexp : String)</code>	Fails the test with the optional message <i>msg</i> if the file at <i>path</i> does not have a line that matches the regular expression <i>regexp</i> ⁴ from start to finish.
<code>assertNotEqualDirectories(expectedPath : String, obtainedPath : String)</code>	Negated version of <code>assertEqualDirectories</code> .
<code>assertNotEqualFiles(expectedPath : String, obtainedPath : String)</code>	Negated version of <code>assertEqualFiles</code> .
<code>assertNotEqualModels([msg : String,] expectedModel : String, obtainedModel : String)</code>	Negated version of <code>assertNotEqualModels</code> .
<code>assertNotEquals([msg : String,] expected : Any, obtained : Any)</code>	Negated version of <code>assertEquals([msg : String,] expected : Any, obtained : Any)</code> .
<code>assertNotEquals([msg : String,] expected : Real, obtained : Real, ulps : Integer)</code>	Negated version of <code>assertEquals([msg : String,] expected : Real, obtained : Real, ulps : Integer)</code> .
<code>assertTrue([msg : String,] cond : Boolean)</code>	Fails the test with the optional message <i>msg</i> if <i>cond</i> is false .
<code>fail(msg : String)</code>	Fails a test with the message <i>msg</i> .

B.4 Examples: testing a model transformation with EUnit

B.4.1 Models and tasks in the buildfile

After describing the basic syntax, this section will show how to use EUnit to test an ETL transformation.

The Ant buildfile is shown in Listing B.5. It has two targets: *run-tests* (lines 2–16) invokes the EUnit suite, and *tree2graph* (lines 17–22) is a helper target which transforms

⁴See footnote for `assertLineWithMatch` for details about the syntax of the regular expressions.

Comparator and key	Usage
EMF, “whitespace”	When set to “ignore”, differences in attribute values due to whitespace will be ignored.
EMF, “ignoreAttributeValueChanges”	Can contain a Sequence of strings of the form “package.class.attribute”. Differences in the values for these attributes will be ignored. However, if the attribute is set on one side and not on the other, the difference will be reported as normal.

Table B.3. Available options by model comparator

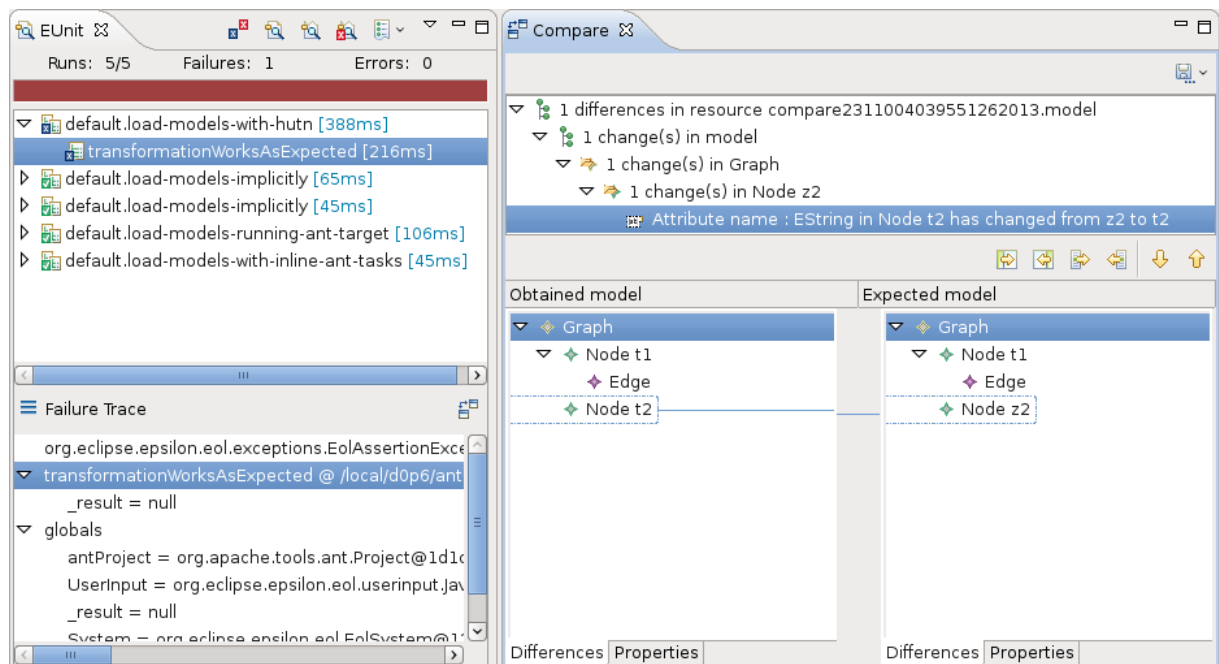


Figure B.3. Screenshot of the EUnit graphical user interface

Listing B.5 Ant buildfile for EUnit with *<modelTasks>* and a helper target

```
<project>
  <target name="run-tests">
    <epsilon.eunit src="test-external.eunit">
      <modelTasks>
        <epsilon.emf.loadModel name="Tree" modelfile="tree.model"
          metamodelfile="tree.ecore" read="true" store="false"/>
        <epsilon.emf.loadModel name="GraphExpected" modelfile="graph.model"
          metamodelfile="graph.ecore" read="true" store="false"/>
        <epsilon.emf.loadModel name="Graph" modelfile="transformed.model"
          metamodelfile="graph.ecore" read="false" store="false"/>
      </modelTasks>
    </epsilon.eunit>
  </target>
  <target name="tree2graph">
    <epsilon.etl src="${basedir}/resources/Tree2Graph.etl">
      <model ref="Tree"/>
      <model ref="Graph"/>
    </epsilon.etl>
  </target>
</project>
```

Listing B.6 EOL script using `runTarget` to run ETL

```
@test
operation transformationWorksAsExpected() {
  runTarget("tree2graph");
  assertEquals("GraphExpected", "Graph");
}
```

model **Tree** into model **Graph** using ETL. The *<modelTasks>* nested element is used to load the input, expected output and output EMF models. **Graph** is loaded with *read* set to **false**: the model will be initially empty, and will be populated by the ETL transformation.

The EOL script is shown in Listing B.6: it invokes the helper task (line 3) and checks that the obtained model is equal to the expected model (line 4). Internally, EMC will perform the comparison using EMF Compare.

B.4.2 Models and tasks in the EOL script

In the previous section, the EOL file is kept very concise because the model setup and model management tasks under test were specified in the Ant buildfile. In this section, the models and the tasks will be inlined into the EOL script instead.

The Ant buildfile is shown in Listing B.7. It is now very simple: all it needs to do is run the EOL script. However, since HUTN will be parsed in the EOL script, the *EPackages* of the metamodels must be previously registered.

The EOL script used is shown in Listing B.8. Instead of loading models through the Ant tasks, the `loadHutn` operation has been used to load the models. The test itself is almost the same, but instead of running a helper target, it invokes an operation which creates and runs the ETL Ant task through the `antProject` variable provided by EUnit, taking advantage of the support in EOL for invoking Java code through reflection.

Listing B.7 Ant buildfile which only runs the EOL script

```
<project>
  <target name="run-tests">
    <epsilon.emf.register file="tree.ecore"/>
    <epsilon.emf.register file="graph.ecore"/>
    <epsilon.eunit src="test-inlined.eunit"/>
  </target>
</project>
```

Listing B.8 EOL script with inlined models and tasks

```
@model
operation loadModels() {
  loadHutn("Tree", '@Spec {Metamodel {nsUri: "Tree" }}
Model {
  Tree "t1" { label : "t1" }
  Tree "t2" {
    label : "t2"
    parent : Tree "t1"
  }
}
');

  loadHutn("GraphExpected", '@Spec {Metamodel {nsUri: "Graph"}}
Graph { nodes :
  Node "t1" {
    name : "t1"
    outgoing : Edge { source : Node "t1" target : Node "t2" }
  },
  Node "t2" {
    name : "t2"
  }
}
');

  loadHutn("Graph", '@Spec {Metamodel {nsUri: "Graph"}}');
}

@test
operation transformationWorksAsExpected() {
  runETL();
  assertEqualsModels("GraphExpected", "Graph");
}

operation runETL() {
  var etlTask := antProject.createTask("epsilon.etl");
  etlTask.setSrc(new Native('java.io.File')(
    antProject.getBaseDir(), 'resources/etl/Tree2Graph.etl'));
  etlTask.createModel().setRef("Tree");
  etlTask.createModel().setRef("Graph");
  etlTask.execute();
}
```

B.5 Extending EUnit

EUnit is based on the Epsilon platform, but it is designed to accommodate other technologies. This section will explain several strategies to add support for these technologies to EUnit.

EUnit uses the Epsilon Model Connectivity abstraction layer to handle different modelling technologies. Adding support for a different modelling technology only requires implementing another driver for EMC. Depending on the modelling technology, the driver can provide optional services such as model comparison, caching or reflection [15]. Currently, Epsilon has built-in support for EMF models, Java object graphs and plain XML files. Third-party drivers for MetaData Repository (MDR) and Z models are also available.

B.5.1 Adding model management tasks

EUnit uses Ant as a workflow language: all model management tasks must be exposed through Ant tasks. It is highly encouraged, however, that the Ant task is aware of the EMC model repository linked to the Ant project. Otherwise, users will have to shuffle the models out from and back into the repository between model management tasks. As an example, a helper target for an ATLAS Transformation Language (ATL) [11] transformation with the existing Ant tasks needs to:

1. Save the input model in the EMC model repository to a file, by invoking the `<epsilon.storeModel>` task.
2. Load the metamodels and the input model with `<atl.loadModel>`.
3. Run the ATL transformation with `<atl.launch>`.
4. Save the result of the ATL transformation with `<atl.saveModel>`.
5. Load it into the EMC model repository with `<epsilon.emf.loadModel>`.

Listing B.9 shows the Ant buildfile which would be required for running these steps, showing that while EUnit can use the existing ATL tasks as-is, the required helper task is quite longer than the one in Listing B.5. Ideally, Ant tasks should be adapted or wrapped to use models directly from the EMC model repository.

Another advantage in making model management tasks EMC-aware is that they can easily “export” their results as models, making them easier to test. For instance, the EVL Ant task allows for exporting its results as a model by setting the attribute `exportAsModel` to `true`. This way, EOL can query the results as any regular model (see Listing B.10). This is simpler than transforming the validated model to a problem metamodel, as suggested in [8]. The example in Listing B.10 checks that a single warning was produced due to the expected rule (`LabelsStartWithT`) and the expected model element.

Listing B.9 Testing an ATL model transformation with EUnit

```
<project>
  <!-- ... omitted ... -->
  <target name="atl">
    <!-- Create temporary files for input and output models -->
    <tempfile property="atl.temp.srcfile" />
    <tempfile property="atl.temp.dstfile" />

    <!-- Save input model to a file -->
    <epsilon.storeModel model="Tree" target="{atl.temp.srcfile}" />

    <!-- Load the metamodels and the source model -->
    <atl.loadModel name="TreeMM" metamodel="MOF" path="metamodels/tree.ecore" />
    <atl.loadModel name="GraphMM" metamodel="MOF" path="metamodels/graph.ecore" />
    <atl.loadModel name="Tree" metamodel="TreeMM" path="{atl.temp.srcfile}" />

    <!-- Run ATL and save the model -->
    <atl.launch path="transformation/tree2graph.atl">
      <inmodel name="IN" model="Tree" />
      <outmodel name="OUT" model="Graph" metamodel="GraphMM" />
    </atl.launch>
    <atl.saveModel model="Graph" path="{atl.temp.dstfile}" />

    <!-- Load it back into the EUnit suite -->
    <epsilon.emf.loadModel name="Graph" modelfile="{atl.temp.dstfile}"
      metamodelfile="metamodels/graph.ecore" read="true" store="false" />

    <!-- Delete temporary files -->
    <delete file="{atl.temp.srcfile}" />
    <delete file="{atl.temp.dstfile}" />
  </target>
</project>
```

Listing B.10 Testing an EVL model validation with EUnit

```
@test
operation valid() {
  var tree := new Tree!Tree;
  tree.label := '1n';
  runTarget('validate-tree');
  var errors := EVL!EvlUnsatisfiedConstraint.allInstances;
  assertEquals(1, errors.size);
  var error := errors.first;
  assertEquals(tree, error.instance);
  assertEquals(false, error.constraint.isCritique);
  assertEquals('LabelsStartWithT', error.constraint.name);
}
```

Listing B.11 Inline model generation in EUnit

```

@data nlevels
operation levels() { return 0.to(4); }

@model
operation generate() {
  // Load an empty model and populate it
  loadHutn("Tree", '@Spec { Metamodel { nsUri: "Tree" }} Model {}');
  generateBinaryTree(new Tree!Node, nlevels);
}

operation generateBinaryTree(root, nlevels) {
  if (nlevels > 0) {
    for (n in Sequence { new Tree!Node, new Tree!Node }) {
      n.parent := root;
      generateBinaryTree(n, nlevels - 1);
    }
  }
}

/* ... tests ... */

```

B.5.2 Integrating model generators

By design, EUnit does not implement any model generation technique, since it is considered that running the tests is orthogonal to generating them. Several model generation tools already exist, such as OMOGEN [6] or Cartier [19]. To EUnit, model generation is just another kind of model management task. There are basically two ways in which models can be generated: *batch* model generation generates all models before repeating every test through them, and *inline* model generation invokes the generator in every test, producing the required models.

Batch model generation can be implemented by calling the Ant task of the model generator before invoking EUnit, and then using a data binding to repeat the tests over every generated model. The Ant tasks required to load these models can be set up by EUnit on the fly in a `@model` operation, using the *antProject* built-in variable. Inline model generation uses data bindings to set the parameters for generating each model, and then invokes the Ant task of the model generation tool in a `@model` operation.

Listing B.11 shows a simple example of inline model generation, using EOL code instead of invoking the Ant task of a model generation tool. Several Tree models are generated by combining data and model bindings. The data variable *nlevels* indicates the number of levels the generated binary tree should have. The `@model` operation loads an empty model and populates it as needed. All tests will be repeated 5 times, with complete binary trees of 0 to 4 levels.

B.6 Case studies

This section will show two practical applications of the EUnit test framework: EUnit is now internally used for several tests of the Epsilon framework, and is also used to test the performance inference algorithms in Chapter 5.

B.6.1 Regression tests for EuGENia

One of the most popular components in Epsilon is its Eugenia [14] tool, which simplifies the creation of graphical model editors based on the Eclipse Graphical Modeling Framework (GMF) [7]. GMF editor models can be very complex, and creating them from scratch can be daunting to a new user. Eugenia can do most of the initial work required to obtain an usable editor, and can automate the customisations that the developer needs.

Testing Eugenia can be difficult, however, as the transformations it uses are implemented in a mix of several languages. The models produced are too complex to verify using manual assertions, and can also change with no warning with a new version of GMF. The ideal approach in this case would be to use regression testing, but before developing EUnit this was an entirely manual process, as it was deemed too difficult to automate.

After developing EUnit, it was internally adopted for adding regression tests for the Eugenia model transformations. A new Ant task for Eugenia was created, and defined the EUnit test suite as follows:

- The Ant buildfile contains a single target which prepares a test environment, runs Eugenia on the test environment and invokes EUnit.
- The EUnit test suite uses a data binding to repeat the tests over each of the six models produced by Eugenia: `.ecore`, `.genmodel`, `.gmfgraph`, `.gmftool`, `.gmfmap` and `.gmfgen`.
- Test setup creates, configures and runs `<epsilon.emf.loadModel>` Ant tasks to load the expected and obtained models.
- Test execution compares the expected and obtained models using the pluggable model comparator that was developed for EMF models, using the Eclipse EMF Compare project.

Using regular Ant tasks to integrate external tools has the added benefit that the same Ant tasks used for testing can also help end-users in automating their own workflows. If a new extension framework for EUnit had been defined, end-users would not be able to take advantage of these improvements.

EUnit has reduced the amount of code required to do the tests, by repeating tests implicitly through data bindings and encapsulating model comparison as a simple assertion. The `antProject` variable supplied by EUnit helped simplify the Ant buildfile as well: instead of specifying everything in it, part of the required Ant tasks are created on the fly inside the EOL script. Together, the tests only require about 180 lines of Java code (Eclipse-specific test setup), 30 lines of Apache Ant configuration (EUnit invocation), and 39 lines of EOL code (the tests themselves).

B.6.2 Unit testing for SODM+T

The SODM+T methodology presented in this work has been tested using a combination of EUnit and JUnit. Before EUnit was developed, some of the tests had to be written using regular Java code, which was very verbose and repetitive, due to the nature of the language and the API offered by the Eclipse Modeling Framework (EMF).

Listing B.12 Several model validation tests for SODM+T using EUnit

```

@setup
operation setUp() {
  -- available directly within the @test
  var sp = new SP!ServiceProcess;
}

@test
operation ExactlyOneProcessStart_noStart() {
  validate();
  assertHasProblem('ExactlyOneProcessStart');
}

@test
operation ExactlyOneProcessStart_oneStart() {
  sp.newInitial();
  validate();
  assertNoProblem('ExactlyOneProcessStart');
}

@test
operation ExactlyOneProcessStart_twoStart() {
  sp.newInitial();
  sp.newInitial();
  validate();
  assertHasProblem('ExactlyOneProcessStart');
}

```

After developing EUnit, most of the Java code doing the actual tests was removed. Instead, some of the EUnit tests had to be supported by Java code in order to integrate them with external tooling, such as Eclipse, the Maven build system and the Jenkins continuous integration server. Nevertheless, the code required is quite trivial in most cases, as will be shown below.

B.6.2.1 Model validation

The model editors provide automatic model validation, backed by a set of Epsilon Validation Language (EVL) scripts. Testing these validation scripts requires building a large number of both valid and invalid models and ensuring that the scripts classified them properly and found the expected errors.

Using Java, this would have required either manually creating over 140 models or using the cumbersome EMF APIs to generate them on the fly. In addition, each of these models would need its own test case.

Instead, by using EUnit, the tests only take up 1390 lines of EOL code and 143 lines of Java code. This includes generating the models, running the test cases and verifying their results. The Java code allows JUnit-aware tools to run the tests and provides an additional EOL operation `validate()` that runs the EVL validation scripts on the current model.

Listing B.12 shows an excerpt of the EUnit `setUp` operation that creates the main element of the model under test and the three first test cases, which are quite simple. The *ServiceProcess* class has been augmented with `newXXX()` EOL context operations for easily creating new nodes inside its instances, and the `assert*Problems()` assertions have been implemented in only 30 lines of EOL code. This shows that the conciseness of EOL and the focus on code reuse in EUnit allow tests to be short and readable.

Listing B.13 Java class used to run EUnit from JUnit-compatible tools

```
@RunWith(EUnitTestRunner.class)
public class FourNodeExhaustiveTest implements IEUnitSuite {
    @Override
    public URI getModuleURI() throws Exception {
        return new File("exhaustive-4nodes.eunit").toURI();
    }

    @Override
    public List<IModel> prepareModels() throws Exception {
        // use the Epsilon Model Connectivity layer to load an empty model
        final EmfModel model = new EmfModel();
        model.setName("Model");
        model.setMetamodelUri(ServiceProcessPackage.eNS_URI);
        model.setModelFile("dummy.model");
        model.setReadOnLoad(false);
        model.setStoredOnDisposal(false);
        model.load();
        return Arrays.asList((IModel)model);
    }

    @Override
    public OperationContributor getOperationContributor() {
        // no additional EOL operations are needed for this test
        return null;
    }
}
```

B.6.2.2 Performance inference (custom annotations)

The performance inference algorithms in Chapter 5 were originally tested using pure Java code. The tests ran the throughput and time limit inference algorithms on a selection of models, checking their results and ensuring that their results were equivalent. These tests required 343 lines of Java code to run. By switching to EUnit, this was dropped to 142 lines of EOL code and 74 lines of Java code (for compatibility with JUnit-based tools).

In addition, inline model creation (as shown in Section B.5.2) was used to implement an exhaustive test suite on a sample of over 100000 two-level fork-join models (see Figure 5.8 on page 5.30) with varying minimum times and weights. The tests checked that the incremental time limit algorithm honored the global time limit was honored on all its paths. This only required adding 90 lines of EOL code and 40 lines of Java code for the JUnit bridge. The JUnit bridge in Listing B.13: essentially, developers only need to add a `@RunWith` annotation to a class that implements the *IEUnitSuite* interface.

B.6.2.3 Performance inference (MARTE annotations)

The tests for the MARTE-based performance inference algorithms ensure that the throughput and time limit inference algorithms produced the expected results on a selection of models. The tests also validate the first stage of the incremental graph-based algorithm, in which some information is aggregated over the model while traversing it in reverse topological order.

Similarly to the algorithms that used custom annotations, the MARTE-based algorithms were originally written using only JUnit. The models produced by the Papyrus editor were considerably more complex to work with than the models using the original custom editor, so the test suites required 468 lines of Java code in total.

After switching to EUnit, this was reduced to 216 lines of EOL code for the tests themselves. 101 lines of Java support code (similar to that in Listing B.13) were also needed.

B.6.2.4 Conclusions

The above sections show that EUnit can be considerably more concise and readable than the equivalent Java code, while providing more functionality. EUnit has been successfully used to test the SODM+T model validators and the two families of *ad hoc* performance inference algorithms (based on different annotations).

On the two migrated test suites, EUnit has required about 50% less lines by itself. However, this reduction is dampened to about 25% after adding some of the Java support code required to reuse EUnit tests as standard Eclipse JUnit plug-in tests. The Java support code is much simpler than the original Java code and can be easily reused over several test cases, but ideally it should not be necessary. It is planned to improve EUnit in future releases to avoid having to write these support classes in most cases.

In addition to being more concise, the inline model creation facilities in EUnit allow for testing some of the performance inference algorithms with over 100000 automatically generated models while using very little code. This proves the usefulness of the model and data binding concepts presented by EUnit.

References

- [1] Apache Foundation. Apache Ant 1.7.1, June 2008. URL <http://ant.apache.org/>. B.3, B.5
- [2] B. Baudry, S. Ghosh, F. Fleurey, R. France, Y. Le Traon, and J. Mottu. Barriers to systematic model transformation testing. *Communications of the ACM*, 53:139–143, June 2010. ISSN 0001-0782. B.1
- [3] K. Beck. JUnit.org, April 2013. URL <http://www.junit.org/>. B.2
- [4] C. Beust. TestNG, March 2013. URL <http://testng.org/>. B.6
- [5] J. Bézivin, F. Jouault, P. Rosenthal, and P. Valduriez. The AMMA platform support for modeling in the large and modeling in the small. Research Report 04.09, LINA, University of Nantes, Nantes, France, February 2005. B.2
- [6] E. Brottier, F. Fleurey, J. Steel, B. Baudry, and Y. Le Traon. Metamodel-based test generation for model transformations: an algorithm and a tool. In *Proc. of the 17th Int. Symposium on Software Reliability Engineering*, pages 85–94, Los Alamitos, California, USA, 2006. IEEE Computer Society. doi: 10.1109/ISSRE.2006.27. B.17
- [7] Eclipse Foundation. Graphical Modeling Project, 2013. URL <http://www.eclipse.org/modeling/gmp/>. Last checked: November 6th, 2013. B.18
- [8] Frédéric Jouault, Jean Bézivin. Using ATL for Checking Models. In *Proc. International Workshop on Graph and Model Transformation (GraMoT)*, Tallinn, Estonia, September 2005. B.15

- [9] M. Guttman and J. Parodi. *Real-Life MDA: Solving Business Problems with Model Driven Architecture*. Morgan Kaufmann, first edition, December 2006. ISBN 0123705924. B.1
- [10] A. Haase, M. Völter, S. Efftinge, and B. Kolb. Introduction to openArchitectureWare 4.1. 2. In *Proceedings of the MDD Tool Implementers Forum, TOOLS Europe 2007*, 2007. B.2
- [11] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: a model transformation tool. *Science of Computer Programming*, 72(1-2):31–39, June 2008. ISSN 0167-6423. doi: 10.1016/j.scico.2007.08.002. B.15
- [12] K. Kawaguchi. Jenkins CI, April 2013. URL <http://jenkins-ci.org/>. B.6
- [13] D. S. Kolovos, R. F. Paige, L. M. Rose, and F. A. Polack. Unit testing model management operations. In *Proceedings of the IEEE International Conference on Software Testing Verification and Validation Workshop, 2008 (ICSTW '08)*, pages 97–104, 2008. B.1
- [14] D. S. Kolovos, L. M. Rose, S. B. Abid, R. F. Paige, F. A. C. Polack, and G. Botterweck. Taming EMF and GMF using model transformation. In D. C. Petriu, N. Rouquette, and O. Haugen, editors, *Model Driven Engineering Languages and Systems*, volume 6394 of *LNCS*, pages 211–225. Springer-Verlag, Berlin, Germany, 2010. ISBN 978-3-642-16144-5. B.18
- [15] D. S. Kolovos, L. M. Rose, R. F. Paige, and A. García-Domínguez. The Epsilon book, 2013. URL <http://dev.eclipse.org/svnroot/modeling/org.eclipse.epsilon/trunk/doc/org.eclipse.epsilon.book/EpsilonBook.pdf>. Last checked: November 6th, 2013. B.6, B.15
- [16] J. Mottu, B. Baudry, and Y. Le Traon. Model transformation testing: oracle issue. In *Proc. of the 2008 IEEE Int. Conf. on Software Testing Verification and Validation*, pages 105–112, Lillehammer, Norway, April 2008. ISBN 978-0-7695-3388-9. doi: 10.1109/ICSTW.2008.27. B.1
- [17] Object Management Group. Human-Usable Textual Notation (HUTN) 1.0, August 2004. URL <http://www.omg.org/technology/documents/formal/hutn.htm>. Last checked: November 6th, 2013. B.5, B.9
- [18] D. Saff. Theory-infected: or how I learned to stop worrying and love universal quantification. In *Companion to the 22nd ACM SIGPLAN Conf. on Object-oriented Programming Systems and Applications, OOPSLA '07*, pages 846–847, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-865-7. doi: 10.1145/1297846.1297919. B.4
- [19] S. Sen, B. Baudry, and J. Mottu. Automatic model generation strategies for model transformation testing. In R. F. Paige, editor, *Theory and Practice of Model Transformations*, volume 5563, pages 148–164. Springer-Verlag, Berlin, Germany, 2009. ISBN 978-3-642-02407-8. B.17

- [20] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, second edition, December 2008. ISBN 978-0321331885. B.2
- [21] R. Straeten, T. Mens, and S. Baelen. Challenges in Model-Driven software engineering. In M. R. V. Chaudron, editor, *Models in Software Engineering*, volume 5421 of *LNCs*, pages 35–47. Springer-Verlag, Berlin, Germany, 2009. ISBN 978-3-642-01647-9. B.1

C

List of acronyms

ACM Application Component Model

AGV Automated Guided Vehicle

AMICE European CIM Architecture

AMMA ATLAS Model Management Architecture

AMPL A Mathematical Programming Language

AMW ATLAS Model Weaver

API Application Programming Interface

ATL ATLAS Transformation Language

B2MML Business to Manufacturing Markup Language

BCM Business Component Model

BDI Belief-Desire-Intention

BDM Business Domain Model

BFU Basic Fractal Unit

BMS Biological Manufacturing System

BPMN Business Process Modelling Notation

BPSOM Business Process Service Oriented Methodology

BSD Berkeley Software Distribution

CAD Computer Aided Design

CAE Computer Aided Engineering

CAM Computer Aided Manufacturing

CASE Computer Assisted Software Engineering

CBD Component-Based Development

CIMOSA Computer-Integrated Manufacturing Open System Architecture

CIM Computer Integrated Manufacturing

CIM Computation Independent Model

CNC Computer Numerical Control

CN Collaborative Network

CORBA Common Object Request Broker Architecture

CSV	Comma-Separated Values
DAG	Directed Acyclic Graph
DNS	Domain Name System
DRM	Detailed Resource Modelling
DTD	Document Type Definition
EE	Extended Enterprise
EGL	Epsilon Generation Language
EMC	Epsilon Model Connectivity
EMF	Eclipse Modeling Framework
EOL	Epsilon Object Language
EPL	Eclipse Public License
ERP	Enterprise Resource Planning
ESPRIT	European Strategic Programme for Research in Information Technology
ETL	Epsilon Transformation Language
EVL	Epsilon Validation Language
EWL	Epsilon Wizard Language
FIPA	Foundation for Intelligent Physical Agents
FrMS	Fractal Manufacturing System
GCM	Generic Component Modelling
GERAM	Generalised Enterprise Reference Architecture and Methodology
GIM	GRAI Integrated Methodology
GLPK	GNU Linear Programming Kit
GMF	Graphical Modelling Framework
GMPL	GNU MathProg Language
GPL	General Public License
GQAM	Generic Quantitative Analysis Modelling
GRAI	Graphes et Résultats et Activités Interreliés
GRM	General Resource Modelling

HLAM High-Level Application Modelling

HMS Holonic Manufacturing System

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

HUTN Human-Usable Textual Notation

ICM Implementation Component Models

IDE Integrated Development Environment

IMS Intelligent Manufacturing Systems

IT Information Technology

JADE Java Agent DEvelopment

JAX-WS Java API for XML Web Services

JET Java Emitter Templates

JSP Java Server Pages

KM3 Kernel Meta Meta Model

LCA Least Common Ancestor

LGPL Lesser General Public License

LP Linear Programming

M2M Model to Model

M2T Model to Text

MARTE Modelling and Analysis of Real-Time and Embedded Systems

MASCOT Multi-Agent Supply Chain cOordination Tool

MAS Multi-Agent System

MDA[®] Model-Driven Architecture[®]

MDR MetaData Repository

MDSE Model Driven Software Engineering

MDT Model Development Tools

MES Manufacturing Execution System

MILP Mixed Integer Linear Programming

MIME	Multipurpose Internet Mail Extensions
MMT	Model to Model Transformation
MOFM2T	MOF Model to Text Transformation
MOF	Meta-Object Facility
MRP	Material Requirements Planning
MRP II	Manufacturing Resource Planning
NC	Numerical Control
NFP	non-functional property
oAW	openArchitectureWare
OCL	Object Constraint Language
OMG	Object Management Group
PAM	Performance Analysis Modelling
PERA	Purdue Enterprise Reference Architecture
PIM	Platform Independent Model
PROSA	Product-Resource-Order-Staff Architecture
PSL	Process Specification Language
PSM	Platform Specific Model
QVT	Query/View/Transformation
QoS/FT	Quality of Service and Fault Tolerance Characteristics and Mechanisms
QoS	Quality of Service
RMI	Remote Method Invocation
RSM	Repetitive Structured Modelling
RT/E	Real Time/Embedded
SAM	Schedulability Analysis Modelling
SF	Software Factories
SGML	Standard Generalised Markup Language
SLA	Service Level Agreement
SLCA	Set of Least Common Ancestors

SME Small and Medium Enterprise

SOAP Simple Object Access Protocol

SOA Service-Oriented Architecture

SODM+T SODM with Testing

SODM Service Oriented Development Method

SOMA Service Oriented Modeling and Architecture

SPT Schedulability, Performability and Time

SVG Scalable Vector Graphics

TAFIM Technical Architecture Framework for Information Management

TOGAF The Open Group Architecture Framework

UDDI Universal Description, Discovery and Integration

UML Unified Modelling Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

VE Virtual Enterprise

VSL Value Specification Language

VSM Value Stream Mapping

W3C World Wide Web Consortium

WADE Workflows and Agents Development Environment

WFMS Workflow Management System

WS-BPEL Web Services Business Process Execution Language

WS-I BP Web Services Interoperability Basic Profile

WS-I Web Services Interoperability Organisation

WSDL Web Services Description Language

WS Web Services

XML eXtensible Markup Language

Yams Yet Another Manufacturing System

D

Bibliography

-
- [22] S. Adhau, M. Mittal, and A. Mittal. A multi-agent system for distributed multi-project scheduling: An auction-based negotiation approach. *Engineering Applications of Artificial Intelligence*, December 2011. ISSN 0952-1976. doi: 10.1016/j.engappai.2011.12.003.
- [23] F. Aguayo González. *Diseño y Fabricación de Productos en Sistemas Holónicos: Aplicación al Desarrollo de un Módulo Holónico de Diseño*. PhD thesis, University of Cádiz, 2003.
- [24] F. Aguayo González, J. Lama Ruiz, M. Sánchez Carrilero, R. Bienvenido Bárcena, J. González Madrigal, and M. Marcos Bárcena. Concepción holónica de la ergonomía en sistemas de fabricación automatizados. *Anales de Ingeniería Mecánica*, pages 1087–1095, 2004.
- [25] F. Aguayo González, M. Marcos Bárcena, M. Sánchez Carrilero, and J. Lama Ruiz. *Sistemas Avanzados de Fabricación Distribuida*. Ra-Ma, Madrid, España, 2007. ISBN 9788478978045.
- [26] R. S. Aguilar-Savén. Business process modelling: Review and framework. *International Journal of Production Economics*, 90(2):129–149, July 2004. ISSN 0925-5273. doi: 10.1016/S0925-5273(03)00102-6.
- [27] M. Alhaj and D. C. Petriu. Approach for generating performance models from UML models of SOA systems. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research, CASCON '10*, pages 268–282, New York, USA, 2010. ACM. doi: 10.1145/1923947.1923975.
- [28] Apache Foundation. Apache Ant 1.7.1, June 2008. URL <http://ant.apache.org/>.
- [29] Apache Software Foundation. Apache Velocity Project homepage, November 2010. URL <http://velocity.apache.org>. Last checked: November 6th, 2013.
- [30] Apache Software Foundation. Apache JMeter, November 2013. URL <http://jakarta.apache.org/jmeter/>. Last checked: November 6th, 2013.
- [31] Apache Software Foundation. Apache Maven homepage, January 2013. URL <http://maven.apache.org>. Last checked: November 6th, 2013.
- [32] Apache Software Foundation. Apache CXF, November 2013. URL <https://cxf.apache.org/>. Last checked: November 6th, 2013.
- [33] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering*, 33(6):369–384, 2007. doi: 10.1109/TSE.2007.1011.
- [34] P. Aston and C. Fitzgerald. The Grinder, a Java Load Testing Framework, 2012. URL <http://grinder.sourceforge.net/>. Last checked: November 6th, 2013.
- [35] C. Atkinson, P. Bostan, D. Brenner, G. Falcone, M. Gutheil, O. Hummel, M. Juhasz, and D. Stoll. Modeling components and Component-Based systems in Kobra. In *The Common Component Modeling Example*, volume 5153 of *Lecture Notes in*

- Computer Science*, pages 54–84. Springer Berlin, Heidelberg, Alemania, 2008. ISBN 978-3-540-85288-9. doi: 10.1007/978-3-540-85289-6_4.
- [36] A. Avritzer and E. J. Weyuker. Deriving workloads for performance testing. *Software: Practice and Experience*, 26(6):613–633, 1996. ISSN 1097-024X.
 - [37] R. Babiceanu and F. Chen. Development and applications of holonic manufacturing systems: A survey. *Journal of Intelligent Manufacturing*, 17(1):111–131, 2006. ISSN 0956-5515. doi: 10.1007/s10845-005-5516-y.
 - [38] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: a survey. *IEEE Transactions on Software Engineering*, May 2004. doi: 10.1109/TSE.2004.9.
 - [39] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. A timed extension of WSCoL. In *Proceedings of the IEEE International Conference on Web Services, 2007 (ICWS 2007)*, pages 663–670, 2007. doi: 10.1109/ICWS.2007.25.
 - [40] M. Barnett, K. R. M. Leino, and W. Schulte. The Spec# programming system: an overview. In *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices*, pages 49–69. Springer Berlin, 2005.
 - [41] B. Baudry, S. Ghosh, F. Fleurey, R. France, Y. Le Traon, and J. Mottu. Barriers to systematic model transformation testing. *Communications of the ACM*, 53:139–143, June 2010. ISSN 0001-0782.
 - [42] T. Bear. Grinder Analyzer homepage, July 2012. URL <http://track.sourceforge.net/>. Last checked: November 6th, 2013.
 - [43] K. Beck. JUnit.org, April 2013. URL <http://www.junit.org/>.
 - [44] M. A. Bender, G. Pemmasani, S. Skiena, and P. Sumazin. Finding least common ancestors in directed acyclic graphs. *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA’01)*, pages 845–853, 2001. doi: 10.1.1.15.9161.
 - [45] V. Bergmann. ContiPerf 2, September 2011. URL <http://databene.org/contiperf.html>. Last checked: November 6th, 2013.
 - [46] S. Bernardi, J. Merseguer, and D. C. Petriu. A dependability profile within MARTE. *Software & Systems Modeling*, 2009. ISSN 1619-1366. doi: 10.1007/s10270-009-0128-1.
 - [47] S. Bernardi, J. Campos, and J. Merseguer. Timing-Failure risk assessment of UML design using time petri net bound techniques. *IEEE Transactions on Industrial Informatics*, 2010. ISSN 1551-3203. doi: 10.1109/TII.2010.2098415.
 - [48] C. Beust. TestNG, March 2013. URL <http://testng.org/>.
 - [49] J. Bézivin, F. Jouault, P. Rosenthal, and P. Valduriez. The AMMA platform support for modeling in the large and modeling in the small. Research Report 04.09, LINA, University of Nantes, Nantes, France, February 2005.

-
- [50] R. Bienvenido Bárcena, M. Álvarez Alcón, J. González Madrigal, M. Marcos Bárcena, and M. Sánchez Carrilero. Holonic manufacturing systems: an emergent proposal for the 21st century. *The International Journal for Manufacturing Science and Production*, 1999.
- [51] C. Bock. Interprocess communication in the process specification language. Technical Report NISTIR 7348, National Institute of Standards and Technology, Gaithersburg, MD, USA, October 2006.
- [52] C. Bock and M. Gruninger. PSL: a semantic domain for flow models. *Software & Systems Modeling*, 4(2):209–231, 2005. ISSN 1619-1366. doi: 10.1007/s10270-004-0066-x.
- [53] E. Brottier, F. Fleurey, J. Steel, B. Baudry, and Y. Le Traon. Metamodel-based test generation for model transformations: an algorithm and a tool. In *Proc. of the 17th Int. Symposium on Software Reliability Engineering*, pages 85–94, Los Alamitos, California, USA, 2006. IEEE Computer Society. doi: 10.1109/ISSRE.2006.27.
- [54] J. Browne, I. Hunt, and J. Zhang. The Extended Enterprise (EE). In L. M. Camarinha-Matos, H. Afsarmanes, and V. Merik, editors, *Intelligent Systems for Manufacturing: Multi-Agent Systems and Virtual Organizations*, pages 3–30. Kluwer Academic Publishers, Londres, 1998.
- [55] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot. MoDisco: a generic and extensible framework for model driven reverse engineering. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, pages 173–174, Antwerp, Belgium, September 2010.
- [56] H. V. Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters. Reference architecture for holonic manufacturing systems: PROSA. *Computers in Industry*, 37(3):255–274, November 1998. ISSN 0166-3615.
- [57] S. Brückner, J. Wyns, P. Peeters, and M. Kollingbaum. Designing agents for manufacturing control. In *Proceedings of the 2nd AI & Manufacturing Research Planning Workshop*, pages 40–46, 1998.
- [58] L. Burdy, Y. Cheon, and D. R. Cok. An overview of JML tools and applications. *International Journal on Software Tools for Technology Transfer (STTT)*, 7(3): 212–232, June 2005.
- [59] S. Bussmann. An agent-oriented architecture for holonic manufacturing control. In *Proceedings of the First Open Workshop IMS Europe*, Lausanne, Switzerland, 1998. URL <http://stefan-bussmann.de/downloads/ims98.pdf>.
- [60] S. Bussmann and K. Schild. An agent-based approach to the control of flexible production systems. In *2001 8th IEEE International Conference on Emerging Technologies and Factory Automation, 2001. Proceedings*, volume 2, pages 481–488 vol.2, October 2001. doi: 10.1109/ETFA.2001.997722.
- [61] J. Bézivin. On the unification power of models. *Software and Systems Modeling*, 4(2):171–188, May 2005. ISSN 1619-1366. doi: 10.1007/s10270-005-0079-0.

- [62] G. Caire, D. Gotta, and M. Banzi. WADE: a software platform to develop mission critical applications exploiting agents and workflows. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, pages 29–36, 2008.
- [63] L. M. Camarinha-Matos and H. Afsarmanesh. Elements of a base VE infrastructure. *Computers in Industry*, 51(2):139–163, June 2003. ISSN 01663615. doi: 10.1016/S0166-3615(03)00033-2.
- [64] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web*, April 2004. doi: 10.1016/j.websem.2004.03.001.
- [65] D. Chen, B. Vallespir, and G. Doumeingts. GRAI integrated methodology and its mapping onto generic enterprise reference architecture and methodology. *Computers in Industry*, 33(2–3):387–394, September 1997. ISSN 0166-3615. doi: 10.1016/S0166-3615(97)00043-2.
- [66] P. P.-s. Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1:9–36, 1976.
- [67] C.-M. Chituc, C. Toscano, and A. Azevedo. Interoperability in collaborative networks: Independent and industry-specific initiatives – the case of the footwear industry. *Computers in Industry*, 59(7):741–757, September 2008. ISSN 0166-3615.
- [68] J. H. Christensen. Holonic manufacturing systems: Initial architecture and standards directions. In *Proceedings of the First European Conference on Holonic Manufacturing Systems*, Hannover, Germany, December 1994.
- [69] G. Confessore, S. Giordani, and S. Rismondo. A market-based multi-agent system model for decentralized multi-project scheduling. *Annals of Operations Research*, 150(1):115–135, 2007. ISSN 0254-5330. doi: 10.1007/s10479-006-0158-9.
- [70] P. I. Cowling, D. Ouelhadj, and S. Petrovic. Dynamic scheduling of steel casting and milling using multi-agents. *Production Planning & Control*, 15(2):178–188, 2004.
- [71] M. V. de Castro. *Aproximación MDA para el desarrollo orientado a servicios de sistemas de información web: del modelo de negocio al modelo de composición de servicios web*. PhD thesis, Universidad Rey Juan Carlos, March 2007.
- [72] V. De Castro, E. Marcos, and J. M. Vara. Applying CIM-to-PIM model transformations for the service-oriented development of information systems. *Information and Software Technology*, 53(1):87–105, 2011. ISSN 0950-5849. doi: 10.1016/j.infsof.2010.09.002.
- [73] S. M. Deen. HMS/FB architecture and its implementation. In *Agent Based Manufacturing: Advances in the Holonic Approach*. Springer, July 2003. ISBN 9783540440697.
- [74] M. D. Del Fabro, J. Bézin, and P. Valduriez. Weaving models with the eclipse AMW plugin. In *Proceedings of the 2006 Eclipse Modeling Symposium, Eclipse Summit Europe*, Esslingen, Germany, October 2006.

-
- [75] A. Delgado, F. Ruiz, I. de Guzmán, and M. Piattini. Business process service oriented methodology (BPSOM) with service generation in SoaML. In H. Mouratidis and C. Rolland, editors, *Advanced Information Systems Engineering*, volume 6741 of *Lecture Notes in Computer Science*, pages 672–680. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-21639-8.
- [76] D. M. Dilts, N. P. Boyd, and H. H. Whorms. The evolution of control architectures for automated manufacturing systems. *Journal of Manufacturing Systems*, 10(1): 79–93, 1991. ISSN 0278-6125.
- [77] G. Doumeingts, Y. Ducq, B. Vallespir, and S. Kleinhans. Production management and enterprise modelling. *Computers in Industry*, 42(2–3):245–263, June 2000. ISSN 0166-3615. doi: 10.1016/S0166-3615(99)00074-3.
- [78] D. F. D’Souza and A. C. Wills. *Objects, Components, and Frameworks with UML: The Catalysis(SM) Approach*. Addison-Wesley Professional, October 1998. ISBN 0201310120.
- [79] Eclipse Foundation. Eclipse wiki – Xcore, 2012. URL <http://wiki.eclipse.org/Xcore>.
- [80] Eclipse Foundation. Eclipse Modeling Framework, 2013. URL <http://eclipse.org/modeling/emf/>. Last checked: November 6th, 2013.
- [81] Eclipse Foundation. Graphical Modeling Project, 2013. URL <http://www.eclipse.org/modeling/gmp/>. Last checked: November 6th, 2013.
- [82] Eclipse Foundation. Main page of the Model to Text project (M2T), 2013. URL <http://www.eclipse.org/modeling/m2t/>. Last checked: November 6th, 2013.
- [83] Eclipse Foundation. Model Development Tools (MDT) project homepage, 2013. URL <http://www.eclipse.org/modeling/mdt/?project=ocl>. Last checked: November 6th, 2013.
- [84] Eclipse Foundation. Model to Model Transformation (MMT) project homepage, 2013. URL <http://www.eclipse.org/mmt/>. Last checked: November 6th, 2013.
- [85] Eclipse Foundation. Emfatic project homepage, 2013. URL <http://www.eclipse.org/modeling/emft/emfatic/>. Last checked: November 6th, 2013.
- [86] Eclipse Foundation. Graphiti project homepage, 2013. URL <http://www.eclipse.org/graphiti/>. Last checked: November 6th, 2013.
- [87] Eclipse Foundation. Homepage of the mdt uml2 project, June 2013. URL <http://www.eclipse.org/modeling/mdt/?project=uml2>. Last checked: November 6th, 2013.
- [88] Eclipse Foundation. Homepage of the papyrus project, June 2013. URL <http://www.eclipse.org/papyrus/>. Last checked: November 6th, 2013.

- [89] G. Engels, A. Hess, B. Humm, O. Juwig, M. Lohmann, J. Richter, M. Voß, and J. Willkomm. A method for engineering a true Service-Oriented Architecture. In J. Cordeiro and J. Filipe, editors, *Proceedings of the 10th International Conference on Enterprise Information Systems*, pages 272–281, Barcelona, España, 2008. ISBN 978-989-8111-38-8.
- [90] T. Erl. *SOA: Principles of Service Design*. Prentice Hall, Indiana, EEUU, 2008. ISBN 0132344823.
- [91] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao. The daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1-3):35–45, 2007.
- [92] eviware.com. soapUI home page, 2012. URL <http://www.soapui.org/>.
- [93] eviware.com. loadUI homepage, 2013. URL <http://www.loadui.org/>. Last checked: November 6th, 2013.
- [94] M. Fletcher, E. Garcia-Herreros, J. Christensen, S. Deen, and R. Mittmann. An open architecture for holonic cooperation and autonomy. In *11th International Workshop on Database and Expert Systems Applications, 2000. Proceedings*, pages 224–230, 2000. doi: 10.1109/DEXA.2000.875031.
- [95] Foundation for Intelligent Physical Agents. FIPA abstract architecture specification SC00001L, December 2002. URL <http://www.fipa.org/specs/fipa00001/SC00001L.pdf>. Last checked: November 6th, 2013.
- [96] Foundation for Intelligent Physical Agents. FIPA standard status specifications, 2002. URL <http://www.fipa.org/repository/standardspecs.html>. Last checked: November 6th, 2013.
- [97] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: a modeling language for mathematical programming*. Thomson/Brooks/Cole, California, EEUU, 2003. ISBN 9780534388096.
- [98] M. S. Fox, J. F. Chionglo, and M. Barbuceanu. The integrated supply chain management system. Technical report, University of Toronto, Department of Industrial Engineering, 1993.
- [99] Frédéric Jouault, Jean Bezivin. Using ATL for Checking Models. In *Proc. International Workshop on Graph and Model Transformation (GraMoT)*, Tallinn, Estonia, September 2005.
- [100] G. Doumeingts, B. Vallespir, M. Zannittin, and D. Chen. GIM-GRAI integrated methodology, a methodology for designing CIM systems, version 1.0. Technical report, University Bordeaux, Bordeaux, France, May 1992.
- [101] A. García-Domínguez. Homepage of the SODM+T project, April 2013. URL <https://neptuno.uca.es/redmine/projects/sodmt>.

-
- [102] S. Ghosh, A. Arsanjani, and A. Allam. SOMA: a method for developing service-oriented solutions. *IBM Systems Journal*, 47(3):377–396, 2008.
- [103] A. Giret Boggino. *ANEMONA: una metodología multiagente para sistemas holónicos de fabricación*. PhD thesis, Universidad Politécnica de Valencia, July 2005.
- [104] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991. doi: 10.1145/103162.103163.
- [105] D. Goldberg, V. Cicirello, M. B. Dias, R. Simmons, S. Smith, and A. Stentz. Task allocation using a distributed market-based planning mechanism. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 996–997, 2003.
- [106] C. F. Goldfarb. The roots of SGML – a personal recollection, 1996. URL <http://www.sgmlsource.com/history/roots.htm>. Last checked: November 6th, 2013.
- [107] J. González Madrigal, J. Sánchez Sola, M. Marcos Bárcena, and M. Sánchez Carrilero. Aproximaciones a los sistemas de fabricación holónicos. *Informacion de Máquinas-Herramientas y Equipos*, pages 59–65, 1998.
- [108] GoPivotal. Spring Framework homepage, November 2013. URL <http://projects.spring.io/spring-framework/>. Last checked: November 6th, 2013.
- [109] GoPivotal. Spring Roo homepage, August 2013. URL <http://projects.spring.io/spring-roo/>. Last checked: November 6th, 2013.
- [110] J. Gordijn and H. Akkermans. Value-based requirements engineering: exploring innovative e-commerce ideas. *Requirements Engineering*, 8(2):114–134, July 2003. doi: 10.1007/s00766-003-0169-x.
- [111] J. Gordijn and H. Akkermans. e3value™ toolset, August 2006. URL <http://www.e3value.com/tools/>.
- [112] R. Gorrieri, H. Wehrheim, F. Jouault, and J. Bézivin. KM3: a DSL for metamodel specification. In *Formal Methods for Open Object-Based Distributed Systems*, volume 4037 of *Lecture Notes in Computer Science*, pages 171–185. Springer Berlin Heidelberg, 2006.
- [113] J. Greenfield. Software factories: Assembling applications with patterns, models, frameworks, and tools, November 2004. URL <http://msdn.microsoft.com/en-us/library/ms954811.aspx>. Last checked: November 6th, 2013.
- [114] J. Greenfield, K. Short, S. Cook, S. Kent, and J. Crupi. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, first edition, August 2004. ISBN 9780471202844.
- [115] M. Guttman and J. Parodi. *Real-Life MDA: Solving Business Problems with Model Driven Architecture*. Morgan Kaufmann, first edition, December 2006. ISBN 0123705924.

- [116] A. Haase, M. Völter, S. Efftinge, and B. Kolb. Introduction to openArchitectureWare 4.1. 2. In *Proceedings of the MDD Tool Implementers Forum, TOOLS Europe 2007*, 2007.
- [117] D. Harel and B. Rumpe. Meaningful modeling: what’s the semantics of “semantics”? *Computer*, 37(10):64–72, 2004. ISSN 0018-9162. doi: 10.1109/MC.2004.172.
- [118] P. Hoyer, M. Gebhart, I. Pansa, A. Dikanski, and S. Abeck. Service-oriented integration using a model-driven approach. *International Journal On Advances in Software*, 3(1):304–317, September 2010. ISSN 1942-2628.
- [119] G. Iacono and F. Muñoz-Castillo. grinder-maven-plugin homepage, June 2013. URL <http://code.google.com/p/grinder-maven-plugin/>.
- [120] IBM Corporation, MESA International, and Capgemini. SOA in Manufacturing Guidebook, May 2008.
- [121] IFAC/IFIP Task Force. GERAM: generalised enterprise reference architecture and methodology, March 1999. URL <http://www.ict.griffith.edu.au/~bernus/taskforce/geram/versions/geram1-6-3/v1.6.3.html>.
- [122] International Electrotechnical Commission. IEC/FDIS 62264-1:2003 – enterprise-control system integration – part 1: Models and terminology, 2003.
- [123] International Electrotechnical Commission. IEC/FDIS 62264-2:2004 – enterprise-control system integration – part 2: Model object attributes, 2004.
- [124] International Electrotechnical Commission. Function blocks - part 1: Architecture. Technical Report IEC 61499-1, IEC, 2005.
- [125] International Electrotechnical Commission. IEC/DIS 62264-3:2007 – enterprise-control system integration – part 3: Activity models of manufacturing operations management, 2007.
- [126] International Standards Organization. ISO 8879:1986 – information processing – text and office systems – standard generalized markup language (SGML), 1986.
- [127] International Standards Organization. ISO 15704 – industrial automation systems – requirements for enterprise-reference architectures and methodologies, August 1999.
- [128] International Standards Organization. ISO 18629-1 – process specification language – part 1: Overview and basic principles, 2004.
- [129] International Standards Organization. ISO 19439 – enterprise integration – framework for enterprise modelling, 2006.
- [130] International Standards Organization. ISO 19440 – enterprise integration – constructs for enterprise modelling, 2007.
- [131] P. R. James and P. Chalin. Faster and more complete extended static checking for the java modeling language. *Journal of Automated Reasoning*, 44(1-2):145–174, February 2010. ISSN 0168-7433. doi: 10.1007/s10817-009-9134-9.

-
- [132] Java.net. JAX-WS reference implementation, November 2013. URL <http://jax-ws.java.net/>. Last checked: November 6th, 2013.
- [133] JBoss Community. Hibernate homepage, November 2013. URL <http://hibernate.org/>. Last checked: November 6th, 2013.
- [134] K. Johansen, M. Comstock, and M. Winroth. Coordination in collaborative manufacturing mega-networks: a case study. *Journal of Engineering and Technology Management*, 22(3):226–244, September 2005. ISSN 0923-4748.
- [135] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: a model transformation tool. *Science of Computer Programming*, 72(1-2):31–39, June 2008. ISSN 0167-6423. doi: 10.1016/j.scico.2007.08.002.
- [136] K. Kawaguchi. Jenkins CI, April 2013. URL <http://jenkins-ci.org/>.
- [137] A. Knutilla, C. Schlenoff, S. Ray, S. T. Polyak, A. Tate, S. C. Cheah, and R. C. Anderson. Process specification language: An analysis of existing representations. Technical Report NISTIR 6133, National Institute of Standards and Technology, Gaithersburg, MD, USA, 1998.
- [138] A. Koestler. Some general properties of self-regulating open hierarchic order (SOHO). In A. Koestler and J. R. Smythies, editors, *Beyond Reductionism: New Perspectives In The Life Sciences*. Houghton Mifflin Co, 1971. ISBN 0807015350.
- [139] A. Koestler. *The Ghost in the Machine*. Penguin Books, June 1990. ISBN 978-0140191929.
- [140] D. S. Kolovos. Epsilon ModelLink, 2012. URL <http://eclipse.org/gmt/epsilon/doc/modelink/>. Last checked: November 6th, 2013.
- [141] D. S. Kolovos, R. F. Paige, L. M. Rose, and F. A. Polack. Unit testing model management operations. In *Proceedings of the IEEE International Conference on Software Testing Verification and Validation Workshop, 2008 (ICSTW '08)*, pages 97–104, 2008.
- [142] D. S. Kolovos, L. M. Rose, S. B. Abid, R. F. Paige, F. A. C. Polack, and G. Botterweck. Taming EMF and GMF using model transformation. In D. C. Petriu, N. Rouquette, and O. Haugen, editors, *Model Driven Engineering Languages and Systems*, volume 6394 of *LNCS*, pages 211–225. Springer-Verlag, Berlin, Germany, 2010. ISBN 978-3-642-16144-5.
- [143] D. S. Kolovos, L. M. Rose, R. F. Paige, and A. García-Domínguez. The Epsilon book, 2013. URL <http://dev.eclipse.org/svnroot/modeling/org.eclipse.epsilon/trunk/doc/org.eclipse.epsilon.book/EpsilonBook.pdf>. Last checked: November 6th, 2013.
- [144] K. Kosanke. CIMOSA – overview and status. *Computers in Industry*, 27(2):101–109, October 1995. ISSN 0166-3615. doi: 10.1016/0166-3615(95)00016-9.

- [145] K. Kosanke and M. Zelm. CIMOSA modelling processes. *Computers in Industry*, 40(2–3):141–153, November 1999. ISSN 0166-3615. doi: 10.1016/S0166-3615(99)00020-2.
- [146] K. Kosanke, F. Vernadat, and M. Zelm. CIMOSA: enterprise engineering and integration. *Computers in Industry*, 40(2–3):83–97, November 1999. ISSN 0166-3615. doi: 10.1016/S0166-3615(99)00016-0.
- [147] B.-R. Lea, M. C. Gupta, and W.-B. Yu. A prototype multi-agent ERP system: an integrated architecture and a conceptual framework. *Technovation*, 25(4):433–441, 2005. ISSN 0166-4972. doi: 10.1016/S0166-4972(03)00153-6.
- [148] Y.-H. Lee, S. R. T. Kumara, and K. Chatterjee. Multiagent based dynamic resource scheduling for distributed multiple projects using a market mechanism. *Journal of Intelligent Manufacturing*, 14(5):471–484, 2003. ISSN 0956-5515. doi: 10.1023/A:1025753309346.
- [149] P. Leitão. Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence*, 22(7):979–991, October 2009. ISSN 0952-1976.
- [150] M. Lohmann, S. Sauer, and G. Engels. Executable visual contracts. In *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 63–70, 2005. doi: 10.1109/VLHCC.2005.35.
- [151] M. Lohmann, G. Engels, and S. Sauer. Model-driven monitoring: generating assertions from visual contracts. In *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering, 2006 (ASE '06)*, pages 355–356, 2006. ISBN 1527-1366. doi: 10.1109/ASE.2006.52.
- [152] M. Lohmann, L. Mariani, and R. Heckel. A model-driven approach to discovery, testing and monitoring of web services. In *Test and Analysis of Web Services*, pages 173–204. Springer Berlin, 2007. ISBN 978-3-540-72911-2. doi: 10.1007/978-3-540-72912-9_7.
- [153] J. P. López-Grao, J. Merseguer, and J. Campos. From UML activity diagrams to Stochastic Petri nets: application to software performance engineering. *SIGSOFT Softw. Eng. Notes*, 2004. doi: 10.1145/974043.974048.
- [154] H. Lucas. Performance evaluation and monitoring. *ACM Computing Surveys*, September 1971. doi: 10.1145/356589.356590.
- [155] F. Macia-Perez, J. V. Berna-Martinez, D. Marcos-Jonquera, I. Lorenzo-Fonseca, and A. Ferrandiz-Colmeiro. A new paradigm: cloud agile manufacturing. *International Journal of Advanced Science and Technology*, 45:47–54, August 2012. ISSN 2005-4238.
- [156] C. M. MacKenzie, K. Laskey, F. McCabe, P. Brown, and R. Metz. Reference Model for Service Oriented Architecture 1.0, October 2006. URL <http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf>. Last checked: November 6th, 2013.

-
- [157] Manufuture High Level Group. Manufuture: a vision for 2020. Technical report, European Commission, Brussels, Belgium, November 2004. ISBN 92-894-8322-9.
- [158] M. Marcos, F. Aguayo, M. Sánchez Carrilero, L. Sevilla, and J. R. Lama. Toward the next generation of manufacturing systems. Frabiho: a synthesis model for distributed manufacturing. In *Proceedings of the First I*proms Virtual Conference*, pages 35–40. Elsevier, 2005.
- [159] M. Marcos Bárcena, M. Álvarez Alcón, M. Sánchez Carrilero, and J. Sánchez Sola. Sistemas de fabricación holónicos: una propuesta para el siglo XXI. *Anales de Ingeniería Mecánica*, 12(3):275–281, 1998.
- [160] V. Marik, M. Fletcher, and M. Pechoucek. Holons & agents: Recent developments and mutual impacts. In V. Marik, O. Stepankova, H. Krautwurmova, and M. Luck, editors, *Multi-Agent Systems and Applications II*, volume 2322 of *Lecture Notes in Computer Science*, pages 89–106. Springer Berlin / Heidelberg, 2002. ISBN 978-3-540-43377-4.
- [161] Martin Fowler. UmlAsSketch, August 2012. URL <http://martinfowler.com/bliki/UmlAsSketch.html>. Last checked: November 6th, 2013.
- [162] R. J. Mayer, C. P. Menzel, M. K. Painter, P. S. de Witte, T. Blinn, and B. Perakath. IDEF3 process description capture method report. Interim Technical Report AL-TR-1995-XXXX, Knowledge Based Systems Inc., Texas, USA, September 1995.
- [163] L. Monostori, J. Váncza, and S. Kumara. Agent-based systems for manufacturing. *CIRP Annals - Manufacturing Technology*, 55(2):697–720, 2006. ISSN 0007-8506. doi: 10.1016/j.cirp.2006.10.004.
- [164] J. Mottu, B. Baudry, and Y. Le Traon. Model transformation testing: oracle issue. In *Proc. of the 2008 IEEE Int. Conf. on Software Testing Verification and Validation*, pages 105–112, Lillehammer, Norway, April 2008. ISBN 978-0-7695-3388-9. doi: 10.1109/ICSTW.2008.27.
- [165] G. J. Myers. *The Art of Software Testing*. John Wiley & Sons, 2 edition, 2004. ISBN 0471469122.
- [166] L. Mönch, M. Stehli, and J. Zimmermann. FABMAS: an agent-based system for production control of semiconductor manufacturing processes. In V. Marík, D. McFarlane, and P. Valckenaers, editors, *Holonic and Multi-Agent Systems for Manufacturing*, volume 2744 of *Lecture Notes in Computer Science*, pages 258–267. Springer Berlin / Heidelberg, 2003. ISBN 978-3-540-40751-5.
- [167] F. Nachira. Towards a network of digital business ecosystems fostering the local development. Discussion paper, European Commission, Brussels, Belgium, September 2002. URL <http://www.digital-ecosystems.org/doc/discussionpaper.pdf>. Last checked: November 6th, 2013.
- [168] Y.-E. Nahm and H. Ishikawa. A hybrid multi-agent system architecture for enterprise integration using computer networks. *Robotics and Computer-Integrated Manufacturing*, 21(3):217–234, June 2005. ISSN 0736-5845. doi: 10.1016/j.rcim.2004.07.016.

- [169] D. K. Nguyen, W.-J. van den Heuvel, M. Papazoglou, V. de Castro, and E. Marcos. GAMBUSE: a gap analysis methodology for engineering SOA-Based applications. In *Conceptual Modeling: Foundations and Applications*, volume 5600 of *Lecture Notes in Computer Science*, pages 293–318. Springer Berlin Heidelberg, 2009.
- [170] C. Nikolai and G. Madey. Tools of the trade: A survey of various agent based modeling platforms. *Journal of Artificial Societies and Social Simulation*, 12(2):2, 2009.
- [171] OASIS. Web Service Business Process Execution Language (WS-BPEL) 2.0, April 2007. URL <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>. Last checked: November 6th, 2013.
- [172] Object Management Group. MDA Guide version 1.0.1, June 2003. URL <http://www.omg.org/cgi-bin/doc?omg/03-06-01>. Last checked: November 6th, 2013.
- [173] Object Management Group. Human-Usable Textual Notation (HUTN) 1.0, August 2004. URL <http://www.omg.org/technology/documents/formal/hutn.htm>. Last checked: November 6th, 2013.
- [174] Object Management Group. UML Profile for Schedulability, Performance, and Time (SPTP) 1.1, January 2005. URL <http://www.omg.org/spec/SPTP/1.1/>. Last checked: November 6th, 2013.
- [175] Object Management Group. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QFTP) 1.1, April 2008. URL <http://www.omg.org/spec/QFTP/1.1/>. Last checked: November 6th, 2013.
- [176] Object Management Group. Business process model and notation 2.0, January 2011. URL <http://www.omg.org/spec/BPMN/2.0/>. Last checked: November 6th, 2013.
- [177] Object Management Group. UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) 1.1, June 2011. URL <http://www.omg.org/spec/MARTE/1.1/>. Last checked: November 6th, 2013.
- [178] Object Management Group. Meta-Object Facility (MOF) 2.4.1, August 2011. URL <http://www.omg.org/spec/MOF/2.4.1/>. Last checked: November 6th, 2013.
- [179] Object Management Group. Query/View/Transformation (QVT) 1.1, January 2011. URL <http://www.omg.org/spec/QVT/1.1/>. Last checked: November 6th, 2013.
- [180] Object Management Group. Unified Modeling Language (UML) 2.4.1, August 2011. URL <http://www.omg.org/spec/UML/2.4.1/>. Last checked: November 6th, 2013.
- [181] Object Management Group. Object Constraint Language Specification (OCL) 2.3.1, January 2012. URL <http://www.omg.org/spec/OCL/2.3.1/>. Last checked: November 6th, 2013.
- [182] Object Management Group. Service oriented architecture modeling language (SoaML) 1.0.1, May 2012. URL <http://www.omg.org/spec/SoaML/1.0.1/>.

-
- [183] Organization for the Advancement of Structured Information Standards. Universal Description Discovery and Integration Standard 3.0, October 2004. URL http://uddi.org/pubs/uddi_v3.htm. Last checked: November 6th, 2013.
- [184] Organization for the Advancement of Structured Information Standards. Web service implementation methodology, July 2005. URL https://www.oasis-open.org/committees/documents.php?wg_abbrev=fwsi. Last checked: November 6th, 2013.
- [185] D. Ouelhadj and S. Petrovic. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, 12(4):417–431, 2009. ISSN 1094-6136. doi: 10.1007/s10951-008-0090-8.
- [186] M. P. Papazoglou and W.-J. V. D. Heuvel. Service-oriented design and development methodology. *Int. J. Web Eng. Technol.*, 2(4):412–442, 2006.
- [187] H. V. D. Parunak. Manufacturing experience with the contract net. *Distributed Artificial Intelligence*, 1:285–310, 1987.
- [188] D. C. Petriu and H. Shen. Applying the UML Performance Profile: Graph Grammar-based Derivation of LQN Models from UML Specifications. In *Proceedings of the 12th Int. Conference on Computer Performance Evaluation: Modelling Techniques and Tools (TOOLS 2002)*, volume 2324 of *Lecture Notes in Computer Science*, pages 159–177, London, UK, 2002. Springer Berlin.
- [189] A. Poggi, M. Tomaiuolo, and P. Turci. An agent-based service oriented architecture. In *Proc. 8th AI* IA/TABOO Joint Workshop From Objects to Agents: Agents and Industry: Technological Applications of Software Agents, Genova*, pages 157–165, 2007.
- [190] R. Poler, F. Lario, and G. Doumeingts. Dynamic modelling of decision systems (DMDS). *Computers in Industry*, 49(2):175–193, October 2002. ISSN 0166-3615. doi: 10.1016/S0166-3615(02)00083-0. URL <http://www.sciencedirect.com/science/article/pii/S0166361502000830>.
- [191] J. T. Pollock. The big issue: Interoperability vs integration. *eAI Journal*, October 2001. URL http://me.jtpollock.us/pubs/2001.08-BigIssue_eAIJournal.pdf.
- [192] L. Ribeiro, J. Barata, and P. Mendes. MAS and SOA: complementary automation paradigms. In A. Azevedo, editor, *Innovation in Manufacturing Networks*, volume 266 of *IFIP International Federation for Information Processing*, pages 259–268. Springer Boston, 2008. ISBN 978-0-387-09491-5.
- [193] L. M. Rose, D. S. Kolovos, R. F. Paige, and F. A. C. Polack. Model migration with Epsilon Flock. In D. Hutchison, T. Kanade, J. Kittler, et al., editors, *Theory and Practice of Model Transformations*, volume 6142, pages 184–198. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-13687-0.
- [194] J. Rothenberg. The nature of modeling. *Artificial Intelligence, Simulation, and Modeling*, pages 75–92, 1989.

- [195] M. Rother and J. Shook. *Learning to See: Value Stream Mapping to Add Value and Eliminate MUDA*. Lean Enterprise Institute, June 1999. ISBN 978-0966784305.
- [196] K. Ryu and M. Jung. Agent-based fractal architecture and modelling for developing distributed manufacturing systems. *International Journal of Production Research*, 41(17):4233–4255, 2003. ISSN 0020-7543. doi: 10.1080/0020754031000149275.
- [197] N. M. Sadeh, D. W. Hildum, and D. Kjenstad. Agent-based e-supply chain decision support. *Journal of Organizational Computing and Electronic Commerce*, 13(3-4): 225–241, 2003.
- [198] D. Saff. Theory-infected: or how I learned to stop worrying and love universal quantification. In *Companion to the 22nd ACM SIGPLAN Conf. on Object-oriented Programming Systems and Applications*, OOPSLA '07, pages 846–847, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-865-7. doi: 10.1145/1297846.1297919.
- [199] M. Sánchez Carrilero, M. Marcos Bárcena, M. Álvarez Alcón, J. Sánchez Sola, and R. Bienvenido Bárcena. El diseño en los sistemas de fabricación holónicos. In *Actas del X Congreso Internacional de Ingeniería Gráfica*, pages 312–330, 1998.
- [200] M. Sánchez Carrilero, F. Aguayo González, J. Lama Ruiz, R. Bienvenido Barcena, and M. Marcos Barcena. Integración de modelos biónicos, holónicos y fractales para fabricación distribuida. *Anales de Ingeniería Mecánica*, pages 395–403, 2004.
- [201] SAP News. Microsoft, IBM, SAP to discontinue UDDI web services registry effort, January 2006. URL <http://soa.sys-con.com/node/164624>. Last checked: November 6th, 2013.
- [202] D. C. Schmidt. Model-Driven Engineering. *IEEE Computer*, 39(2):25–31, 2006. ISSN 0018-9162.
- [203] E. Seidewitz. What models mean. *Software, IEEE*, 20(5):26–32, 2003. ISSN 0740-7459. doi: 10.1109/MS.2003.1231147.
- [204] S. Sen, B. Baudry, and J. Mottu. Automatic model generation strategies for model transformation testing. In R. F. Paige, editor, *Theory and Practice of Model Transformations*, volume 5563, pages 148–164. Springer-Verlag, Berlin, Germany, 2009. ISBN 978-3-642-02407-8.
- [205] W. Shen, Q. Hao, H. J. Yoon, and D. H. Norrie. Applications of agent-based systems in intelligent manufacturing: An updated review. *Advanced Engineering Informatics*, 20(4):415–431, October 2006. ISSN 1474-0346.
- [206] G. A. Silver, A. Maduko, J. Rabia, J. Miller, and A. Sheth. Modeling and simulation of quality of service for composite web services. In *Proceedings of 7th World Multiconference on Systemics, Cybernetics and Informatics*, pages 420–425. Int. Institute of Informatics and Systems, November 2003.
- [207] A. Sinha and A. Paradkar. Model-based functional conformance testing of web services operating on persistent data. In *Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications*, pages 17–22, Portland, Maine, 2006. ACM. ISBN 1-59593-458-8. doi: 10.1145/1145718.1145721.

-
- [208] C. U. Smith and L. G. Williams. Software performance engineering. In L. Lavagno, G. Martin, and B. Selic, editors, *UML for Real: Design of Embedded Real-Time Systems*, pages 343–366, The Netherlands, May 2003. Kluwer.
- [209] N. Spanoudakis and P. Moraitis. Using ASEME methodology for model-driven agent systems development. In D. Weyns and M.-P. Gleizes, editors, *Agent Oriented Software Engineering XI*, volume 6788 of *Lecture Notes in Computer Science (LNCS)*, pages 106–127. Springer-Verlag Berlin Heidelberg, 2011.
- [210] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, second edition, December 2008. ISBN 978-0321331885.
- [211] Z. Stojanović. *A Method for Component-Based and Service-Oriented Software Systems Engineering*. PhD thesis, Delft University of Technology, 2005.
- [212] R. Straeten, T. Mens, and S. Baelen. Challenges in Model-Driven software engineering. In M. R. V. Chaudron, editor, *Models in Software Engineering*, volume 5421 of *LNCS*, pages 35–47. Springer-Verlag, Berlin, Germany, 2009. ISBN 978-3-642-01647-9.
- [213] D. Tapia, S. Rodríguez, J. Bajo, and J. Corchado. FUSION@: a SOA-based multi-agent architecture. In J. Corchado, S. Rodríguez, J. Llinas, and J. Molina, editors, *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*, volume 50, pages 99–107. Springer Berlin/Heidelberg, 2009.
- [214] E. Tatara, M. North, C. Hood, F. Teymour, and A. Cinar. Agent-based control of spatially distributed chemical reactor networks. In S. Brueckner, G. Di Marzo Seruendo, D. Hales, and F. Zambonelli, editors, *Engineering Self-Organising Systems*, volume 3910 of *Lecture Notes in Computer Science*, pages 222–231. Springer Berlin / Heidelberg, 2006. ISBN 978-3-540-33342-5.
- [215] A. Tharumarajah, A. Wells, and L. Nemes. Comparison of emerging manufacturing concepts. In *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics*, pages 325–331, California, EEUU, 1998.
- [216] The hsql Development Group. HSQLDB homepage, January 2013. URL <http://hsqldb.org/>. Last checked: November 6th, 2013.
- [217] The Open Group. *The Open Group Architecture Framework (TOGAF) Version 9.1*. The Open Group, 2011. ISBN 978-90-8753-679-4.
- [218] The PostgreSQL Global Development Group. PostgreSQL homepage, November 2013. URL <http://www.postgresql.org/>. Last checked: November 6th, 2013.
- [219] M. Tribastone and S. Gilmore. Automatic extraction of PEPA performance models from UML activity diagrams annotated with the MARTE profile. In *Proceedings of the 7th Int. Workshop on Software and Performance*, pages 67–78, Princeton, NJ, USA, 2008. ACM. doi: 10.1145/1383559.1383569.

- [220] M. M. Tseng, M. Lei, C. Su, and M. E. Merchant. A collaborative control system for mass customization manufacturing. *CIRP Annals - Manufacturing Technology*, 46(1):373–376, 1997. ISSN 0007-8506. doi: 10.1016/S0007-8506(07)60846-4.
- [221] M. Ulieru and M. Cobzaru. Building holonic supply chain management systems: an e-logistics application for the telephone manufacturing industry. *IEEE Transactions on Industrial Informatics*, 1(1):18–30, 2005. ISSN 1551-3203. doi: 10.1109/TII.2005.843827.
- [222] M. Utting, A. Pretschner, and B. Legeard. A taxonomy of model-based testing, April 2006. URL <http://researchcommons.waikato.ac.nz/handle/10289/81>. Last checked: November 6th, 2013.
- [223] J. Vaario and K. Ueda. Biological concept of self-organization for dynamic shop-floor configuration. In N. Okino, T. Hiroyuki, and F. Susumu, editors, *Selected, revised proceedings of the IFIP TC5/WG5.7 International Conference on Advances in Production Management Systems*, volume 114 of *IFIP Conference Proceedings*, pages 55–66, Kyoto, Japan, November 1996. Chapman & Hall. ISBN 0-412-82350-0.
- [224] J. Vara Mesa, E. Marcos, and M. V. de Castro. Obteniendo modelos de sistemas de información a partir de modelos de negocios de alto nivel: un enfoque dirigido por modelos. In *Actas de las IV Jornadas Científico-Técnicas en Servicios Web y SOA*, pages 15–28, Sevilla, España, October 2008.
- [225] J. Warmer, K. Thoms, M. Boger, F. Filipelli, M. Bauer, and J. Reichert. Spray project homepage, 2012. URL <https://code.google.com/a/eclipselabs.org/p/spray/>. Last checked: November 6th, 2013.
- [226] H. Warnecke. *The Fractal Company: A Revolution in Corporate Culture*. Springer-Verlag, August 1997. ISBN 038756537X.
- [227] Web Services Interoperability Organization. Basic profile - version 1.1 (Final), August 2004. URL <http://www.ws-i.org/Profiles/BasicProfile-1.1.html>. Last checked: November 6th, 2013.
- [228] M. Weidlich, G. Decker, A. Großkopf, and M. Weske. BPEL to BPMN: the myth of a straight-forward mapping. In *On the Move to Meaningful Internet Systems: OTM 2008*, volume 5331 of *Lecture Notes in Computer Science*, pages 265–282, Monterrey, Mexico, November 2008. Springer Berlin. ISBN 978-3-540-88870-3. doi: 10.1007/978-3-540-88871-0_19.
- [229] T. J. Williams, editor. *A Reference Model for Computer Integrated Manufacturing (CIM)*. Instrument Society of America, North Carolina, USA, second edition, 1989. ISBN 1-55617-225-7.
- [230] T. J. Williams. The Purdue enterprise reference architecture. *Computers in Industry*, 24(2–3):141–158, September 1994. ISSN 0166-3615. doi: 10.1016/0166-3615(94)90017-5.

-
- [231] M. Woodside, G. Franks, and D. Petriu. The future of software performance engineering. In *Proceedings of Future of Software Engineering 2007*, pages 171–187, Los Alamitos, CA, USA, 2007. IEEE Computer Society. doi: 10.1109/FOSE.2007.32.
- [232] Workflow Management Coalition. WFMC-TC-1011: terminology and glossary 3.0, February 1999. URL http://www.workflowpatterns.com/documentation/documents/TC-1011_term_glossary_v3.pdf. Last checked: November 6th, 2013.
- [233] World Batch Forum. Business to manufacturing markup language (B2MML), 2008. URL http://www.isa.org/Content/NavigationMenu/General_Information/Partners_and_Affiliates/WBF/Working_Groups2/XML_Working_Group/B2MML/B2MML.htm. Last checked: November 6th, 2013.
- [234] World Wide Web Consortium. XML Schema Part 0: Primer (Second Edition). Technical report, November 2004. URL <http://www.w3.org/TR/xmlschema-0/>. Last checked: November 6th, 2013.
- [235] World Wide Web Consortium. Web services architecture, February 2004. URL <http://www.w3.org/TR/ws-arch/>. Last checked: November 6th, 2013.
- [236] World Wide Web Consortium. Web services glossary, February 2004. URL <http://www.w3.org/TR/ws-gloss/>. Last checked: November 6th, 2013.
- [237] World Wide Web Consortium. Extensible Markup Language (XML) 1.1 (Second Edition), August 2006. URL <http://www.w3.org/TR/xml11/>. Last checked: November 6th, 2013.
- [238] World Wide Web Consortium. Web Services Description Language (WSDL) Version 2.0 Part 0: Primer, June 2007. URL <http://www.w3.org/TR/wsdl20-primer>. Last checked: November 6th, 2013.
- [239] World Wide Web Consortium. SOAP version 1.2 part 0: Primer, April 2007. URL <http://www.w3.org/TR/soap12-part0/>. Last checked: November 6th, 2013.
- [240] J. A. Zachmann. The Zachman Framework™: the Official Concise Definition, 2008. URL <http://www.zachmaninternational.com/index.php/the-zachman-framework>. Last checked: November 6th, 2013.